31st International Conference on Concurrency Theory

CONCUR 2020, September 1–4, 2020, Vienna, Austria (Virtual Conference)

Edited by

Igor Konnov

Laura Kovács

Editors

Igor Konnov
Informal Systems, Vienna, Austria
igor@informal.systems

Laura Kovács
TU Wien, Austria
lkovacs@forsyte.tuwien.ac.at

ACM Classification 2012
Theory of computation → Concurrency

ISBN 978-3-95977-160-3

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-160-3.

Publication date
August, 2020

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at https://portal.dnb.de.

Digital Object Identifier: 10.4230/LIPIcs.CONCUR.2020.0

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

31st International Conference on Concurrency Theory (CONCUR 2020).
Editors: Igor Konnov and Laura Kovács
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Session Types

Verification

Concurrency

Probabilistic Systems

Games

Algebras and Co-Algebras

Invariant Synthesis

Process Algebras

Vector Addition Systems with States

Markov Decision Processes

Automata

Petri Nets and Counter Machines

Reachability and Decidability

Preface

This proceedings volume contains peer-reviewed contributions accepted at the 31st International Conference on Concurrency Theory (CONCUR), 2020.

The CONCUR conference series brings together researchers, developers, and students in order to advance the theory of concurrency, and promote its applications. Amid the COVID-19 situation, CONCUR 2020 could not take place at TU Wien (Vienna, Austria), as it was originally planned. Instead it was organized as a virtual conference, as part of the umbrella conference QONFEST 2020. In addition to CONCUR 2020, the QONFEST 2020 comprised also the 25th International Conference on Formal Methods for Industrial Critical Systems (FMICS) 2020, the 18th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS) 2020 and the 17th International Conference on Quantitative Evaluation of SysTems (QEST) 2020, alongside with several workshops and tutorials.

Despite the COVID-19 crisis, we have received a high number of submissions. Out of 112 submissions, we have accepted 45 papers for presentation at CONCUR 2020. Given great quality of many submissions, the acceptance bar was quite high. The quality criteria for acceptance were very strict and we thank our program committee and external reviewers for their excellent job in reviewing the CONCUR 2020 submissions. We are especially very grateful to all our reviewers for their efforts in providing high-quality and timely reviews and conducting active discussions on each submission at CONCUR 2020.

We thank the authors of our proceeding's papers for repaying the efforts of our reviewers and submitting their revised works to the CONCUR 2020 proceedings. We hope that details of the papers included in the present proceedings will bring lively discussions within the virtual conference platform of CONCUR 2020, initiating new research directions and collaboration within the CONCUR scientific community and beyond.

In addition to the exceptional papers accepted at CONCUR 2020, the proceedings also include four invited papers accompanying the works presented by our invited speakers at CONCUR 2020. The invited talk by Prof. Annabelle McIver (Macquarie University, Australia) served as a plenary keynote talk of QONFEST 2020, whereas the invited talks by Prof. Roderick Bloem (TU Graz, Austria) and Prof. Thomas A. Henzinger (IST Austria) were shared with other QONFEST 2020 conferences, including FMICS 2020, FORMATS 2020, and QEST 2020. We are also delighted to have had Prof. Catuscia Palamidessi (INRIA Saclay and LIX, France) as our invited speaker discussing recent applications of game theory in the fields of machine learning and privacy, topics that are of core interests for the CONCUR community.

Starting with CONCUR 2020, we are also happy to announce the CONCUR Test-of-Time (ToT) Award series, established by the CONCUR conference and the IFIP 1.8 Working Group on Concurrency Theory. The purpose of this award is to recognize important achievements in Concurrency Theory that were published at CONCUR conferences and have stood the test of time. All papers published in CONCUR between 1990 and 1995 were eligible for the CONCUR ToT Award 2020. The award winners for the CONCUR ToT Awards 2020 have been selected by a Jury composed of Prof. Luca Aceto (Chair), Prof. Jos Baeten, Prof. Patricia Bouyer-Decitre, Prof. Holger Hermanns, and Prof. Alexandra Silva. The results and winners of the CONCUR ToT Award 2020 selection process are described in the invited contribution of Prof. Luca Aceto in these proceedings.

31st International Conference on Concurrency Theory (CONCUR 2020).
Editors: Igor Konnov and Laura Kovács

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We finally would like to thank Interchain Foundation (Switzerland) for its generous sponsorship for running CONCUR 2020. We also gratefully acknowledge the sponsorship of the Vienna Center for Logics and Algorithms - VCLA and the TU Wien, as well as the FORSYTE research group of the Faculty of Informatics of the TU Wien. We finally thank the EasyChair conference management system for assisting us in the reviewing and organization process of CONCUR 2020 together with QONFEST 2020.

As usual, the CONCUR 2020 proceedings are open access thanks to the LIPIcs series. We thank the authors of CONCUR 2020 papers, the CONCUR 2020 participants, as well as the organizers and student volunteers of CONCUR 2020 for making CONCUR 2020 a sucessful virtual event.

Igor Konnov and Laura Kovács
CONCUR 2020 PC Chairs

Committees

CONCUR 2020 Program Committee

Alessandro Abate
Elvira Albert
Giovanni Bacci
Filippo Bonchi
Patricia Bouyer
Yu-Fang Chen
Veronique Cortier
Pedro R. D'Argenio
Michael Emmi
Bernd Finkbeiner
Silvio Ghilardi
Roberto Giacobazzi
Rob van Glabbeek
Alberto Griggio
Ichiro Hasuo
Amir Kafshdar Goharshady
Benjamin Lucien Kaminski
Sophia Knight
Igor Konnov (co-chair)
Laura Kovács (co-chair)
Antonin Kucera
Marijana Lazic
Karoliina Lehtinen
Kuldeep S. Meel
Jan Otop
Joel Ouaknine
Tatjana Petrov
Nir Piterman
Jorge A. Pérez
Giselle Reis
Philipp Ruemmer
Lutz Schröder
Alexandra Silva
Ana Sokolova
Jun Sun
Max Tschaikowski
Valeria Vignudelli
Yakir Vizel
Nobuko Yoshida
Lijun Zhang

CONCUR Steering Committee

Javier Esparza
Pedro D'Argenio
Wan Fokkink
Joost-Pieter Katoen
Catuscia Palamidessi
Davide Sangiorgi
Jiri Srba

QONFEST 2020 General Chair

Ezio Bartocci

QONFEST 2020 Workshop Chair

Florian Zuleger

31st International Conference on Concurrency Theory (CONCUR 2020).
Editors: Igor Konnov and Laura Kovács

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Yehia Abd Alrahman
S. Akshay
Shaull Almagor
Robert Atkey
Clément Aubert
Giorgio Bacci
A.R. Balasubramanian
Paolo Baldan
Suguman Bansal
Kevin Batz
Bartosz Bednarczyk
Giovanni Bernardi
Edwin Brady
Tomas Brazdil
Tomasz Brengos
Franck van Breugel
Roberto Bruni
Michaël Cadilhac
Georgiana Caltais
Marco Carbone
Luca Cardelli
Simon Castellan
Valentina Castiglioni
Pablo Castro
Nathalie Cauchi
Thomas Chatain
Liang-Ting Chen
Liqian Chen
Vincenzo Ciancia
Florence Clerc
Norine Coenen
Jesús Correas Fernández
Emanuele D'Osualdo
Fredrik Dahlqvist
Ugo Dal Lago
Silvano Dal Zilio
Antoine Dallon
Oscar Darwin
Laure Daviaud
Jan de Muijnck-Hughes
Giulia De Santis
Ugo De' Liguoro
Joanna Delicaris
Giorgio Delzanno
Ramiro Demasi
Volker Diekert
Cezara Dragoi
Jérémy Dubut
Jeffrey M. Dudek
Constantin Enea
Zafer Esen
Wan Fokkink
Pierre Fraigniaud
Adrian Francalanza
Robert Ganian
Pierre Ganty
Francesco Gavazzo
Simon Gay
Luca Geatti
Samir Genaim
Silvia Ghilezan
Saverio Giallorenzo
Marco Giunti
Amir Goharshady
Sergey Goncharov
Pablo Gordillo
Tao Gu
Christoph Haase
Christopher Hahn
Ernst Moritz Hahn
Matej Hajnal
Lewis Hammond
Marcel Hark
Jesko Hecking-Harbusch
Gerco van Heerdt
Léo Henry
Jana Hofmann
Ross Horne
Mehran Hosseini
Raymond Hu
Khmelnitsky Igor
Keigo Imai
Miguel Isabel
Shachar Itzhaky
Swen Jacobs
Peter Gjøl Jensen
Martin Jonas
Sebastian Junges
Deepak Kapur
Edon Kelmendi
Ayrat Khalimov
Wen Kokke
Dimitrios Kouzapas
Manfred Kufleitner
Clemens Kupke
Stepan Kuznetsov
Martin Kölbl
James Laird
Ivan Lanese
Martin Lange
Sławomir Lasota
Engel Lefaucheux
Jérôme Leroux
Shang-Wei Lin
Depeng Liu
Gavin Lowe
Lorenzo Luperi Baglini
Enrico Magnago
Anirban Majumdar
Radu Mardare
Sonia Marin
Enrique Martin-Martin
Christoph Matheja
Bastien Maubert
Alicia Merayo
Claudio Antares Mezzina
Jakub Michaliszyn
Joshua Moerman
Fabio Mogavero
Sahar Mohajerani
J. Garrett Morris
Sergio Mover
Renato Neves
Petr Novotný
Federico Olmedo
Jovanka Pantovic
Noemi Passing
Loïc Paulevé
Andrea Peruffo

31st International Conference on Concurrency Theory (CONCUR 2020).
Editors: Igor Konnov and Laura Kovács
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Daniela Petrisan
Frank Pfenning
Robin Piedeleu
Amaury Pouly
M. Praveen
Marcin Przybyłko
Francesco Ranzato
Vojtech Rehak
Fernando Rosa-Velardo
Jurriaan Rot
Arnaud Sangnier
Ocan Sankur
Alessio Santamaria
Luigi Santocanale
Gabriel Santos
Alceste Scalas
Malte Schledjewski
Maximilian Schwenger
Aritra Sengupta
Ilya Sergey
Fausto Spoto
Dominic Steinhöfel
Albin Stjerna
Ilina Stoilkovska
Kohei Suenaga
Vasco T. Vasconcelos
Peter Thiemann
Ramanathan Thinniyam Srinivasan
Chun Tian
Simone Tini
Stefano Tognazzi
Bernardo Toninho
Stavros Tripakis
Andrea Turrini
Daniele Varacca
Dominik Velan
Jana Wagemaker
Chana Weil-Kennedy
Maximilian Weininger
Bas Westerbaan
Markus Whiteland
Josef Widder
Viraj Brian Wijesuriya
Paul Wild
Piotr Witkowski
Thorsten Wißmann
Zhilin Wu
Hongwei Xi
Pengfei Yang
Di-De Yen
Shoji Yuen
Roberto Zunino
Johannes Åman Pohjola

On Privacy and Accuracy in Data Releases

Mário S. Alvim
Computer Science Department, Universidade Federal de Minas Gerais (UFMG), Belo Horizonte, Brasil

Natasha Fernandes
Department of Computing, Macquarie University, Sydney, Australia

Annabelle McIver
Department of Computing, Macquarie University, Sydney, Australia

Gabriel H. Nunes
Computer Science Department, Universidade Federal de Minas Gerais (UFMG), Belo Horizonte, Brasil

Abstract

In this paper we study the relationship between privacy and accuracy in the context of correlated datasets. We use a model of quantitative information flow to describe the the trade-off between privacy of individuals' data and and the utility of queries to that data by modelling the effectiveness of adversaries attempting to make inferences after a data release.

We show that, where correlations exist in datasets, it is not possible to implement optimal noise-adding mechanisms that give the best possible accuracy or the best possible privacy in all situations. Finally we illustrate the trade-off between accuracy and privacy for local and oblivious differentially private mechanisms in terms of inference attacks on medium-scale datasets.

2012 ACM Subject Classification Security and privacy

Keywords and phrases Privacy/utility trade-off, Quantitative Information Flow, inference attacks

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.1

Category Invited Paper

Funding *Mário S. Alvim*: Supported by CNPq, CAPES, and FAPEMIG.
Gabriel H. Nunes: Supported by CNPq, CAPES, and FAPEMIG.

1 Introduction

Consider the following stories about actual and potential privacy breaches.

(i) *In 2009 a lawsuit (Doe v. Netflix)* was filed against Netflix alleging that it had violated fair-trade laws and a federal privacy law. The complainants argued that Netflix's "anonymisation" still allowed individuals to be identified (and indeed plenty were) by combining movie preferences with other generally available data. And that Netflix should have known about these risks.
[Source: `https://www.wired.com/2009/12/netflix-privacy-lawsuit/`]

(ii) *Also in 2009, researchers in Natural Language Processing* showed that using powerful machine learning algorithms, authors of anonymously-penned documents could be identified with a high degree of accuracy.
[Source: On the Feasibility of Internet-Scale Author Identification [15]]

(iii) Finally *Privacy researchers at the University of Melbourne* noted that in any "anonymised datasets" records that can be characterised uniquely put the individuals who contributed thoese records at risk. "While uniqueness does not imply re-identification,

particular data that is known to be held by certain parties, does imply the opportunity for re-identification". [Source: `https://pursuit.unimelb.edu.au/articles/the-simple-process-of-re-identifying-patients-in-public-health-records`]

These three real examples share a common theme: the breaches, or the potential for breaches, are all related to "unintended inferences," meaning that the data published was deemed to be innocuous but turned out to have the potential for inferring information that individuals could claim as infringements to their privacy.

Modern approaches to privacy recognise that inferences are a problem and, whilst they likely cannot be eliminated, can be mitigated by providing "plausible deniability" for individuals. Differential privacy in particular does not rule out inferences, but rather provides guarantees couched in terms of *indistinguishability* – an individual can plausibly deny that their personal information contributed to some precise value v because the query to the data that is finally announced (so the argument goes) could have been produced just as plausibly with some other non-related value v'. This feature is formalised by equations of the form:

$$M(v)(Z) \leq M(v')(Z) \times e^{\epsilon d(v,v')} \ , \tag{1}$$

where M is a mechanism that delivers a noisy output, and v, v' are possible instances of input, one that could be related to an individual, and one that does not. The term $M(v)(Z)$ means "the probability that the output of M evaluated at input v is contained in the subset Z". Here $d(v, v')$ is a distance measure defined on inputs and $\epsilon \geq 0$ is a security parameter often described as a "privacy budget".

This powerful protection rests crucially on the particulars of the adversarial setting. However the practical application of differential privacy in machine learning settings (for example) has been shown to be problematic, with the ϵ parameter typically tuned to optimise accuracy (of some utility measure) without a good understanding of the ramifications for privacy [8]. Moreover the scope and variations of differential privacy mechanisms have been tabulated in other work [5] and what emerges is that there is often little clarification around the basic questions: how does some particular version of differential privacy (and its choice of ϵ) affect *both* the potential for unintended inferences to be made of the individuals contributing their data *and* the accuracy of the "useful" data that is released?

In this paper we aim to study these basic questions from the point of view of inferences. We use a model of *quantitative information flow* to analyse the extent to which inferences succeed once data has been released, and to estimate the accuracy of "useful" queries. From this viewpoint we are able to gauge direct risks to individuals when compared to the benefits of those wishing to use the data, as well as some understanding about the setting of the parameter ϵ in terms of the trade-off between privacy risks and accuracy of data releases.

Significant in this analysis is a contribution to the "no free lunch" style theorems of Kifer et al. [11]. We demonstrate that it is impossible to design differentially-private mechanisms that offer minimal risk benefits to all individuals in all situations at the same time as preserving a fixed accuracy threshold. And dually it is impossible to design differentially-private mechanisms that offer optimal accuracy in all scenarios whilst guaranteeing a fixed threshold for plausible deniability for all individuals.

The implication of these impossibility results is that the manner in which noisy outputs are created is crucial, and by making reasonable assumptions about adversaries, as argued by Kifer at al. [11], a better understanding of the relationship between inferences about privacy and accuracy can be obtained. Our final contribution is to compare experimentally the different risks to privacy versus accuracy in a large to medium-sized datasets when differential privacy is used as the noise-adding mechanism.

2 Informal example: does it matter how to randomise?

In this section we illustrate, using a simple scenario, the challenges faced by designers of privacy mechanisms in deciding what and how to randomise.

Suppose there are N participants invited to contribute to a survey. The survey question is of a personal nature and some respondents might be embarrassed if their true response came to be known. However the organisers of the survey only want to know the total number of (say) "yes" replies. Fig. (1) and Fig. (2) are two possible designs for collecting the responses and announcing a count of the total "yes" responses. Fig. (1) is the well-known "randomised response" protocol which was designed to give respondents plausible deniability so that even the data collectors do not know exactly whether the response they receive from any individual is the "true" response [20].

Fig. (2) follows the blueprint of traditional "oblivious" privacy mechanisms. An accurate tally of the true answers of the respondents is computed, and then some randomness is added. In this particular example we use the geometric distribution.

In both cases there is a corresponding differential privacy guarantee. In Fig. (1) the guarantee grants plausible deniability wrt. a $e^{\log 3}$ threshold for each respondent individually. For Fig. (2) the differential-privacy guarantee is not handed down (directly at least) to an individual, but rather gives a guarantee on the plausible deniability for the final output. In this case more work is required to determine the privacy risk to an individual, but it is relatively easy to provide a guarantee of accuracy: that's because the randomisation is added to the *useful* output, namely the true answer to the query and there are strong results which show that good accuracy can be guaranteed using the geometric distribution for this type of data release. In contrast for Fig. (1), as mentioned above, the randomness is added directly to the data that is considered to be private (by the owner of that data), and so in this scenario the survey organisers would want to know the affect on the accuracy of the final count, which also requires more work to compute.

Much of the theoretical analysis of privacy mechanisms is carried out within the context of an adversarial model. In the case of differential privacy that model is deliberately worst-case: namely it is assumed that the adversary knows *all the data* except that of a given individual. Within that setting though, it seems not straightforward to examine vulnerabilities in regard to "unintended inferences" or the potential for such inferences as described by (iii) above. Moreover Chatzikokolakis et al. [10] have shown that such adversarial models offer surprisingly weak guarantees of privacy operating against other reasonable adversarial settings.

In the case of trying to decide whether to use Fig. 1 or Fig. 2 if we use the weak differential privacy adversarial setting, a designer would be unable to determine how randomness might defend against an actual privacy breach i.e. an unintended inference. When focussing on randomness as a defence, relevant issues are not only that respondents have plausible deniability, but also what level of ϵ to use that balances the risk of an unintended inference with a reasonably accurate tally of "yes" respondents, and what might be the adversary's prior knowledge.

In the remaining sections we analyse the potential for inferences using a model of Quantitative Information Flow (QIF), and adversarial settings which include assumptions about an adversary's prior knowledge. We begin with a brief summary if QIF in the next section.

```
// Assume resp is an array of length N set to
// participants' responses, to a survey question.
i := 0;
count:= 0;
while (i<N) {
        coin:= 0 [1/2] 1;        // Random response
        count:= (count + coin
                [1/2]            // Randomly include or not
                count + resp[i]);
        i++;
}
Print count;     // Announce the approximate count
```

Assume that all variables cannot be observed by an adversary except for the final "Print" of the count. On each iteration, the participant i is randomly selected for inclusion in the count or not. In the case that the participant's true response $resp[i]$ is not included, a random response *coin* for that participant is delivered instead.
This mechanism $\mathcal{R}$ is able to guarantee the following differential-privacy constraint for each participant i providing plausible deniability for their true response:

$$\mathcal{R}(\texttt{resp[i]=0})(Z) \leq \mathcal{R}(\texttt{resp[i]=1})(Z)e^{\log 3} \ .$$

Figure 1 Randomised response with `N` participants.

```
// Assume resp is an array of length N set to
// participants' responses, to a survey question
// epsilon is a parameter for randomising the final tally.
i := 0;
tally:= 0;
while (i<N) {
        tally:= tally + resp[i];
        i++;
}
count:= Geom(tally,  epsilon);
Print count;     // Announce the approximate count
```

Assume that all variables cannot be observed by an adversary except for the final "Print" of the count. On each iteration, the participant i's response $resp[i]$ is included in the count. After the full count has been tallied, the result is randomised according to the (truncated) Geometric distribution. This mechanism $\mathcal{G}$ is able to guarantee the following differential-privacy constraint for the value of the tally, for $\epsilon = \log 3$:

$$\mathcal{G}(\texttt{tally} = k)(Z) \leq \mathcal{G}(\texttt{tally} = k{+}1)(Z)e^{\log 3} \ .$$

Figure 2 Oblivious response mechanism to announce the total for `N` participants.

3 Review of Quantitative Information Flow

The fundamental notion in the analysis of information flow is a *secret* which is a value that is unknown to an adversary, or at least about which there is uncertainty as to its precise value. The relation between secrets and privacy is that any information designated as "sensitive" means that it should be kept secret, meaning that its precise value should remain unknown, or uncertain from the point of view of the adversary. When a privacy mechanism releases information from a data set containing sensitive and insensitive information, of course some sensitive (or "secret") information is likely to be released as well. If the uncertainty about the secret's value is reduced sufficiently, then we might say that the privacy has been breached. This is the essence of an inference attack: if the uncertainty about a secret's value is reduced sufficiently then the adversary is able to predict the true values of secrets with high likelihood.

Quantitative Information Flow makes these intuitions mathematically precise. Given a range of possible secret values of (finite) type $\mathcal{X}$, we model a secret as a probability distribution of type $\mathbb{D}\mathcal{X}$, because it ascribes "probabilistic uncertainty" to the secret's exact value. Given $\pi\colon \mathbb{D}\mathcal{X}$, we write π_x for the probability that π assigns to $x\colon \mathcal{X}$, with the idea that the more likely it is that the real value is some specific x, then the closer π_x will be to 1. Normally the uniform distribution over $\mathcal{X}$ models a secret which could equally take any one of the possible values drawn from its type and we might say that, beyond the existence of the secret, nothing else is known. In any case, once we have a secret, we are interested in analysing whether a mechanism that uses it might leak some information about it. To do this we define a measure for uncertainty, and use it to compare the uncertainty of the secret before and after executing the algorithm. If we find that the two measurements differ then we can say that there has been an information leak.

The original QIF analyses of information leaks in computer systems [3, 4] used Shannon entropy [18] to measure uncertainty because it captures the idea that more uncertainty implies "more secrecy", and indeed the uniform distribution corresponds to maximum Shannon entropy (corresponding to maximum "Shannon uncertainty"). More recent treatments have shown that Shannon entropy is not the best way to measure uncertainty in security contexts because it does not model scenarios relevant to the goals of the adversary. In particular there are some circumstances where a Shannon analysis actually gives a more favourable assessment of security than is actually warranted if the adversary's motivation is taken into account [19].

Alvim et al. [2] proposed a more general notion of uncertainty based on "gain functions". This is the notion we will use. A *gain function* measures a secret's uncertainty according to how it affects an adversary's actions within a given scenario. We write $\mathcal{W}$ for a (usually finite) set of actions available to an adversary corresponding to an "attack scenario" where the adversary tries to infer something (e.g. some property) about the secret. For a given secret $x\colon \mathcal{X}$, an adversary's choice of $w\colon \mathcal{W}$ results in the adversary gaining something beneficial to his objective. This gain can vary depending on the adversary's choice (w) and the exact value of the secret (x). The more effective is the adversary's choice in how to act, the more he is able to overcome any uncertainty concerning the secret's value, and increase his gain.

▶ **Definition 1.** *Given a type $\mathcal{X}$ of secrets, a gain function $g\colon \mathcal{W}\times\mathcal{X} \to \mathbb{R}$ is a real-valued function such that $g(w, x)$ determines the gain to an adversary if he chooses w and the secret is x.*

A simple example of a gain function is bv, where $\mathcal{W} := \mathcal{X}$, and

$$\mathsf{bv}(x, x') \quad := \quad 1 \;\; \textit{if} \;\; x = x' \;\; \textit{else} \;\; 0\,. \tag{2}$$

For this scenario, the adversary's goal is to determine the exact value of the secret, so he receives a gain of 1 if he correctly guesses the value of a secret, and zero otherwise. Assuming that he knows the range of possible secrets $\mathcal{X}$, he therefore has $\mathcal{W}{:=}\mathcal{X}$ for his set of possible guesses.

Given a gain function we define the *vulnerability* of a secret in $\mathbb{D}\mathcal{X}$ relative to the scenario it describes: it is the maximum average gain to an adversary.

▶ **Definition 2.** *Let $g{:}\,\mathcal{W}{\times}\mathcal{X}{\to}\mathbb{R}$ be a gain function, and $\pi{:}\,\mathbb{D}\mathcal{X}$ be a secret. The* vulnerability *$V_g[\pi]$ of the secret wrt. g is:*

$$V_g[\pi] \quad := \quad \max_{w\in\mathcal{W}} \sum_{x\in\mathcal{X}} g(w,x){\times}\pi_x \ .$$

For a secret $\pi{:}\,\mathbb{D}\mathcal{X}$, the vulnerability wrt. bv is $V_{\mathsf{bv}}[\pi]{:=}\max_{x:\,\mathcal{X}}\pi_x$, i.e. the maximum probability assigned by π to possible values of x. The adversary's best strategy for optimising his gain would therefore be to choose the value x that corresponds to the maximum probability under π. This vulnerability V_{bv} is called *Bayes Vulnerability*.

A *mechanism* is an algorithm that inputs a secret and outputs some observable, which could be determined by the value of the secret. An example of a privacy mechanism could be an query to a database which outputs numbers of residents living in various regions or counties. We define $\mathcal{Y}$ to be the type for observables. The model of a mechanism now assigns a probability that $y{:}\,\mathcal{Y}$ can be observed given that the secret is x. Such observables could be sample timings in a timing analysis in cryptography, for example.

▶ **Definition 3.** *A* mechanism *is a stochastic channel*[1] *$C{:}\,\mathcal{X}{\times}\mathcal{Y}\to[0,1]$. The value C_{xy} is the probability that y is observed given that the secret is x.*

Given a (prior) secret $\pi{:}\,\mathbb{D}\mathcal{X}$ and mechanism C we write $\pi{\triangleright}C$ for the joint distribution *in $\mathcal{X}{\times}\mathcal{Y}$ defined*

$$(\pi{\triangleright}C)_{xy} \quad := \quad \pi_x{\times}C_{xy} \ .$$

For each $y{:}\,\mathcal{Y}$, the marginal probability *that y is observed is $p_y{:=}\sum_{x:\,\mathcal{X}}(\pi{\triangleright}C)_{xy}$. For each observable y, the corresponding* posterior probability *of the secret is the conditional $\pi|^y{:}\,\mathbb{D}\mathcal{X}$ defined $(\pi|^y)_x{:=}(\pi{\triangleright}C)_{xy}/p_y$.* [2]

Intuitively, given a prior secret $\pi\in\mathbb{D}\mathcal{X}$ and mechanism C, the entry $\pi_x{\times}C_{xy}$ of the joint distribution $\pi{\triangleright}C$ is the probability that the actual secret value is x and the observation is y. This joint distribution contains two pieces of information: the probability p_y of observing y and the corresponding posterior $\pi|^y$ which represents the adversary's updated view about the uncertainty of the secret's value. If the vulnerability of the posterior increases, then information about the secret has leaked and the adversary can use it to increase his gain by changing how he chooses to act. The adversary's average overall gain, taking the observations into account, is defined to be the average posterior vulnerability (i.e. the average gain of each posterior distribution, weighted according to their respective marginals):

$$V_g[\pi{\triangleright}C] := \sum_{y\in\mathcal{Y}} p_y{\times}V_g[\pi|^y] \ , \qquad \textit{where } p_y,\ \pi|^y \textit{ are defined at Def. 3.} \tag{3}$$

---

[1] Stochastic means that the rows sum to 1, i.e. $\sum_{y\in\mathcal{Y}} C_{xy}=1$.

[2] We assume for convenience that when we write p_y the terms C, π and y are understood from the context. Notation suited for formal calculation would need to incorporate C and π explicitly.

Now that we have Def. 2 and Def. 3 we can start to investigate whether the information leaked through observations $\mathcal{Y}$ actually have an impact in terms of whether it is useful to an adversary. It is easy to see that for any gain function g, prior π and mechanism C we have that $V_g[\pi] \leq V_g[\pi\rangle C]$. In fact the greater the difference between the prior and posterior vulnerability, the more the adversary is able to use the leaked information within the scenario defined by g.

We can also compare the information leak properties of mechanisms – if one mechanism C is more vulnerable than another D under all priors and gain functions, then we say that D is *more secure* than C.

▶ **Definition 4.** *Given C, D mechanisms we say that C is refined by D, or $C \sqsubseteq D$ if for all priors $\pi \in \mathbb{D}\mathcal{X}$ and gain functions $g\colon \mathcal{W}\times\mathcal{X} \to \mathbb{R}$ that $V_g[\pi\triangleright C] \geq V_g[\pi\triangleright D]$.*

Def. 4 is a very robust ordering as it applies to all scenarios. It also characterises post-processing of mechanisms. A post-processing step describes how outputs can be reassigned, or merged. If $C\colon \mathcal{X}\times\mathcal{Y} \to [0,1]$ that outputs values of type $\mathcal{Y}$, then we can "remap" them to outputs of type $\mathcal{Z}$ using a mechanism $D\colon \mathcal{Y}\times\mathcal{Z} \to [0,1]$. The result is therefore the matrix multiplication $C{\cdot}D$. Not surprisingly it can be shown that $C \sqsubseteq C{\cdot}D$, but perhaps surprisingly if $C \sqsubseteq D$ then there is some postprocessing mechanism E such that $C{\cdot}E = D$. We use the following facts about refinement [14].

F1 The maximal element in the refinement ordering is the *unit* channel $\mathbb{I}$ that releases no information at all. It satisfies $V_g[\pi\triangleright\mathbb{I}] = V_g[\pi]$, for all gain functions g.

F2 If $C \not\sqsubseteq D$ then there is some gain function such that $V_g[\pi\triangleright C] < V_g[\pi\triangleright D]$.

F3 QIF enables reasoning about correlated data as follows. Let $C\colon \mathcal{X}\times\mathcal{Y} \to [0,1]$, and let $\Pi \in \mathbb{D}(\mathcal{Z}\times\mathcal{X})$ be a joint distribution representing a correlation between $\mathcal{Z}$. We can define a channel $C^\star\colon \mathcal{Z}\times\mathcal{X}\times\mathcal{Y} \to [0,1]$ as $C^\star_{zxy} = C_{xy}$. Notice that $C^\star$ simply repeats rows of C; moreover $C^\star$ now allows us to investigate how much we can infer about $\mathcal{Z}$ from information flows about $\mathcal{X}$ through C.

F4 If $D \not\sqsubseteq C$ then there is some correlation $\Pi \in \mathbb{D}(\mathcal{Z}\times\mathcal{X})$ such that $V_{\mathsf{bv'}}[\pi\triangleright C] > V_{\mathsf{bv'}}[\pi\triangleright D]$, where bv' defines Bayes' Vulnerability for $\mathcal{Z}$.

3.1 Modelling inferences

An inference is commonly regarded as the ability to determine a property about an individual or a group of individuals using any available information. Thus the inferred property might not be obvious, which is why almost all data releases are vulnerable, to some extent, from inference attacks. In this section we show how to use a QIF model to study such vulnerabilities.

As we have noted above we can use a gain function to model an adversary trying to guess some value within a scenario defined prior knowledge $\pi \in \mathbb{D}\mathcal{X}$. We generalise the idea of Bayes vulnerability Def. 1, modelling an adversary trying to guess a specific value, to an adversary trying to guess a property (or set of values).

▶ **Definition 5.** *Given a state space $\mathcal{X}$ let $\mathcal{P}{:=}\{p_1, p_2 \ldots p_n\}$ be a partition of $\mathcal{X}$. Define gain function $\overline{\mathcal{P}} : \mathcal{P}\times\mathcal{X} \to \{0,1\}$:*

$$\overline{\mathcal{P}}(p,x) \;:=\; 1 \text{ iff } x \in p \text{ else } 0\ .$$

Suppose now that M is a mechanism operating within a scenario where the adversary's prior knowledge is π. We can determine the adversary's ability to use the information in the data release to infer a property defined by a partition $\mathcal{P}$: it is $V_{\overline{\mathcal{P}}}[\pi\triangleright M]$.

It turns out that we can use this idea to analyse the effectiveness of a mechanism in terms of both privacy *and* accuracy. To see how this works recall the mechanisms in Fig. 1 and Fig. 2. The *privacy aspect* is to prevent observers from inferring true responses, whereas the *accuracy* is to deliver a result from which observers can infer with a high level of confidence the true count. [3]

To analyse how well a participant's true response can be inferred from the data release we first set the state space $\mathcal{X}$ to be the set of total possible responses, and choose a prior distribution $\pi \in \mathbb{D}\mathcal{X}$. Later in our experiments §5 we describe how to choose a prior to capture reasonable prior knowledge of the adversary. Writing $\langle r_1, r_2\rangle$ for a scenario where participant 1's true response is $r_1 \in \{0,1\}$ and participant 2's true response is $r_2 \in \{0,1\}$ then $\mathcal{X}{:=}\{\langle 0,0\rangle, \langle 0,1\rangle, \langle 1,0\rangle, \langle 1,1\rangle\}$. Let $\mathcal{S}$ be the partition $\{\{\langle 0,0\rangle, \langle 0,1\rangle\}, \{\langle 1,0\rangle, \langle 1,1\rangle\}\}$ – notice that it contains two sets, with the first grouping the scenarios where participant 1's true response is 0, and the second where participant 1's true response is 1. Computing $V_{\overline{\mathcal{S}}}[\pi \triangleright M]$ gives the probability that the adversary can infer participant 1's true response, whatever it turns out to be. For M to defend strongly against unintended inferences about an individual participant we would like this to be as close to $V_{\overline{\mathcal{S}}}[\pi]$ as possible, because then the information delivered by the data release cannot be used by any adversary in his attack. In fact in our later experiments §5, we define privacy loss to be the ratio $V_{\overline{\mathcal{S}}}[\pi \triangleright M]/V_{\overline{\mathcal{S}}}[\pi]$.

On the other hand in both mechanisms the output count is randomised, even though delivering an accurate count is the purpose of the data release. We can therefore gauge the accuracy of the mechanism by calculating the probability with which the observer can infer the true count for the same prior π. In detail, let $\mathcal{U}$ be the partition $\{\{\langle 0,0\rangle\}, \{\langle 0,1\rangle, \langle 1,0\rangle\}, \{\langle 1,1\rangle\}\}$. Here there are three subsets – the first is the (only) case in which the true count is 0, the second contains two instances where the true count is 1 and the third is the unique case where the count is 2. Computing $V_{\overline{\mathcal{U}}}[\pi \triangleright M]$ therefore gives the probability that the observer (or adversary) can infer the true count. Clearly we would like this to be close to 1 for good accuracy.

We can see these ideas working out for our two examples above by constructing the information flow channels for each of them.

Consider first the channel associated with randomised response Fig. 1.

$$\texttt{randomresp}{:=} \quad \begin{array}{c} \\ \langle 0,0\rangle \\ \langle 1,0\rangle \\ \langle 0,1\rangle \\ \langle 1,1\rangle \end{array} \begin{array}{c} \begin{array}{ccc} \texttt{0} & \texttt{1} & \texttt{2} \end{array} \\ \begin{pmatrix} 0.56 & 0.38 & 0.06 \\ 0.19 & 0.62 & 0.19 \\ 0.19 & 0.62 & 0.19 \\ 0.06 & 0.38 & 0.56 \end{pmatrix} \end{array} \tag{4}$$

Each row of the matrix corresponds to the probabilities of observing the output count given by $0, 1$ or 2 when executed within a scenario defined by the participants' true responses. For example if both participants' responses are 0 (first row corresponding to $\langle 0,0\rangle$) the observer would see the output at 0 with probability 0.56 because 0 is output when both respondents' true values are counted and, when they are not, the random value that *is* counted is still zero. For each participant, his true value is counted with probability $1/2$, but also with probability $1/2$ he randomly picks to respond with zero anyway. Thus each participant contributes 0 to the final survey count with probability 0.75, and since participant responses are independent of each other the count reported will be 0 with probability $0.75{\times}0.75 \sim 0.56$.

[3] Thus in this scenario the adversary and observer are the same.

The other probabilities are computed similarly. Assuming then a uniform prior distribution over possible scenarios, we see that the probability of inferring participant 1's true response using the information in the data release is

$$V_{\overline{\mathcal{S}}}[\pi \triangleright \texttt{randomresp}] = 0.63 \ ,$$

which is only a little more than the prior $V_{\overline{\mathcal{S}}}[\pi] = 0.5$. On the other hand the accuracy of inferring the sum of the responses is *worse*:

$$V_{\overline{\mathcal{U}}}[\pi \triangleright \texttt{randomresp}] = 0.59 \ ,$$

signifying that the observer cannot have a great deal of confidence in inferring the true tally.

We can perform the same analysis on Fig. 2 for comparison. Here the channel matrix is:

$$\texttt{obliviousresp} := \begin{array}{c} \\ \langle 0,0\rangle \\ \langle 1,0\rangle \\ \langle 0,1\rangle \\ \langle 1,1\rangle \end{array} \begin{array}{c} \begin{array}{ccc} \texttt{0} & \texttt{1} & \texttt{2} \end{array} \\ \begin{pmatrix} 0.75 & 0.1667 & 0.0833 \\ 0.25 & 0.5 & 0.25 \\ 0.25 & 0.5 & 0.25 \\ 0.0833 & 0.1667 & 0.75 \end{pmatrix} \end{array} \tag{5}$$

Notice that the (truncated) geometric mechanism is used to compute the probabilities. In the first row, corresponding to scenario $\langle 0,0\rangle$, the true tally is 0 which is then reported accurately with probability 0.75. As above we an compute the probabilities of inferring participant 1's true response and the true tally as follows:

$$V_{\overline{\mathcal{S}}}[\pi \triangleright \texttt{obliviousresp}] = 0.67 \quad \textit{and} \quad V_{\overline{\mathcal{U}}}[\pi \triangleright \texttt{obliviousresp}] = 0.625 \ .$$

Notice that whilst the accuracy has improved from 0.59 to 0.625, this has come at a cost of increasing the probability of inferring participant 1's response from 0.63 to 0.67. The reason that the increase in accuracy incurs a decrease in privacy is because the two properties are related: if the ability to infer the participants' responses increases then the ability to infer an accurate tally which depends on those responses must also increase. The extent to which we can have accuracy of utility and privacy depends crucially on how the $\mathcal{S}$ and $\mathcal{U}$ are correlated in the prior.

In fact we can define a distribution $\Pi \in \mathbb{D}(\mathcal{S} \times \mathcal{U})$ which expresses the correlation between the privacy property defined by partition $\mathcal{S}$ and the useful property defined by partition $\mathcal{U}$:

$$\Pi_{su} := \pi(s \cap u) \ .$$

From here we can observe that $\mathcal{S}$ and $\mathcal{U}$ can be highly correlated for some of their partition subsets suggesting that a more accurate inference of one must lead to a more accurate inference of the other. For example if a tally of 0 can be accurately inferred then the probability of subsequently inferring that participant 1's true response is also 0 is $\Pi_{s_0,u_0}/\pi(u_0) = 1$, meaning that a 0 tally implies absolutely that participant 1's true value must be 0. On the other hand, the probability of inferring that his value is 0 if the tally is accurately reported as 1 is $\Pi_{s_0,u_1}/\pi(u_1) = 1/2$. Thus studying the impact of the mechanism in terms of the abstraction of the correlation will enable us to understand the trade off between privacy and accuracy, and the limitations on delivering highly accurate data releases in some scenarios.

3.2 Accuracy versus privacy

In this section we study the trade off between accuracy and privacy in terms of inferences. Rather than working with the raw data precisely we use an abstraction based on a correlation between $\mathcal{S}$ and $\mathcal{U}$ where, as described above, $\mathcal{S}$ is defined by a partition on raw data that specifies some privacy criterion, and $\mathcal{U}$ similarly is defined by a partition that specifies the useful data to be released as accurately as possible. We describe the correlation by a distribution $\mathbb{D}(\mathcal{S}\times\mathcal{U})$. For a stochastic channel in $\mathcal{X}\times\mathcal{Y}\to[0,1]$ representing a mechanism we use notation $\mathcal{X}\mapsto\mathcal{Y}$ to differentiate between the input type $\mathcal{X}$ (denoting the secret) and the output type $\mathcal{Y}$ denoting the observable.

Using these conventions, the next definition gives the effect of a mechanism in the ability for an adversary to infer the component $\mathcal{S}$.

▶ **Definition 6.** *Let $\Pi\colon\mathbb{D}(\mathcal{S}\times\mathcal{U})$ represent a correlation in $\mathcal{S}\times\mathcal{U}$, and let $M\colon(\mathcal{S}\times\mathcal{U})\mapsto\mathcal{Y}$ be a stochastic channel representing a data release. We say that M is susceptible to an inference leak for $\mathcal{S}$ if $V_{\overline{\mathcal{S}}}[\Pi] < V_{\overline{\mathcal{S}}}[\Pi\triangleright M]$.*

M is completely privacy preserving wrt. $\overline{\mathcal{S}}$ if and only if $V_{\overline{\mathcal{S}}}[\Pi] = V_{\overline{\mathcal{S}}}[\Pi\triangleright M]$. We measure the privacy loss by the ratio $V_{\overline{\mathcal{S}}}[\Pi\triangleright M]/V_{\overline{\mathcal{S}}}[\Pi]$.

Next, as described above, we can also define the accuracy of inferring the utility.

▶ **Definition 7.** *Let $\Pi\colon\mathbb{D}(\mathcal{S}\times\mathcal{U})$ represent a correlation in $\mathcal{S}\times\mathcal{U}$, and let $M\colon(\mathcal{S}\times\mathcal{U})\mapsto\mathcal{Y}$ be a stochastic channel representing a data release. M's accuracy for $\mathcal{U}$ is the probability that $\mathcal{U}$ can be inferred, i.e. $V_{\overline{\mathcal{U}}}[\Pi\triangleright M]$.*

M is completely accurate for $\mathcal{U}$ if and only if $V_{\overline{\mathcal{U}}}[\Pi\triangleright M] = 1$.

Now we have these definitions, we can provide a simple proof of the well-known "no free lunch" theorem of Kifer [11], which states that it is not possible to have a single mechanism that is arbitrarily accurate and arbitrarily private for all possible correlated secrets.

▶ **Theorem 8.** *There exists no mechanism M which guarantees* both *arbitrary levels of accuracy* and *privacy for all datasets $\Pi\colon\mathbb{D}(\mathcal{S}\times\mathcal{U})$.*

Proof. Let M be a mechanism acting on a (correlated) data set. Pick $\mathcal{S}=\mathcal{U}$, and partition $\mathcal{P}$ on both $\mathcal{S},\mathcal{U}$ so that the private and the useful property are entirely correlated. For any $\epsilon>0$ we might hope that $V_{\overline{\mathcal{P}}}[\Pi\triangleright M]\geq 1-\epsilon$ for good accuracy, whilst at the same time $V_{\overline{\mathcal{P}}}[\Pi\triangleright M]\leq V_{\overline{\mathcal{P}}}[\Pi]+\epsilon$ for good privacy. However if both constraints hold simultaneously then we deduce that $2\epsilon\geq 1-V_{\overline{\mathcal{P}}}[\Pi]$ for all Π, something that cannot be guaranteed. ◀

Thm. 8 applies to all privacy mechanisms, including differential privacy of course. In the next section we investigate two common implementations for differential privacy – one which uses the randomisation in a way to obtain good accuracy, and the other to obtain good privacy. We investigate the trade-off between accuracy and privacy in both cases.

4 Differential privacy

Alvim et al. [1] show that when a mechanism is modelled as a channel the above definition (1) is equivalent to comparing rows of channels relating to x, x'. In particular (1) applied to a channel M says that M is ϵ-differentially private for x, x' if

$$M_{xy}\leq e^{\epsilon}M_{x'y}\quad\textit{and}\quad M_{x'y}\leq e^{\epsilon}M_{xy} \tag{6}$$

for all $y\in\mathcal{Y}$. Notice that (6) compares two channel rows corresponding to secret values x, x'.

When we think of modelling mechanisms in this way, in the context of correlated data, we see that there are two ways in which we can add the noise. The first is to apply it directly to the $\mathcal{S}$ component, in the style of random response sometimes called "local differential privacy". The second is to the $\mathcal{U}$ component, i.e. the accurate result of the query; this is sometimes referred to as an "oblivious (differentially private) mechanism". As we saw in Fig. (1) which is an example of the first kind, and Fig. (2), an example of the second kind, both have an impact on accuracy and privacy. We define these two approaches to differential privacy more generally and examine their respective trade-offs.

▶ **Definition 9.** *A mechanism $C \in (\mathcal{S}\times\mathcal{U}) \to\!\!\triangleright \mathcal{Y}$ is $\mathcal{U}$-insensitive if there is some $L \in \mathcal{S} \to\!\!\triangleright \mathcal{Y}$ such that $C_{suy} = L_{sy}$ (i.e. $C = L^\star$). (Compare [F3] above.) Dually, C is $\mathcal{S}$-insensitive if there is some $R \in \mathcal{U} \to\!\!\triangleright \mathcal{Y}$ such that $C_{suy} = R_{uy}$ (i.e. $C = R^\star$).*

Such insensitive mechanisms as described in Def. 9 add noise either to the $\mathcal{S}/\mathcal{U}$-component respectively, so that the information flow on the other i.e. $\mathcal{U}/\mathcal{S}$-component is derived from the original correlation between $\mathcal{U}$ and $\mathcal{S}$. In general the use of randomisation can often be tailored to achieve particular probabilistic effects. The use of the Laplace distribution for example in oblivious mechanisms means that good utility can be maintained on $\mathcal{U}$, and the impact on privacy therefore can be investigated through the original correlation. We explore the trade-off between accuracy and privacy in differentially-private mechanisms next.

4.1 Utility-focussed privacy

An *utility-focussed mechanism* for differential privacy adds noise to the result of a query, thereby creating opportunities to tune the randomness for accuracy of that query. The differentially-private guarantee for utility-focussed mechanisms entails a property of indistinguishability only on similar query *results*. Any indistinguishability relating to privacy of any sensitive data is dependent on how it correlates with those query results.

With our model for correlated data we can decompose an oblivious mechanism into one which acts directly on $\mathcal{U}$ independently of $\mathcal{S}$, effectively treating it as the secret, and then reinstalling the correlation with $\mathcal{S}$.

▶ **Definition 10.** *$M \in (\mathcal{S}\times\mathcal{U})\to\!\!\triangleright\mathcal{Y}$ is a* utility-focussed *ϵ-private mechanism if it is ϵ-private on $\mathcal{U}$, and $\mathcal{S}$-insensitive.*

The well-known oblivious mechanisms for differential privacy are examples of utility-focused mechanisms. Whilst the randomisation can be tuned to optimise the accuracy of the query this mode of adding randomness does not tell us about ability of an adversary that has knowledge of the correlation to infer the secret component. Thm. 8 tells us we need to consider that correlation to determine the risk. [4]

A weaker property though is to consider whether there is a distinguished mechanism that protects better than all other (oblivious) mechanisms wrt. inference attacks. We write $\overline{\mathcal{S}}$ ($\overline{\mathcal{U}}$) for the gain function derived from the partitioning $\mathcal{S}$ ($\mathcal{U}$) into its singleton sets.

▶ **Definition 11.** *An ϵ-differentially private mechanism M for datasets $\mathbb{D}(\mathcal{S}\times\mathcal{U})$ is optimal wrt. inference attacks on $\mathcal{S}$, if $V_{\overline{\mathcal{S}}}[\Pi \triangleright M] \leq V_{\overline{\mathcal{S}}}[\Pi \triangleright M']$ for all $\Pi \in \mathbb{D}(\mathcal{S}\times\mathcal{U})$ and all ϵ-differentially private mechanisms M'.*

[4] This problem of some individuals being "outliers" and thus potentially identified even in the release of aggregate data is well known and a worst-case mitigation of this situation is the use of sensitivity analysis in for example Laplacian mechanisms [6]. The result is to make any individual's contribution to a noisy announcements of an aggregate statistic minuscule.

Unfortunately there is no mechanism that is optimal wrt. inference attacks except for the trivial mechanisms that release no information at all.

▶ **Theorem 12.** *There is no non-trivial utility-focused mechanism amongst all ϵ-private mechanisms on $\mathcal{S}\times\mathcal{U}$ that is universally optimal wrt. inference attacks on $\mathcal{S}$.*

Proof. (Sketch) Let M be such a non-trivial ϵ-private mechanism. We show that it cannot be universally optimal wrt. inference attacks on $\mathcal{S}$. By Def. 10 this means that M can be decomposed with ϵ-private component $U \in \mathcal{U} \twoheadrightarrow \mathcal{Y}$ such that $U^\star = M$. Moreover if M is not the trivial mechanism, then there exists some mechanism G such that $U \sqsubset G$ (strict refinement, F1), implying that $G \not\sqsubseteq U$. In this case, by (F4) we have immediately that there is some $\Pi \in \mathbb{D}(\mathcal{S}\times\mathcal{U})$ such that $V_{\overline{\mathcal{S}}}[\Pi \triangleright U^\star] > V_{\overline{\mathcal{S}}}[\Pi \triangleright G^\star]$, as required. ◀

Thm. 12 tells us that if privacy is of utmost concern, the use of a utility-focussed mechanism is really about preserving accuracy of the result of the query, and gives no optimal guarantees regarding whether the actual sensitive data is vulnerable to an inference attack under some other mechanism. We see for example with Fig 2 that whilst this does deliver the most accurate result it is not the "most private" as measured by vulnerability to inferences, amongst non-trivial mechanisms.

4.2 Secrecy-focussed privacy

There are many alternative ways to add randomness in privacy mechanisms rather than to the result of a query. We can model this idea as follows.

▶ **Definition 13.** *$M \in (\mathcal{S}\times\mathcal{U})\twoheadrightarrow\mathcal{Y}$ is a* secrecy-focussed *ϵ-private mechanism if it is ϵ-private on $\mathcal{S}$, and is $\mathcal{U}$-insensitive.*

Locally differential private mechanisms are somewhat in this style and therefore are, in some sense, dual to utility-focussed mechanisms because they first produce a noisy version of the data through a noise-adding mechanism applied to $\mathcal{S}$, rather than to the result of a query. Subsequent queries are then applied to the noisy version of the data, in the style of randomised response Fig.1. Next, we define optimality wrt. utility, and show that secrecy-focussed mechanisms cannot be universally optimal wrt. accuracy of utility.

▶ **Definition 14.** *An ϵ-differentially private mechanism $M \in \mathbb{D}(\mathcal{S}\times\mathcal{U})$ is optimal wrt. accuracy on $\mathcal{U}$, if $V_{\overline{\mathcal{U}}}[\Pi \triangleright M] \geq V_{\overline{\mathcal{U}}}[\Pi \triangleright M']$ for all $\Pi \in \mathbb{D}(\mathcal{S}\times\mathcal{U})$ and all ϵ-differentially private mechanisms $M' \in \mathbb{D}(\mathcal{S}\times\mathcal{U})$.*

▶ **Theorem 15.** *Let $|\mathcal{S}| \geq 3$. There is no secrecy-focussed ϵ-differentially private mechanisms on $\mathbb{D}(\mathcal{S}\times\mathcal{U})$ amongst all ϵ-differentially private mechanisms which is optimal wrt. accuracy on $\mathcal{U}$.*

Proof. (Sketch.) Let M be such a secrecy-focussed mechanism, and let L be such that $L^\star = M$ as per Def. 10. The proof is dual to that of Thm. 12 once we show that there is some ϵ-differentially private L' over $\mathcal{S}$ such that $L \not\sqsubseteq L'$. But this follows from an analysis of optimal ϵ-private mechanisms [10]. ◀

Thm. 15 is interesting because it says that mechanisms that add randomness to enable a direct guarantee to the privacy component, cannot be optimised universally for all utility measures. For example, mechanisms that are designed to be locally-differentially private use post-processing to compute the utility on the noisy data. Post-processing, however

is just refinement. Indeed it can be shown, for example, that Fig.1 is a refinement of a *secrecy-focussed* ϵ-private mechanism but more work is needed to explain the observation that locally private mechanisms often exhibit poor accuracy on medium-sized datasets. We end this section by showing that any refinements of secrecy-focussed mechanisms cannot provide universal accuracy.

▶ **Corollary 16.** *If $M \sqsubseteq M'$ and M is secrecy-focussed as in Thm. 15, then M' is not optimal wrt. accuracy on $\mathcal{U}$.*

Proof. Follows since $V_{\overline{\mathcal{U}}}[\Pi \triangleright M] \geq V_{\overline{\mathcal{U}}}[\Pi \triangleright M']$. ◀

5 Experiments

In this section we present some experiments illustrating the main points presented in previous sections. We consider scenarios in which both a data analyst and an adversary can observe the output of a mechanism that reports a (possibly randomised) count of the number of rows in a dataset that satisfy some property. However, whereas the data analyst wants to infer the real value of the counting query performed on the dataset, the adversary wants only to infer the value of a sensitive attribute in a row just added to the dataset. More precisely, we consider experimental scenarios under the following conditions.

1. There is a dataset D of interest, consisting in a multiset of *rows*, each of which is a tuple defined on a set $\mathcal{A}$ of *attributes*. Each attribute $a{\in}\mathcal{A}$ has domain $\mathit{domain}(a)$, and we denote by $\mathit{rows}(\mathcal{A})$ the set of all possible rows that can be formed from attribute set $\mathcal{A}$, and by $x[a]$ the value assumed by attribute $a{\in}\mathcal{A}$ on a row $x{\in}\mathit{rows}(\mathcal{A})$.
2. A new row $x^\star{\in}\mathit{rows}(\mathcal{A})$ will be added to D (due to, e.g., data from a new individual), yielding an extended dataset denoted (with a slight abuse of notation) by $D{\cup}x^\star$.
3. A *data analyst* wants to learn the result of a counting query $\mathtt{count}_q$ that returns the number of rows in the extended dataset $D{\cup}x^\star$ satisfying a given condition q on a *useful attribute* $a_u{\in}\mathcal{A}$ (e.g., how many rows have attribute *gender* set to *female*). The value $\mathtt{count}_q(D{\cup}x^\star)$ is the *real count* for the counting query performed on the extended dataset.
4. An *adversary* has full knowledge of all rows in the dataset D, but is unsure about the contents of the newly added row $x^\star$. His goal is to learn the value $x^\star[a_s]$ of a *sensitive attribute* $a_s{\in}\mathcal{A}$ for this new row.
5. Both the data analyst and the adversary learn the count on the extended dataset $D{\cup}x^\star$ via a *query mechanism* $M_{\mathtt{count}_q}$ that returns a (possibly randomised) version of the real count $\mathtt{count}_q(D{\cup}x^\star)$. We call the output $M_{\mathtt{count}_q}(D{\cup}x^\star)$ the *reported count*, by the mechanism, for the counting query performed on the extended dataset.
6. Both the data analyst and the adversary know the value of the real count $\mathtt{count}_q(D)$ on the original dataset D, and both have as prior knowledge a distribution $\pi^\star{:}\mathbb{D}(\mathit{rows}(\mathcal{A}))$ on all values that the new added row $x^\star$ can assume (e.g., the adversary and the data analyst may be the same entity). From that, they can derive, in the usual way, distributions on the value $x^\star[a_s]$ of the sensitive value of the newly added individual and on the real count $\mathtt{count}_q(D{\cup}x^\star)$ for the query.

To properly formalize the privacy loss and utility of such scenarios, we introduce the following notation. Given a scenario Γ as described above, let Pr^Γ denote the corresponding joint probability distribution –depending on the coin tosses of the distribution $\pi^\star$ on the values for the new row $x^\star$ and on the query mechanism $M_{\mathtt{count}_q}$ employed–, s.t. $Pr^\Gamma(x^\star{=}x, x^\star[a_s]{=}s, \mathtt{count}_q(D{\cup}x^\star){=}u, M_{\mathtt{count}_q}(D{\cup}x^\star){=}u')$ is the probability that in scenario Γ: (i) the

new added row $x^\star$ assumes value $x \in rows(\mathcal{A})$; (ii) the sensitive value $x^\star[a_s]$ of the new added row $x^\star$ assumes value $s \in domain(a_s)$; (iii) the real count of query $\texttt{count}_q$ performed on the extended dataset $D \cup x^\star$ assumes value $u \in \mathbb{N}$; and (iv) the reported count of query $\texttt{count}_q$ produced by the mechanism $M_{\texttt{count}_q}$, w.r.t. the extended dataset $D \cup x^\star$, assumes value $u' \in \mathbb{N}$.

We then define the *privacy loss* of a scenario as the multiplicative Bayes leakage of the new row's sensitive value $x^\star[a_s]$ given the reported count $M_{\texttt{count}_q}(D \cup x^\star)$ on the extended dataset $D \cup x^\star$. (Compare Def. 6.) Intuitively, privacy loss reflects by how much knowledge of the reported count increases the adversary's chance of correctly guessing the secret value in one try. Formally:

$$\textit{privacy-loss}(\Gamma) \stackrel{\text{def}}{=} \frac{\sum_{u' \in \mathbb{N}} \max_{s \in domain(a_s)} Pr^\Gamma(x^\star[a_s]{=}s, M_{\texttt{count}_q}(D \cup x^\star){=}u')}{\max_{s \in domain(a_s)} Pr^\Gamma(x^\star[a_s]{=}s)} \ . \tag{7}$$

On the other hand, we define the *utility* of a scenario as the posterior Bayes vulnerability of the real count $\texttt{count}_q(D \cup x^\star)$ of query $\texttt{count}_q$ performed on the extended dataset $D \cup x^\star$ given the reported count $M_{\texttt{count}_q}(D \cup x^\star)$ on the extended dataset $D \cup x^\star$. (Compare Def. 7.) Formally:

$$\textit{utility}(\Gamma) \stackrel{\text{def}}{=} \sum_{u' \in \mathbb{N}} \max_{u \in \mathbb{N}} Pr^\Gamma(\texttt{count}_q(D \cup x^\star){=}u, M_{\texttt{count}_q}(D \cup x^\star){=}u') \ . \tag{8}$$

Equations (7) and (8) depend on the joint probability distribution Pr^Γ induced by the scenario, which itself depends on the coin tosses of the query mechanism $M_{\texttt{count}_q}$ employed. Hence, to fully flesh out these definitions we need to determine how the mechanism $M_{\texttt{count}_q}$ works. We consider two deferentially-private mechanisms adding noise in different ways.

- An *oblivious mechanism* $M^{obv}_{\texttt{count}_q}$ that first applies counting query $\texttt{count}_q$ to the input dataset to produce a real count u, and then applies a (differentially-private) *oblivious randomisation function* $R^{obv}{:}\mathbb{N}{\to}\mathbb{D}(\mathbb{N})$ to u in order to produce a reported count u' as the output of the mechanism. This is similar to a utility-focussed mechanism.
- A *local mechanism* $M^{loc}_{\texttt{count}_q}$ that first applies a (differentially-private) *local randomisation function* $R^{loc}{:}domain(a_u){\to}\mathbb{D}(domain(a_u))$ independently to each row's useful value to produce a randomised dataset D', and only then applies the counting query $\texttt{count}_q$ to D' in order to produce a reported count $u'{=}\texttt{count}_q(D')$ as the output of the mechanism. This is in the spirit of a privacy-focussed mechanism.

In our experiments we used the dataset released by ProPublica [5] corresponding to two years worth of data from the COMPAS tool (Correctional Offender Management Profiling for Alternative Sanctions), which is one of the most popular algorithmic tools used in the United States criminal justice system for pretrial and sentencing evaluation of the risk of bad behaviour for criminal defendants. We focused on the tool's assessment scores for "Risk of Failure to Appear", and eliminated from the dataset all but the most recent record for any given person, as well as all records with invalid entries for attributes `marital_status` and `score_text`. This treatment has left us with a database containing 11,710 unique records.

We then considered two scenarios. In both, the adversary's goal is to learn the newly added row's value for sensitive attribute `custody_status`, which can be 0-"pretrial defendant", 1-"residential program", 2-"probation", 3-"parole", 4-"jail inmate", or 5-"prison inmate". However, in scenario A the data analyst's query of interest is `"select count *`

[5] `https://github.com/propublica/compas-analysis`

`from D where custody_status=0"`, whose result is highly correlated with the secret information, whereas in scenario B the query of interest is `"select count * from D where marital_status=0"` (the range for `marital_status` is 0-"single", 1-"significant other", 2-"married", 3-"separeted", 4-"divorced", or 5-"widowed"), whose result is highly independent from the secret information. In both scenarios data analyst and adversary assume the new row $x^\star$ added to dataset D follows a distribution $\pi^\star{:}\mathbb{D}(\mathit{rows}(\mathcal{A}))$ s.t. the probability of each $x{\in}\mathit{rows}(\mathcal{A})$ is the value's frequency in the dataset D. Moreover, for fairness in comparison, we instantiate both the oblivious randomisation function R^{obv} and the local randomisation function R^{loc} as the truncated geometric mechanism with the same value of ϵ. [6]

Table 1 presents the results of our experiments for various values of ϵ. In terms of inferences, we want the privacy loss to be low to offer good protection for individuals. This means, in the scale of Bayes vulnerability leakage, that the value should be close to 1 because then we can argue that in the studied scenario the information flow does not increase the adversary's prior knowledge. [7] For accuracy however, to offer good utility, we want it to be as high as possible. In the scale of Bayes vulnerability a value close to 1 represents high certainty that the true value of the query can be accurately inferred.

As we can notice, in both scenarios described above and for each value of ϵ the local mechanism is consistently more private, but less useful, than its oblivious counterpart. The experiments also illustrate the well known fact from the literature that local mechanisms tend not to provide good utility in small datasets like the one we consider here: utility remains at its theoretical minimum for relatively high values of ϵ ($\approx \ln 200$ in Scenario A and $\approx \ln 10^3$ in Scenario B). Finally, note that in Scenario A, where real count and secret are highly correlated, it is hard to achieve a satisfactory trade-off between utility and privacy, as these values are always in opposition to each other. On the other hand, since in Scenario B the real count is practically independent of the secret, it is possible to maintain privacy at a minimum level even for very high values of utility.

6 Related work

Our work builds upon the "no free lunch" theorem of Kifer and Machanavajjhala [11], who provided the first analysis of the limitations of differential privacy in the presence of correlations between secrets. Their work also uses inference attack models to demonstrate the effect of correlations on possible inferences resulting in unexpected privacy breaches. Our work complements theirs by utilising the QIF framework to model the effect of inference attacks through the lens of information flow. The same authors proposed Pufferfish, a framework providing a more nuanced approach to privacy that depends on the idiosyncracies of particular datasets [12]. Inspired by Pufferfish, He et al. [7] introduced the Blowfish framework to allow a more tailored approach to privacy policies which depends on known (public) correlations in the dataset.

Other authors have observed that differential privacy does not protect known correlations in datasets. Zhu et al. [21] proposed strengthening privacy mechanisms for correlated datasets based on a modified measure of the sensitivity of the dataset. Liu et al. [13] demonstrate an inference attack against a differentially private dataset by exploiting known correlations in

[6] The *truncated geometric mechanism* with parameter $\epsilon{>}0$, domain $\{0,1,2,\ldots,m\}$, and co-domain $\{0,1,2,\ldots,n\}$ is defined as $G(j|i){=}{}^{1}\!/_{(1+\alpha)}\cdot\alpha^{i}$, if $j{=}0$, or $G(j|i){=}{}^{(1-\alpha)}\!/_{(1+\alpha)}\cdot\alpha^{|i-j|}$, if $0{<}j{<}n$, or $G(j|i){=}{}^{1}\!/_{(1+\alpha)}\cdot\alpha^{|i-n|}$, if $j{=}n$, for every $0{\leq}i{\leq}m$ and $0{\leq}j{\leq}n$, where $\alpha{=}e^{-\epsilon}$, and each value $G(j|i)$ represents the probability that integer i is remapped to integer j.

[7] In a leakage measure we do not tabulate the actual probability of inference, but rather than increase in inference compared to the prior.

Table 1 Results of privacy loss and utility on Scenarios A (with highly correlated counting query and secret) and B (with practically independent counting query and secret) for the COMPAS dataset A value close to 1 for privacy means the data release does not pose a risk to an individual; a value close to 1 for utility means that the data release is accurate.

	Scenario A				**Scenario B**			
	Oblivious mechanism		**Local mechanism**		**Oblivious mechanism**		**Local mechanism**	
ϵ	**Priv. loss**	**Util.**	**Priv. loss**	**Util.**	**Priv. loss**	**Util.**	**Priv. loss**	**Util.**
0 (theoretical minimum)	1.0000	0.7984	1.0000	0.7984	1.0000	0.7704	1.0000	0.7704
ln **3**	1.0000	0.7984	1.0000	0.7984	1.0000	0.7704	1.0000	0.7704
ln **5**	1.0452	0.8333	1.0000	0.7984	1.0000	0.8333	1.0000	0.7704
ln **10**	1.1402	0.9091	1.0000	0.7984	1.0000	0.9091	1.0000	0.7704
ln **100**	1.2418	0.9901	1.0000	0.7984	1.0000	0.9901	1.0000	0.7704
ln **200**	1.2480	0.9950	1.0001	0.7984	1.0000	0.9950	1.0000	0.7704
ln **500**	1.2517	0.9980	1.0035	0.8011	1.0000	0.9980	1.0000	0.7704
ln $\mathbf{10^3}$	1.2529	0.9990	1.0234	0.8169	1.0000	0.9990	1.0000	0.7704
ln $\mathbf{10^5}$	1.2542	1.0000	1.2481	0.9952	1.0000	1.0000	1.0000	0.9330
∞ (theoretical maximum)	1.2607	1.0000	1.2607	1.0000	1.2607	1.0000	1.2607	1.0000

the data. Works in this area focus on known correlations – particularly correlations within the dataset – and do not address the more general problem of unknown correlations which may be known only to an adversary.

Other works on inference attacks on private data consider alternate privacy metrics to measure privacy loss. Salamatian et al. [17] use traditional information theoretic measures such as entropy and mutual information to produce an alternate privacy framework to differential privacy which is focussed on protecting against inference attacks. Most recently, Jayaraman et al. [9] propose new privacy metrics to evaluate the risk of inference attacks.

Finally, there is considerable interest in understanding inference attacks on machine learning models which employ differential privacy to protect their training data. Rahman et al. [16] empirically evaluate the success of membership inference attacks against machine learning models trained with different privacy parameters. Most recently, Jayaraman et al. [9] empirically evaluate the risk of inference attacks on differentially private machine learned models using their own privacy metrics. Works in this area typically use ad hoc methods to evaluate privacy leakage, whereas our QIF framework permits rigorous analysis based on an operational inference attack model.

7 Conclusion

In this paper we have studied the relationship between accuracy and privacy in data releases from the perspective of inferences. We have shown, using a channel model for quantitative information flow, how to capture reasonable assumptions about prior knowledge to compare accuracy of a query result versus the ability for the adversary to infer additional information about individuals in the database. We have demonstrated how correlations in databases of moderate size pose challenges for protecting privacy of individuals.

In future work we hope to use this kind of analysis to expose potential vulnerabilities in databases by considering the threats posed by inferences in proposed data releases.

References

1 Mário S. Alvim, Konstantinos Chatzikokolakis, Pierpaolo Degano, and Catuscia Palamidessi. Differential privacy versus quantitative information flow. *CoRR*, abs/1012.4250, 2010. `arXiv:1012.4250`.

2 Mário S. Alvim, Kostas Chatzikokolakis, Catuscia Palamidessi, and Geoffrey Smith. Measuring information leakage using generalized gain functions. In *Proc. 25th IEEE Computer Security Foundations Symposium (CSF 2012)*, pages 265–279, June 2012.

3 David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative analysis of the leakage of confidential data. *Electr. Notes Theor. Comput. Sci.*, 59(3):238–251, 2001.

4 David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantified interference for a while language. *Electr. Notes Theor. Comput. Sci.*, 112:149–166, 2005.

5 Damien Desfontaines and Balázs Pejó. Sok: Differential privacies. *Proceedings on Privacy Enhancing Technologies*, 2020(2):288–313, 2020.

6 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006. `doi:10.1007/11681878_14`.

7 Xi He, Ashwin Machanavajjhala, and Bolin Ding. Blowfish privacy: Tuning privacy-utility trade-offs using policies. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 1447–1458, 2014.

8 Bargav Jayaraman and David Evans. Evaluating differentially private machine learning in practice. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, page 18951912, USA, 2019. USENIX Association.

9 Bargav Jayaraman, Lingxiao Wang, David Evans, and Quanquan Gu. Revisiting membership inference under realistic assumptions. *arXiv preprint*, 2020. `arXiv:2005.10881`.

10 C. Palamidessi K. Chatzikokolakis, N. Fernandes. Comparing systems: Max-case refinement orders and application to differential privacy. In *Proc. CSF*. IEEE Press, 2019.

11 Daniel Kifer and Ashwin Machanavajjhala. No free lunch in data privacy. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD Õ11, page 193Ð204, New York, NY, USA, 2011. Association for Computing Machinery. `doi:10.1145/1989323.1989345`.

12 Daniel Kifer and Ashwin Machanavajjhala. Pufferfish: A framework for mathematical privacy definitions. *ACM Trans. Database Syst.*, 39(1), January 2014. `doi:10.1145/2514689`.

13 Changchang Liu, Supriyo Chakraborty, and Prateek Mittal. Dependence makes you vulnberable: Differential privacy under dependent tuples. In *NDSS*, volume 16, pages 21–24, 2016.

14 Alvim M, K. Chatzikokolakis, A.K. McIver, C.C. Morgan, G. Smith, and C. Palamidessi. *The Science of Quantitative Information Flow*. Information Security and Cryptography. Springer, 2020. To appear.

15 Arvind Narayanan, Hristo S. Paskov, Neil Zhenqiang Gong, John Bethencourt, Emil Stefanov, Eui Chul Richard Shin, and Dawn Song. On the feasibility of internet-scale author identification. In *IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, California, USA*, pages 300–314. IEEE Computer Society, 2012. `doi:10.1109/SP.2012.46`.

16 Md Atiqur Rahman, Tanzila Rahman, Robert Laganière, Noman Mohammed, and Yang Wang. Membership inference attack against differentially private deep learning model. *Trans. Data Priv.*, 11(1):61–79, 2018.

17 Salman Salamatian, Amy Zhang, Flavio du Pin Calmon, Sandilya Bhamidipati, Nadia Fawaz, Branislav Kveton, Pedro Oliveira, and Nina Taft. Managing your private and public data: Bringing down inference attacks against your privacy. *IEEE Journal of Selected Topics in Signal Processing*, 9(7):1240–1255, 2015.

18 C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

19 Geoffrey Smith. On the foundations of quantitative information flow. In Luca de Alfaro, editor, *Proc. 12th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS '09)*, volume 5504 of *Lecture Notes in Computer Science*, pages 288–302, 2009.

20 S.L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60:63Ð69, 1965.

21 Tianqing Zhu, Ping Xiong, Gang Li, and Wanlei Zhou. Correlated differential privacy: Hiding information in non-iid data set. *IEEE Transactions on Information Forensics and Security*, 10(2):229–242, 2014.

A Survey of Bidding Games on Graphs

Guy Avni
IST Austria, Klosterneuburg, Austria

Thomas A. Henzinger
IST Austria, Klosterneuburg, Austria

Abstract

A graph game is a two-player zero-sum game in which the players move a token throughout a graph to produce an infinite path, which determines the winner or payoff of the game. In *bidding games*, both players have budgets, and in each turn, we hold an "auction" (bidding) to determine which player moves the token. In this survey, we consider several bidding mechanisms and study their effect on the properties of the game. Specifically, bidding games, and in particular bidding games of infinite duration, have an intriguing equivalence with *random-turn* games in which in each turn, the player who moves is chosen randomly. We show how minor changes in the bidding mechanism lead to unexpected differences in the equivalence with random-turn games.

2012 ACM Subject Classification Theory of computation → Solution concepts in game theory; Theory of computation → Formal languages and automata theory

Keywords and phrases Bidding games, Richman bidding, poorman bidding, mean-payoff, parity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.2

Category Invited Paper

Funding This research was supported in part by the Austrian Science Fund (FWF) under grant Z211-N23 (Wittgenstein Award).

Acknowledgements We would like to thank all our collaborators Milad Aghajohari, Ventsislav Chonev, Rasmus Ibsen-Jensen, Ismäel Jecker, Petr Novotný, Josef Tkadlec, and Đorđe Žikelić; we hope the collaboration was as fun and meaningful for you as it was for us.

1 Introduction

Two-player zero-sum games on graphs games have deep connections to foundations of logic [31] as well as numerous practical applications, e.g., verification [19], reactive synthesis [29], and reasoning about multi-agent systems [2]. The game proceeds by placing a token on one of the vertices and allowing the players to move it throughout the graph to produce an infinite trace, which determines the winner or payoff of the game.

Several "modes of moving" the token have been studied. The most well-studied mode is *turn-based* games, in which the vertices are partitioned between the players and whoever controls the vertex on which the token is placed, decides its next position. Two other modes of moving that will make appearances in this survey are *stochastic games* and *concurrent games*. Stochastic games generalize turn-based games in that some vertices are controlled by nature and from which the token moves probabilistically. In concurrent games, in each vertex, the players simultaneously select actions, and the next location is determined according to the joint (deterministic or probabilistic) action vector. For formal definitions, see [3] for example.

We study the *bidding* mode of moving. Abstractly speaking, both players have budgets, and in each turn, we hold an "auction" (bidding) to determine which player moves the token. In this survey, we consider several concrete bidding mechanisms and study the properties of the bidding games that they give rise to.

We emphasize that bidding is a mode of moving the token and bidding games can be studied in combination with any objective. We focus on three objectives: *reachability*, *parity*, and *mean-payoff*. We start by surveying results on reachability games that were obtained in two papers in the 1990s [24, 23]. We then turn to summarize more recent results on infinite-duration games. Our most interesting results are for mean-payoff games. In a nutshell, reachability bidding games with a specific bidding mechanism called *Richman bidding* were shown to be equivalent to a class of games called *random-turn games*, which are well-studied in their own right since the seminal paper [27]. We show a generalized equivalence between mean-payoff bidding games and random-turn games. While the equivalence for finite-duration games holds only for Richman bidding, for mean-payoff games, equivalences with random-turn games hold for a wide range of bidding mechanisms.

We keep the presentation in the survey relatively informal. For formal definitions, see the cited papers.

Bidding mechanisms

In all mechanisms that we consider, both players simultaneously submit "legal" bids that do not exceed their available budgets, and the higher bidder moves the token.

We mainly focus on two orthogonal distinctions between the mechanisms: *who pays* and *who is the recipient*. To answer the latter, two mechanisms were defined in [23]: in *Richman bidding* (named after David Richman), payments are made to the other player, and in *poorman bidding* the payments are made to the "bank" thus the money is lost. A third payment scheme called *taxman* spans the spectrum between Richman and poorman.

We make the payment schemes precise below. For $i \in \{1, 2\}$, suppose Player i's budget is B_i prior to a bidding and his bid is $b_i \in [0, B_i]$, and assume for convenience that Player 1 wins the bidding, thus $b_1 > b_2$. The budgets are updated as follows:

- **First-price:** Only the higher bidder pays.
  - **Richman:** $B'_1 = B_1 - b_1$ and $B'_2 = B_2 + b_1$.
  - **Poorman:** $B'_1 = B_1 - b_1$ and $B'_2 = B_2$.
  - **Taxman:** For a fixed $\tau \in [0, 1]$, we have $B'_1 = B_1 - b_1$ and $B'_2 = B_2 + (1 - \tau) \cdot b_1$.
- **All-pay:** Both players pay their bids.
  - **Richman:** $B'_1 = B_1 - b_1 + b_2$ and $B'_2 = B_2 + b_1 - b_2$. Thus, Player 1 pays Player 2 the difference between the two bids.
  - **Poorman:** $B'_1 = B_1 - b_1$ and $B'_2 = B_2 - b_2$.

Discrete vs. continuous bidding. A third orthogonal property of the bidding mechanism concerns the type of bids that are allowed: in *discrete bidding*, the budgets are given in coins and the minimal positive bid a player can make is one coin. In this survey, apart from Section 5, we consider *continuous bidding*, in which bids can be arbitrary small.

▶ Remark 1 (Ties in biddings). In continuous bidding, we avoid the issue of ties in the questions that we study (see Def. 3 for example). So while one needs to determine how ties are resolved, our results do not depend on a specific tie-breaking mechanism (apart from the corner cases in Thm. 32). Ties cannot be avoided in discrete bidding as we discuss in Section 5. ⌟

▶ **Definition 2** (Budget ratio). *For $i \in \{1, 2\}$, let B_i denote Player i's budget. Player i's ratio is then $B_i/(B_1 + B_2)$. In Richman bidding, since the bids only exchange hands, the sum of budgets is constant and we normalize it to* 1*, so that a player's ratio coincides with his actual budget.*

2 Qualitative First-Price Bidding Games

We consider the following qualitative objectives:

▶ **Definition 3** (Qualitative objectives). *Consider a graph with a set V of vertices.*

- ***Reachability:*** *There are two targets $t_1, t_2 \in V$. The game ends once a target t_i is visited, for $i \in \{1, 2\}$. Then, Player i is the winner.*
- ***Parity:*** *A parity index function is $p : V \to \mathbb{N}$. Player 1 wins an infinite play iff the maximal parity index that is visited infinitely often is odd.*

▶ Remark 4. Note that the definition of reachability objectives is slightly different than the typical definition in which only one player has a target and the second player (the *safety* player) wins iff the target is not reached. The main difference is that in the two-target version, there is no winner in a play in which both targets are not reached. As we show in Thm. 7 below, such ties do not occur. Thus, the two types of reachability objectives coincide in bidding games. See [4] for more details. ⌟

The main question studied in qualitative bidding games is the following:

What is a necessary and sufficient initial budget ratio that guarantees winning the game?

We illustrate a solution to this question in the following example.

▶ **Example 5.** Consider the reachability first-price Richman bidding game that is depicted in Figure 1. Player 1's goal is to reach t_1, and Player 2's goal is to reach t_2. We start with a naive solution by showing that Player 1 wins when his[1] budget exceeds 0.75. Suppose that the budgets are $\langle 0.75 + \varepsilon, 0.25 - \varepsilon \rangle$, for Player 1 and 2, respectively, for $\varepsilon > 0$. In the first turn, Player 1 bids 0.25 and wins the bidding since Player 2 cannot bid above $0.25 - \varepsilon$. The budgets are updated to $\langle 0.5 + \varepsilon, 0.5 - \varepsilon \rangle$ (in first-price Richman, only the higher bidder pays his bid to the other player). Player 1 moves the token to v_2. In the second bidding, Player 1 bids all his budget, wins the bidding since Player 2 cannot bid above $0.5 - \varepsilon$, moves the token to t_1, and wins the game.

A ratio of 0.75 thus suffices for winning, but is it necessary? No. The necessary and sufficient budget is in fact 2/3. More formally, we show that for every $\varepsilon > 0$, Player 1 can win when the initial budgets are $\langle 2/3 + \varepsilon, 1/3 - \varepsilon \rangle$. Player 1's first bid is 1/3, thus he wins the wins the bidding and moves the token to v_2. The budgets are updated to $\langle 1/3 + \varepsilon, 2/3 - \varepsilon \rangle$. Next, Player 1 bids all his budget, namely $1/3 + \varepsilon$. If he wins the bidding, he proceeds to t_1 and wins the game. Otherwise, Player 2 wins the bidding, and moves the token back to v_0. The key observation is that since Player 2 overbids, she bids at least $1/3 + \varepsilon$, thus Player 1's new budget is at least $2/3 + 2\varepsilon$. In other words, we are back to v_0 only that Player 1's increases by ε. By continuing in a similar manner, Player 1 forces his budget to increase by a constant every time we return to v_0. Thus, eventually, his budget exceed 0.75 and he can use the strategy above to force the game to t_1. A symmetric argument shows that Player 2 wins when the budgets are $\langle 2/3 - \varepsilon, 1/3 + \varepsilon \rangle$. ⌟

▶ **Definition 6** (Threshold ratios). *Consider a bidding game over a set of vertices V and let $B_1 \in [0, 1]$ be Player 1's initial budget ratio. The threshold ratio in a vertex $v \in V$, denoted $\mathit{Th}(v) \in [0, 1]$ is such that*

- *when $B_1 > \mathit{Th}(v)$, Player 1 can win the game from v, and*
- *when $B_1 < \mathit{Th}(v)$, Player 2 can win the game.*

[1] For fairness, we refer to Player 1/Max as *he* and Player 2/Min as *she*.

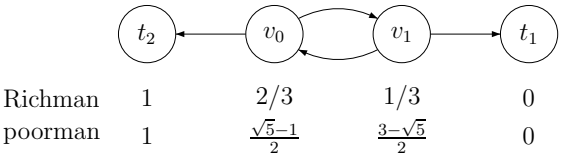


Figure 1 A reachability bidding game with the threshold ratios under first-price Richman and poorman bidding.

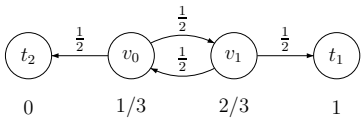


Figure 2 The random-turn game that corresponds the game in Fig. 1 with the probabilities to reach t_1 from each vertex.

2.1 Reachability first-price bidding games

The following theorem identifies threshold ratios in reachability first-price bidding games. See Fig. 1 for a specific instantiation of the theorem.

▶ **Theorem 7** ([23]). *Consider a reachability bidding game with target states t_1 and t_2. Threshold ratios exist under taxman bidding. Moreover, $\mathit{Th}(t_1) = 0$ and $\mathit{Th}(t_2) = 1$, and for all other vertices v, let v^- and v^+ be the neighbors of v such that $\mathit{Th}(v^-) \leq \mathit{Th}(v') \leq \mathit{Th}(v^+)$, for every neighbor v' of v. Then:*

- *under Richman bidding, $\mathit{Th}(v) = \frac{1}{2}(\mathit{Th}(v^+) + \mathit{Th}(v^-))$,*
- *under poorman bidding, $\mathit{Th}(v) = \frac{\mathit{Th}(v^+)}{1-\mathit{Th}(v^-)+\mathit{Th}(v^+)}$, and*
- *under taxman bidding with constant $\tau \in [0,1]$, we have $\mathit{Th}(v) = \frac{\mathit{Th}(v^+)+\mathit{Th}(v^-)-\mathit{Th}(v^-)\cdot\tau}{2-(1+\mathit{Th}(v^-)-\mathit{Th}(v^+))\cdot\tau}$.*

Proof. We sketch the proof for Richman bidding to give a flavor of the techniques. Suppose the initial vertex is v and that Player 1's budget is $B = \mathtt{Th}(v) + \varepsilon$. Player 1 maintains the invariant that whenever the token is placed on a vertex u, his ratio exceeds $\mathtt{Th}(u)$. The invariant holds initially. Suppose Player 1 bids $b = \frac{1}{2} \cdot \big(\mathtt{Th}(v^+) - \mathtt{Th}(v^-)\big)$, and upon winning, moves the token to v^-. If Player 1 wins the bidding, since $\mathtt{Th}(v) - b = \mathtt{Th}(v^-)$, the invariant is maintained at v^-. If he loses the bidding, Player 2 pays him at least b, and since $\mathtt{Th}(v) + b = \mathtt{Th}(v^+)$ and $\mathtt{Th}(v^+) \geq \mathtt{Th}(v')$, for every neighbor v', no matter where Player 2 moves, the invariant is maintained. The invariant implies that Player 1 does not lose. Indeed, recall that the sum of budgets is 1 in Richman bidding. Since $\mathtt{Th}(t_2) = 1$, the game cannot reach t_2, since then Player 1 owns more than the sum of budgets. To win the game, Player 1 plays as in Example 5: he uses his "excess" budget ε to bid slightly more than b so that he guarantees not losing while accumulating budget as the game proceeds. Eventually he accumulates enough budget to win $|V|$ biddings in a row and draws the game to t_1. ◀

Thm. 7 implies an intriguing equivalence between reachability Richman-bidding games and random processes. Before stating it in full generality, we illustrate the "plain vanilla" version of the equivalence, which applies to games in which all vertices have out-degree at most 2.

▶ **Example 8.** Consider the game depicted in Fig. 1. We construct a Markov chain by labeling all edges with probability 0.5 (see Fig. 2). Consider the probability of reaching the target t_1 from a vertex u, denoted $\mathbb{P}[\text{reach}(u, t_1)]$. Clearly, we have $\mathbb{P}[\text{reach}(t_1, t_1)] = 1$ and $\mathbb{P}[\text{reach}(t_2, t_1)] = 0$. As for the other two vertices, we have $\mathbb{P}[\text{reach}(v_0, t_1)] = \frac{1}{2}\mathbb{P}[\text{reach}(v_1, t_1)] + \frac{1}{2}\mathbb{P}[\text{reach}(t_2, t_1)]$, which is technically the same as the expression in Thm. 7 for threshold ratios in Richman bidding. Since the values for the targets are reversed, for every vertex u, we have $\mathtt{Th}(u) = 1 - \mathbb{P}[\text{reach}(u, t_1)]$. ⌟

Graphs with out-degree 2 are easier to reason about since it is clear which vertices are v^- and v^+ (which vertex is which does not matter, since we take a simple average between the two). In general, however, determining which are the two "important" vertices out of all the neighbors is not trivial.

▶ **Definition 9** (Random-turn game). *Consider a bidding game $\mathcal{G}$ that is played on a graph over a set of vertices V. For $p \in [0,1]$, the random-turn game that corresponds to $\mathcal{G}$ w.r.t. p, denoted $\mathsf{RT}(\mathcal{G}, p)$, is a game in which instead of bidding, in each turn we toss a (biased) coin to determine which player gets to move the token: Player 1 is chosen with probability p and Player 2 with probability $1-p$. Formally, $\mathsf{RT}(\mathcal{G}, p)$ is a stochastic game [17]. Every vertex $v \in V$, is replaced by three vertices v_N, v_1, and v_2. The vertex v_N simulates the coin toss, thus it has two outgoing edges: one with probability p to v_1 and a second with probability $1-p$ to v_2. For $i \in \{1,2\}$, the vertex v_i is controlled by Player i and there are deterministic edges from v_i to u_N, for every neighbor u of v. The objective in $\mathsf{RT}(\mathcal{G}, p)$ matches that of $\mathcal{G}$. For example, when $\mathcal{G}$ is a reachability bidding game, then $\mathsf{RT}(\mathcal{G}, p)$ is a simple stochastic game.*

A *positional strategy* f in a stochastic game is a function that maps each vertex to a successor. When Player i, for $i \in \{1,2\}$, plays according to f and the game reaches a vertex v that he controls, he moves the token to $f(v)$. Fixing two positional strategies in a stochastic game gives rise to a Markov chain. We restrict attention to positional strategies since in stochastic games with the objectives that we consider, existence of optimal positional strategies is well known.

▶ **Definition 10** (Values in simple stochastic games). *Consider a simple stochastic game $\mathcal{H}$ and a vertex v in $\mathcal{H}$. The* value *of v in $\mathcal{H}$, denoted $val(\mathcal{H}, v)$, is the probability that Player 1 reaches his target under optimal play of the two players.*

We state the general equivalence between the two models in the following theorem.

▶ **Theorem 11** ([24]). *Consider a reachability first-price Richman bidding game $\mathcal{G}$ over the vertices V. For every vertex $v \in V$, we have $\mathit{Th}(v) = 1 - val\big(\mathsf{RT}(\mathcal{G}, 0.5), v\big)$.*

▶ Remark 12. We note that in graphs of finite size, threshold ratios under Richman bidding are always rational numbers; indeed, they coincide with values of a stochastic game, which in turn are a solution to a linear program. Since for poorman bidding the thresholds can be irrational (see Fig. 1), it seems unlikely that an equivalence with random-turn games exists. ⌟

2.2 Parity first-price bidding games

The solution to parity bidding games is based on the following lemma.

▶ **Lemma 13** ([7]). *Consider a reachability first-price taxman-bidding game $\mathcal{G}$ over the vertices V such that from every vertex, only Player 1's objective is reachable. Then, Player 1 wins with any positive initial budget from every vertex $v \in V$; thus, $\mathit{Th}(v) = 0$.*

We proceed to solve parity games.

▶ **Theorem 14** ([7]). *Parity first-price taxman-bidding games are linearly reducible to reachability taxman-bidding games. In particular, threshold ratios exists.*

Proof. Consider a bidding game $\mathcal{G}$. we identify the bottom strongly-connected components (BSCCs) of $\mathcal{G}$. We claim that in each such BSCC S there is a unique winner. The claim implies the theorem since the threshold ratios in the vertices of S are either all 0 or all 1. We then construct a reachability game on the rest of the graph in which each player's goal is to draw the game to a BSCC that is winning for him.

To prove the claim, suppose the maximal parity index in a BSCC S is obtained in a vertex t and that it is even, and the other case is dual. We think of S as a reachability game in which Player 2's target is t and Player 1 has no target. By Lem. 13, Player 2 can force a visit to t with any positive initial budget. To force infinite many visits, she splits her initial budget $\varepsilon > 0$ into infinite many parts $\varepsilon_1, \varepsilon_2, \ldots$, and uses a budget of ε_i, for $i \geq 1$, to force a visit to t for the i-th time. ◀

2.3 Computational complexity and open problems

Intuitively, the computational problem that we would like to solve is finding the threshold ratio in a vertex. Formally, one way to define the corresponding decision problem is given a game $\mathcal{G}$ and a vertex v in $\mathcal{G}$, decide whether $\mathtt{Th}(v) > 0.5$.

▶ **Theorem 15** ([23, 4, 5, 7])**.** *For parity Richman bidding, finding threshold ratios is in NP ∩ coNP, and it is in P when all vertices have outdegree at most* 2 *or when the graph is undirected. For taxman and poorman bidding the problem is in PSPACE.*

Proof (Sketch). The upper bound for Richman bidding immediately follows from the equivalence with random-turn games in Thm. 11. Indeed, random-turn games are a special case of stochastic games, and solving the later is in NP ∩ coNP [17]. When the outdegree is 2, the solution is obtained by solving a linear program. An algorithm for undirected graphs was shown in [23] (and was extended recently to solve biased undirected random-turn games [28]). For taxman bidding, the properties of threshold ratios in Thm. 7 are polynomial but not linear, thus we reduce the problem to the *existential theory of the reals* [15]. ◀

▶ **Open problem 1** (Tightening the computational complexity gap)**.** *There is a gap in our understanding in terms of computational complexity. It is possible that solving Richman bidding games is in P, and it is possible that they are as hard as solving stochastic games. Possible lower bounds for poorman and taxman bidding include showing that the problem is NP-hard or as hard as solving the existential theory of the reals.*

3 Mean-Payoff First-Price Bidding Games

Before formalizing the mean-payoff objective, we motivate it in the following application.

▶ **Example 16.** Suppose an internet content-provider (e.g., New York Times) has one ad slot for sale on its website. Two advertisers repeatedly (e.g., daily) compete to publish their ad in a first-price auction that the content provider holds. Our goal is to find an optimal bidding strategy for an advertiser that, given his budget constraints, maximizes the long-run ratio of the time that his ad shows. To find such a strategy, we reason about the Bowtie game in Fig. 3; we associate our advertiser with Player 1. Whenever Player 1 wins a bidding, he moves to v_{Max}, which represents his ad being shown that day. Informally, Player 1's goal is to maximize the time his ad shows (e.g., maximize the number of days his ad appears in a year). ⌟

Formally, a *mean-payoff game* is played on a weighted graph $\langle V, E, w\rangle$, where $w : V \to \mathbb{Q}$. Each infinite play has a *payoff*, which is Player 1's reward and Player 2's cost, thus we call the players in a mean-payoff game Max and Min, respectively.

▶ **Definition 17** (Payoff and energy). *Consider an infinite path $\eta = \eta_0, \eta_1, \ldots$. For $n > 1$, let $\eta^n = \eta_0, \ldots, \eta_n$ be a prefix of η. The* energy *of η^n, denoted $\mathsf{energy}(\eta^n)$, is the sum of weights it traverses, thus $\mathsf{energy}(\eta^n) = \sum_{0 \le i < n} w(\eta_i)$. The* payoff *of η, denoted $\mathsf{payoff}(\eta)$, is $\mathsf{payoff}(\eta) = \liminf_{n\to\infty} \frac{1}{n} \cdot \mathsf{energy}(\eta^n)$. Note that the use of* lim inf *gives Min an advantage.*

We consider the following main question when studying mean-payoff bidding games.

Given an initial budget ratio r, what is the optimal payoff a player can guarantee? For example, consider the Bowtie game in Fig. 3. What is the optimal payoff Max can guarantee with a budget ratio of $2/3$ under Richman bidding? How does it compare with the optimal payoff under poorman bidding? Does the answer change when the ratio is $1/3$?

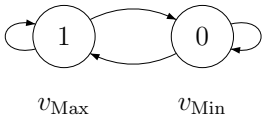


Figure 3 The Bowtie mean-payoff game $\mathcal{G}_{\bowtie}$.

We answer the questions in full generality after the following definitions. The specific solutions in the Bowtie game can be found in Example 21.

▶ **Definition 18** (Mean-payoff value in bidding games). *Consider a strongly-connected mean-payoff bidding game $\mathcal{G}$ and a budget ratio $r \in (0, 1)$. The* mean-payoff value *of $\mathcal{G}$ w.r.t. r, denoted $\mathsf{MP}(\mathcal{G}, r)$, is $c \in \mathbb{R}$ if independent of the initial vertex,*

- *when Max's initial ratio exceeds r, he[1] has a strategy that guarantees a payoff of $c - \varepsilon$, for every $\varepsilon > 0$, and*
- *Max cannot do better: with a ratio that exceeds $1 - r$, Min can guarantee a payoff of at most $c + \varepsilon$, for every $\varepsilon > 0$.*

▶ **Definition 19** (Mean-payoff value in random-turn games). *For $p \in [0, 1]$, since $\mathcal{G}$ is a mean-payoff game, the random-turn game $\mathsf{RT}(\mathcal{G}, p)$ is a stochastic mean-payoff game. The expected payoff under optimal play of the two players is called the mean-payoff value, which we denote $\mathsf{MP}(\mathsf{RT}(\mathcal{G}, p))$. The value is known to exist [30], and since $\mathcal{G}$ is strongly-connected, the value does not depend on the initial vertex.*

The theorem below states the equivalence between mean-payoff first-price bidding games and random-turn games.

▶ **Theorem 20** ([5, 4, 7]). *Consider a strongly-connected mean-payoff game $\mathcal{G}$ and a ratio $r \in (0, 1)$. Then:*

- *under Richman bidding, $\mathsf{MP}(\mathcal{G}, r) = \mathsf{MP}\big(\mathsf{RT}(\mathcal{G}, 0.5)\big)$,*
- *under poorman bidding, $\mathsf{MP}(\mathcal{G}, r) = \mathsf{MP}\big(\mathsf{RT}(\mathcal{G}, r)\big)$, and*
- *under taxman bidding with constant $\tau \in [0, 1]$, we have $\mathsf{MP}(\mathcal{G}, r) = \mathsf{MP}\big(\mathsf{RT}(\mathcal{G}, \frac{r+\tau\cdot(1-r)}{1+\tau})\big)$.*

▶ **Example 21.** We apply Thm. 20 to the bowtie mean-payoff game $\mathcal{G}_{\bowtie}$ (Fig. 3). Since the outdegree in $\mathcal{G}_{\bowtie}$ is 2, the random-turn game that corresponds to it is simple and depicted in Fig. 4. Informally, we expect a random run to "stay" in v_{Max} portion p of the time, and since the weights are simple, we have $\mathsf{MP}(\mathsf{RT}(\mathcal{G}_{\text{Bowtie}}, p)) = p$.

Under Richman bidding, the initial ratio does not matter, the equivalence is always with the fair random-turn game, thus $\mathsf{MP}(\mathcal{G}_{\bowtie}, r) = 0.5$, for every $r \in (0, 1)$. We find the result for mean-payoff poorman bidding more surprising, since no equivalence is known for reachability poorman bidding games (see Remark 12). With an initial ratio of $r \in (0, 1)$, the optimal payoff under poorman bidding is r. Thus, Max prefers poorman over Richman when his ratio is $2/3$ and prefers Richman over poorman when the ratio is $1/3$. Interestingly, when the ratio is $1/2$, the payoffs under poorman and Richman bidding coincide. ⌟

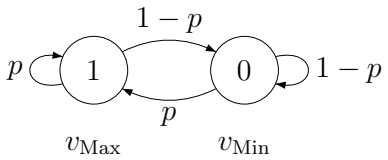


Figure 4 The random-turn game that corresponds to $\mathcal{G}_{\bowtie}$ in Fig. 3 w.r.t. $p \in [0, 1]$.

▶ Remark 22 (Strategies in bidding games vs. in random-turn games). We point out that strategies in bidding games are much more complicated than in stochastic games. At a vertex v in a stochastic game, a strategy only needs to select a vertex u to move the token to from v. In a bidding game, in addition to the choice of u, a strategy prescribes a bid b. As the proof of Thm. 7 shows, in reachability games, knowing the threshold ratios and assuming they are positive, immediately gives us both u and b. In mean-payoff games, this is no longer the case. Indeed, the statement of Thm. 20 only tells us the optimal payoff a player can guarantee. Knowing the optimal payoff does not hint at how to achieve it. The proof of Lem. 23 demonstrates that finding the right bids is not a trivial task. In fact, there is an alternative existential proof for the Richman part of Thm. 20 (see [4]). The proof relies on the equivalence shown in [23] between reachability Richman bidding games and random-turn games adapted to infinite graphs together with results on *one-counter simple-stochastic games* [14, 13]. Beyond the lack of bidding strategy, since no equivalence is known for reachability objectives apart from Richman bidding, the existential proof does not extend to poorman or taxman bidding. ⌟

3.1 Solving the Bowtie game

In the next two sections, we give a flavor of the techniques used to prove Thm. 20. In this section, we solve $\mathcal{G}_{\bowtie}$ and in the next section, we describe a framework to extend a solution to $\mathcal{G}_{\bowtie}$ to general SCCs. See Open problem 3 for a discussion on the asymmetry between Min and Max in the following lemma.

▶ **Lemma 23.** *Consider the mean-payoff game $\mathcal{G}_{\bowtie}$ (Fig. 3) under Richman bidding. Irrespective of the initial ratios:*

- *Min has a strategy that guarantees a payoff of at most* 0.5.
- *For every $\varepsilon > 0$, Max has a strategy that guarantees a payoff of at least $0.5 - \varepsilon$.*

Proof. An optimal Min strategy. The details of the construction can be found in [4]. It is convenient to re-normalize the weights and set $w(v_{\text{Min}}) = -1$ so that $\mathsf{MP}\big(\mathsf{RT}(\mathcal{G}_{\bowtie}, 0.5)\big) = 0$. Recall that the energy of a finite path, which we denote with k below, is the sum of weights that it traverses. The construction is based on the following observation whose proof is a simple exercise.

Observation: Suppose Min plays according to a strategy that guarantees that in every infinite play either the energy (1) equals 0 infinitely often, or (2) is bounded from above by a constant. Then, the payoff is non-negative.

Suppose that the energy starts from $k_0 > 0$ and Min's initial budget is $B^I_{\text{Min}} > 0$. Min chooses $N \in \mathbb{N}$ such that $B^I_{\text{Min}} = k_0/N + \delta$, for some $\delta > 0$. This is possible since B^I_{Min} and k_0 are constants. We think of δ as "spare change" that Min does not use, but will be helpful at the end of the proof. We call the rest of her budget the *main* budget.

Min's strategy maintains the following invariant: when the energy of a finite play is k, Min's budget exceeds $\frac{k}{N}$. The invariant holds initially by our choice of N. Suppose that Min's main budget following a finite play with energy k is $\frac{k}{N}$. Min bids $\frac{1}{N}$. We distinguish between two cases. If Min wins the bidding, the energy drops to $k-1$ and since she pays $\frac{1}{N}$, her main budget drops to $\frac{k-1}{N}$. On the other hand, if she loses, the energy increases to $k+1$, and Max pays her at least $\frac{1}{N}$, thus her main budget increases to $\frac{k+1}{N}$. The strict inequality in the invariant is obtained from the spare change δ. Finally, recall that in Richman bidding the sum of budgets is 1. Thus, the invariant ensures that the energy is bounded from above by N, since by plugging $k = N$ in the invariant, Min's budget would exceed 1.

To conclude the construction, Min plays as follows. When the energy is 0, Min bids 0 and "waits" for the energy to increase. When it increases to k^1_0, she plays according to the strategy above to guarantee a bounded energy N_1. When the energy drops to 0, Min "waits" until the energy increases again to k^2_0, and she plays to keep the energy bounded by N_2, and so on. If there is an $n \geq 1$, such that the energy hits 0 only $n-1$ times, then the energy is bounded from above by N_n. Otherwise, the energy hits 0 infinitely often. Thus, by the observation, the payoff is at most 0.

An optimal Max strategy. An explicit construction for Max was shown in [4] and an existential one in [7]. We describe a simpler construction, which is due to Ismaël Jecker [9].

Let $\varepsilon > 0$. This time, we re-normalize the weights to be $w(v_{\text{Max}}) = c = 1 + \varepsilon$ and $w(v_{\text{Min}}) = -1$.

Observation: Suppose Max plays according to a strategy that guarantees that the energy is bounded from below by a constant. Then, the payoff with the updated weights is non-negative, and the payoff with the original weights is at least $\frac{1}{2+\varepsilon}$.

Let α such that $(1+\alpha) = (1-\alpha)^{-c}$, and such α exists since $c > 1$. Max maintains the invariant that when the energy is $k \in \mathbb{N}$, his budget exceeds $(1+\alpha)^{-k}$. The invariant implies $k > 0$, since otherwise, Max's budget is greater than $(1+\alpha)^0 = 1$, which, again, is impossible since the sum of budgets in Richman bidding is 1. We choose an initial energy level $k_0 \in \mathbb{N}$, thus the energy will in fact be bounded from below by $-k_0$. We choose k_0 such that $B^I_{\text{Max}} = B + \delta$, where $B = (1+\alpha)^{-k_0}$ and $\delta > 0$. This is possible since $\lim_{k\to\infty}(1+\alpha)^{-k} = 0$. Again, the "spare change" δ is never used and we refer to B as Max's main budget. Max maintains the invariant that when the energy is k, his main budget is at least $(1+\alpha)^{-k}$. When Max's main budget is B, he bids $\alpha \cdot B$. We distinguish between the two outcomes of a bidding. If Max loses, the energy decreases to $k-1$. Moreover, Min overbids Max, thus Max's new main budget B' is at least $B + \alpha B \geq \frac{1}{(1+\alpha)^k} + \frac{\alpha}{(1+\alpha)^k} = (1+\alpha)^{k-1}$. On the other hand, if Max wins, the energy increases to $k+c$ and his new main budget B' is at least $B - \alpha B = \frac{1-\alpha}{(1+\alpha)^k}$. Since $(1+\alpha) = (1-\alpha)^{-c}$, we obtain $(1-\alpha) = (1+\alpha)^{-c}$, thus $B' = (1+\alpha)^{-(k+c)}$, as required. ◀

Poorman bidding

An explicit construction of an optimal strategy for Max under poorman bidding was shown in [5], an existential construction was shown in [7], and a significantly simpler explicit construction was shown in [9]. Those constructions use similar ideas to the construction

shown in Lem. 23 for Max, though they are technically more involved. We leave the constructions out of this survey for sake of brevity. We would still like to convince the reader that contrary to Richman bidding, the initial ratio matters in poorman bidding; i.e., with a higher ratio a player can guarantee a better payoff. Below we show a simple strategy construction that is optimal for Min in the Bowtie game.

▶ **Lemma 24** ([5]). *Suppose that Min's initial budget is $\ell \in \mathbb{N}$ and Max's initial budget is 1. Then, in $\mathcal{G}_{\bowtie}$, Min can guarantee a payoff of at most $\frac{\ell}{\ell+1}$.*

Proof. We re-normalize the weights so that $w(v_{\text{Min}}) = -1$ and $w(v_{\text{Max}}) = 1$. Suppose first that both budgets are 1. We describe the "tit for tat" strategy for Min that, intuitively, copies Max's bidding strategy. We assume for simplicity that Min wins bidding ties. Formally, Min's strategy is based on a queue of numbers as follows: if the queue is empty, Min bids 0, and otherwise she bids the maximal number in the queue. If Min wins with a bid of $b > 0$, she removes b from the queue. If Max wins with a bid of $b > 0$, Min adds b to the queue.

We make several observations. (1) Min's strategy is legal: it never bids higher than the available budget; indeed, Max bids legally, the budgets are the same, and Min's sum of winning bids is at most Max's sum of winning bids. (2) The size of the queue is an upper bound on the energy; indeed, every bid in the queue corresponds to a Max winning bid that is not "matched" (the size is an upper bound since Min might win biddings when the queue is empty). (3) If Min's queue fills, it will eventually empty; indeed, if $b \in \mathbb{R}$ is in the queue, in order to keep b in the queue, Max must bid at least b, thus eventually his budget runs out. Combining the three, since the energy is at most 0 when the queue empties, Min's strategy guarantees that the energy is at most 0 infinitely often and as in Lem. 23, the payoff is non-positive.

Now, assume Min's budget is $\ell \in \mathbb{N}$. Min's strategy is the same only that whenever Max wins with $b > 0$, Min adds ℓ copies of b to the queue. The strategy is legal since, intuitively, we can think of Min's budget as if it consists of ℓ parts of size 1, and each copy of the bid b is paid out of a unique part. Legality then follows as in the simple case above. The other two observations above still hold, only that now, since every Max win is matched by ℓ Min wins, when the queue empties, the number of Min wins is at least ℓ times as much as Max's wins. It follows that the payoff is at most $\frac{\ell}{\ell+1}$. ◀

3.2 Solving strongly-connected games

In this section, we describe a general framework that, intuitively, allows us to extend a solution for $\mathcal{G}_{\bowtie}$ to any strongly-connected mean-payoff game. We describe the challenges of the extension in the following example.

▶ **Example 25.** It is helpful to think about changes in the budget and energy throughout a play as two "walks" on two sequences; a "budget sequence" and an "energy sequence". We preserve an invariant between the two walks. Recall, for example, that in Lem. 23, when the energy is k, Max's ratio is $(1+\alpha)^{-k}$. When Max wins a bidding, the energy walk takes a step to $k+1$ and the budget walk takes a step down to $(1+\alpha)^{-(k+1)}$. The key idea that concludes the proof is that the budget walk is bounded from above, e.g., in Richman bidding the sum of budgets is 1. Due to the invariant, the bound on the budget implies a bound on the energy and in turn a guarantee on the payoff.

The structure of $\mathcal{G}_{\bowtie}$ is convenient since in each bidding, both walks always take exactly one step at a time. In general, this is not the case. For example, consider the game depicted in Fig. 5. Cycling v_4 decreases the energy by 2, so to even out, the budget walk should also

take two steps at once. Moreover, there are more involved cycles. Suppose the game starts from v_3. Note that Max always moves left (see the red edges) and Min moves to the right. How should Max bid in v_3 to maintain the invariant between budget and energy? On the one hand, if he wins the first bidding and loses the second, the cycle v_3, v_2, v_3 forms with a change in energy of -2. On the other hand, if he loses the first bidding and wins the second, the cycle $v_3v_4v_3$ forms with a change of energy of -3. Finally, there is no guarantee that the game returns to v_3 ⌟

To overcome these challenges, we introduce a measure of "importance" to vertices, which we call the *strength*. We need several definitions. Let $p \in [0,1]$. Inspired by the notation in reachability games, we denote by v^- and v^+ the optimal vertices to move to from v for Min and Max, respectively. Formally, for optimal memoryless strategies σ_{Max} and σ_{Min} in $\mathsf{RT}(\mathcal{G}, p)$, we define $v^- = \sigma_{\text{Min}}(v)$ and $v^+ = \sigma_{\text{Max}}(v)$. The concept of *potentials* was originally defined in the context of the strategy iteration algorithm [20]. We denote the potential of v by $\text{Pot}_p(v)$ and the strength of v by $\text{St}_p(v)$, and we define them as solutions to the following equations:

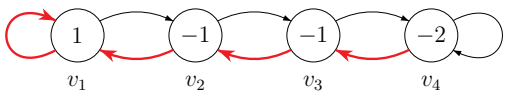
$$\text{Pot}_p(v) = p \cdot \text{Pot}_p(v^+) + (1-p) \cdot \text{Pot}_p(v^-) + w(v) - \mathsf{MP}(\mathsf{RT}(\mathcal{G}, p))$$
$$\text{St}_p(v) = p \cdot (1-p) \cdot \big(\text{Pot}_p(v^+) - \text{Pot}_p(v^-)\big)$$

We suggest an intuitive way to read the definition of potentials. Consider a weighted Markov chain in which each vertex v has two neighbors v^+ and v^-, and the probability of proceeding from v to v^+ and v^- is respectively p and $1-p$. Suppose there is a target that is reached with probability 1. Let $E(v)$ denote the expected energy in a path from v to the target. We note that $E(v)$ roughly coincides with $\text{Pot}_p(v)$. Indeed, to compute $E(v)$, we take the current energy, i.e., $w(v)$, and add, for each neighbor u, the probability of proceeding from v to u multiplied by the expected energy to the target from u, thus we obtain $E(v) = w(v) + p \cdot E(v^+) + (1-p) \cdot E(v^-)$.

The following lemma connects potential, energy, and strength and thus gives rise to the invariant between budget and energy. We need several definitions. Consider a finite path $\eta = v_1, \ldots, v_n$ in $\mathcal{G}$. We intuitively think of η as a play, where for every $1 \leq i < n$, the bid of Max in v_i is $\text{St}_p(v_i)$ and he moves to v_i^+ upon winning. Thus, when $v_{i+1} = v_i^+$, we think of Max as *investing* $\text{St}_p(v_i)$ and when $v_{i+1} \neq v_i^+$, we think of Min winning the bid thus Max *gains* $\text{St}_p(v_i)$. We denote by $I(\eta)$ and $G(\eta)$ the sum of investments and gains, respectively. Recall that the energy of η is $\mathsf{energy}(\eta) = \sum_{0 \leq i < n} w(v_i)$.

▶ **Lemma 26** ([4, 5, 7]). *Consider a strongly-connected game $\mathcal{G}$, and $p = \frac{\nu}{\nu+1} \in (0,1)$, thus Max's budget is ν and Min's budget is 1. For every finite path $\eta = v_1, \ldots, v_n$ in $\mathcal{G}$, we have*

$$Pot_p(v_1) - Pot_p(v_n) + (n-1) \cdot \mathsf{MP}(\mathsf{RT}(\mathcal{G}, p)) \leq \mathsf{energy}(\eta) + G(\eta) \cdot \nu - I(\eta).$$



	v_1	v_2	v_3	v_4
Potential:	6	3	0	-3
Strength:	1	2	2	1

Figure 5 A mean-payoff game $\mathcal{G}$ with $\mathsf{MP}(\mathsf{RT}(\mathcal{G}, 2/3)) = 0$, Max's optimal moves are depicted in red, and $\text{Pot}_{2/3}$ and $\text{St}_{2/3}$ below the vertices.

▶ **Example 27.** Consider the game depicted in Fig. 5. Max always proceeds left, that is, for every vertex v, the vertex v^+ is the one to its left. The potentials and the resulting strengths are depicted below the vertices. It is not hard to verify that the stationary distribution of $\mathsf{RT}(\mathcal{G}, 2/3)$ is $\langle \frac{8}{15}, \frac{4}{15}, \frac{2}{15}, \frac{1}{15} \rangle$. Multiplying by the weights, we obtain $\mathsf{MP}(\mathsf{RT}(\mathcal{G}, 2/3)) = 0$.

It is clearest to exemplify Lem. 26 in cycles, where the left-hand side of the inequality completely cancels out. Consider the cycle $\eta = v_3, v_2, v_1, v_1, v_2, v_3$ in which Max wins the first three bids, thus $I(\eta) = 2 + 2 + 1 = 5$, and loses the last two biddings, thus $G(\eta) = 1 + 2$. We have $\mathsf{energy}(\eta) = -1$ since the last vertex on the path does not contribute to the energy. As Lem. 26 predicts, we have $I(\eta) - G(\eta) \cdot \nu = 5 - 3 \cdot 2 = -1 = \mathsf{energy}(\eta)$. We encourage the reader to verify that every cycle has this property. ⌟

3.3 Solving general games, computational complexity, and open problems

As in the case of qualitative objectives, there is a gap in our understanding of the computational complexity of mean-payoff games. We state the known upper bounds.

▶ **Theorem 28** ([4, 5, 7]). *Given a mean-payoff game $\mathcal{G}$, a vertex v in $\mathcal{G}$, a ratio $r \in (0, 1)$, and a constant $c \in \mathbb{Q}$, deciding whether Max can guarantee a payoff greater than c from v with an initial ratio that exceeds r is:*

- *In PSPACE for taxman bidding and general graphs.*
- *In NP ∩ coNP for Richman bidding, or with taxman bidding when $\mathcal{G}$ is strongly-connected.*
- *In P when $\mathcal{G}$ is strongly-connected and the out-degree of all vertices is 2.*

Proof (Sketch). Thm. 20 allows us to reduce general mean-payoff games to reachability games. Under Richman bidding, the theorem implies that every bottom strongly-connected component (BSCC) $\mathcal{S}$ of $\mathcal{G}$ has a "winner": we say that $\mathcal{S}$ is winning for Max iff $\mathsf{MP}\big(\mathsf{RT}(\mathcal{S}, 0.5)\big) > c$. Indeed, all Max needs to do, is to reach $\mathcal{S}$ with a positive ratio. Upon reaching $\mathcal{S}$, he can switch to a strategy that guarantees $\mathsf{MP}\big(\mathsf{RT}(\mathcal{S}, 0.5)\big)$. Thus, similarly to Thm. 14, we construct a reachability game in which the targets for Max are his winning BSCCs and the targets for Min are her winning BSCCs. Membership in NP and coNP for Richman bidding is obtained by guessing the winning BSCCs and guessing positional strategies both in the reachability random-turn game and the mean-payoff random-turn games in the BSCCs. One can verify in polynomial time that the strategies are optimal and calculate the mean-payoff values that they achieve. Verifying that a BSCC is winning is then trivial.

For poorman and taxman bidding, the ideas are similar though more complicated. For a BSCC $\mathcal{S}$, we need to find a ratio $r(\mathcal{S})$ such that Max can guarantee a payoff greater than c with $r(\mathcal{S})$. The reachability game we construct is more involved: the game ends once a target $\mathcal{S}$ is reached, and Max wins iff his ratio is greater than $r(\mathcal{S})$. Recall that in reachability bidding games, the threshold ratios of the targets is $\mathtt{Th}(t_1) = 0$ and $\mathtt{Th}(t_2) = 1$, and every other vertex it is some "average" of two of its neighbors that depends on the bidding mechanism. The solution here in the intermediate vertices is the same average as in reachability bidding games with the corresponding bidding mechanism. The only change is in the targets; namely, the threshold ratio in a target $\mathcal{S}$ is $r(\mathcal{S})$.

We turn to the last point. Under Richman bidding, since the random-turn game is un-biased, for every vertex v with two neighbors v_1 and v_2, we can construct a linear program without finding which of the vertices is v^+ and which is v^-. Indeed, the linear program ensures that $\mathtt{Th}(v) = \frac{1}{2} \cdot (\mathtt{Th}(v_1) + \mathtt{Th}(v_2))$. A similar solution applies to mean-payoff Richman bidding games. On the other hand, for poorman and taxman bidding, since the random-turn

game is biased, we need to find which of the vertices is v^+ and which is v^-. Fortunately, it is shown in [5] that when the out-degree is 2, it suffices to solve an MDP rather than a stochastic game, hence the polynomial-time upper bound. ◀

Open problems

The bounds in Thm. 28 are not tight, thus Open problem 1 applies here as well. In addition, we state the following open problem.

▶ **Open problem 2** (The target-payoff problem). *Study the dual of the problem that is studied in Thm. 28: given a target payoff, find the minimal ratio that is required to guarantee it.*

▶ **Open problem 3** (ε-optimal vs. optimal strategies). *Consider a strongly-connected mean-payoff game $\mathcal{G}$ and an initial ratio r for Max. For all but Richman bidding, to prove Thm. 20, we construct* ε-optimal strategies *for Max, namely strategies that guarantee a payoff of $\mathsf{MP}(\mathcal{G}, r) - \varepsilon$, for every $\varepsilon > 0$. This is sufficient since the definition of mean-payoff (Def. 17) gives Min an advantage, thus an ε-optimal strategy for Min is obtained by relating Min with Max in the "dual" game in which the weights are multiplied by -1. We call a strategy* optimal *if it guarantees a payoff of $\mathsf{MP}(\mathcal{G}, r)$. This raises two open questions:*

- *Find optimal strategies for Max.*
- *Find optimal strategies for Min for poorman and taxman bidding.*[2]

Energy games. An *energy game* is played on a weighted directed graph. Min wins iff there is a prefix in which the energy hits 0. Energy games are qualitative games and so the main question regards the threshold ratio: given an initial vertex v and an initial energy level k, find the threshold ratio $\mathtt{Th}(v, k)$ that is necessary and sufficient for Min to win. Our solution to mean-payoff games has some implications on energy games. Consider a strongly-connected energy game $\mathcal{G}$. It is not hard to adapt Min's strategy in Lem. 23 to show that under Richman bidding, when $\mathsf{MP}(\mathsf{RT}(\mathcal{G}, 0.5)) = 0$, Min wins from every initial energy level and with any positive initial budget, thus $\mathtt{Th}(v, k) = 0$, for every $v \in V$ and $k \in \mathbb{N}$. On the other hand, if $\mathsf{MP}(\mathsf{RT}(\mathcal{G}, 0.5)) > 0$, then $\lim_{k\to\infty} \mathtt{Th}(v, k) = 1$, thus for every initial ratio $r > 0$, there is an initial energy level from which Max wins. A similar corollary holds for poorman and taxman bidding. In [23], the threshold ratios for the Bowtie game are classified under poorman bidding: for every $k \in \mathbb{N}$, we have $\mathtt{Th}(v, k) = \frac{k+2}{2k+2}$.

▶ **Open problem 4** (Energy bidding games). *Find solutions to other energy bidding games. For example, solving the Bowtie game with weights $+2$ and -1 under poorman bidding is open. The only thing we know is that when Max's budget exceeds $2/3$, there is an initial energy level from which he wins.*

4 All-Pay Bidding Games

Before describing the theoretical properties of all-pay bidding games, we illustrate applications of the model. Throughout the survey we always think of the players' budgets as money. Applications arise, however, from thinking of the budgets as bounded resources with little or no inherent value that the players use in order to achieve some goal. *Colonel Blotto* games, which date back to [12] and have been extensively studied since, have been used to

[2] Note that Lem. 24 constructs an optimal strategy for the Bowtie game under poorman bidding. We do not know how to extend this strategy to general strongly-connected games.

model such settings; e.g., rent seeking [32], patent races (e.g., [10]), political campaigning or lobbying, sport competitions, or biological auctions. The terminology there is that two colonels own armies and compete in one-shot in n battlefields; the colonels need to distribute their armies across the battlefields, in each battlefield the outnumbering army wins, and possible goals include maximizing the number of battlefields won or the probability that at least $k < n$ battlefields are won. In our terminology, the colonels are performing n concurrent biddings. It was argued in [21] that many of the examples above are actually dynamic in nature. We thus argue that all-pay poorman bidding games, which are a dynamic version of Colonel Blotto games, capture these settings in a more precise way.

Another application of bidding games is reasoning about systems in which the scheduler accepts payment in exchange for priority. Blockchain technology is one such system, where the *miners* accept payment and have freedom to decide the order of blocks they process based on the proposed transaction fees. Transaction fees are not refundable, thus all-pay poorman bidding is the most appropriate modelling. Manipulating transaction fees is possible and can have dramatic consequences: such a manipulation was used to win a popular game on Ethereum called FOMO3d[3]. There is thus ample motivation for reasoning and verifying blockchain technology [16].

Rechability all-pay poorman games

Reachability all-pay poorman bidding games were studied in [8]. The bad news is that reachability games under all-pay bidding are significantly more complicated than under first-price bidding. In light of the technical difficulty, it is encouraging that simple yet powerful results are obtained on this model. We focus mostly on the following class of games.

▶ **Definition 29** (Race games). *In the race game $G(i, j)$, for $i, j \in \mathbb{N}$, Player 1 wins iff he wins i biddings before Player 2 wins j biddings.*

▶ **Example 30.** Suppose for example that two NBA teams play a "best of 7" tournament. The resources (the athletes' strengths) need to be invested throughout the tournament so as to maximize the probability that the tournament is won. Note that the resources have no inherent value, that invested resources are exhausted rather than transferred between the teams, and that the budgets need not be equal (one the teams can be fresher or have a deeper bench). Thus, in order to find an optimal strategy to select how much to invest in each game, we solve the race game $G(4, 4)$ under all-pay poorman bidding. ⌟

The simplest interesting reachability game is $G(2, 1)$, in which Player 1 needs to win two biddings in a row. Throughout all of this section we fix Player 2's budget to 1. Under first-price bidding, the solution to this game is almost trivial: the threshold ratios are 2 and 3 under poorman and Richman bidding, respectively. Let B_1 be Player 1's budget. It was observed already in [23] that under all-pay poorman bidding, for $B_1 \in [1, 2]$, neither player has a deterministic winning strategy. We thus need the following definitions.

▶ **Definition 31.** *A* mixed *strategy assigns a probability distribution over bids in each step. The* lower value *in an all-pay game $\mathcal{G}$ w.r.t. an initial ratio r is $val^{\downarrow}(\mathcal{G}, r) = \sup_f \inf_g \int_{\pi \sim D(f,g)} \mathbb{P}[\pi$ is winning for Player 1$]$. The* upper value*, denoted $val^{\uparrow}(\mathcal{G}, r)$ is defined dually. It is always the case that $val^{\downarrow}(\mathcal{G}, r) \leq val^{\uparrow}(\mathcal{G}, r)$, and when $val^{\uparrow}(\mathcal{G}, r) = val^{\downarrow}(\mathcal{G}, r)$, we say that the value exists and we denote it by $val(\mathcal{G}, r)$.*

[3] `https://bit.ly/2wizwjj`

The following problem was left open in [23] already for $G(2,1)$:

Does the value exist in every vertex of every all-pay bidding game?
If it does, characterize the value.

We solve this problem for the game $G(2,1)$.

▶ **Theorem 32** ([8]). *Consider the all-pay bidding game $G(2,1)$ in which Player 1 needs to win twice in a row. The value exists for every pair of initial budgets, the optimal Player 1 strategy has finite support, and the value as a function of Player 1's budget is the following step-wise function: if Player 1's budget is B_1 and Player 2's budget is 1, then, when $B_1 > 2$, the value is 1, when $B_1 < 1$, the value is 0, and when $B_1 \in (1+\frac{1}{n+1}, 1+\frac{1}{n}]$, for $n \in \mathbb{N}$, then the value is $\frac{1}{n+1}$.*

Thm. 32 is positive in several aspects, however all-pay bidding games are not that simple.

▶ **Theorem 33** ([8]). *Consider the game $G(3,1)$ in which Player 1 needs to win three biddings in a row. Suppose the initial budgets are 1.25 for Player 1 and 1 for Player 2. Then, an optimal strategy for Player 1 requires infinite support in the first bidding.*

We turn to present simple yet powerful positive news about all-pay bidding games.

An approximation algorithm. We obtain an approximation algorithm for the upper- and lower-values in games played on DAGs. The algorithm is based on a discretization of the budgets similar in spirit to discrete bidding. That is, we restrict the granularity of the bids and require Player 1 to bid multiples of some $\varepsilon > 0$. The smaller the ε, the better the discrete game approximates the continuous game (see [8] for the formal guarantees). A discrete bidding game is a finite game (the smaller ε is, the larger the game). Finding values can thus be done in polynomial time using linear programming.

We implemented the algorithm and ran it on several race games, see the plots in Fig. 6. Each game is associated with two plots; a lower- and an upper-bound. It is encouraging that the difference between the two plots is barely visible already for $\varepsilon = 0.01$. That is, the plots hint that the lower- and upper-values coincide and that the value exists in the games that we experiment with. It is also reassuring to find an experimental confirmation for Thm. 32 (see the red plots). While the statement in Thm. 33 cannot be confirmed by the plots, the value function in the budget region of 1.25 seems involved (see the purple plots). For more complicated games, the plots hint that the value as a function of the budgets is continuous rather than step-wise as in $G(2,1)$.

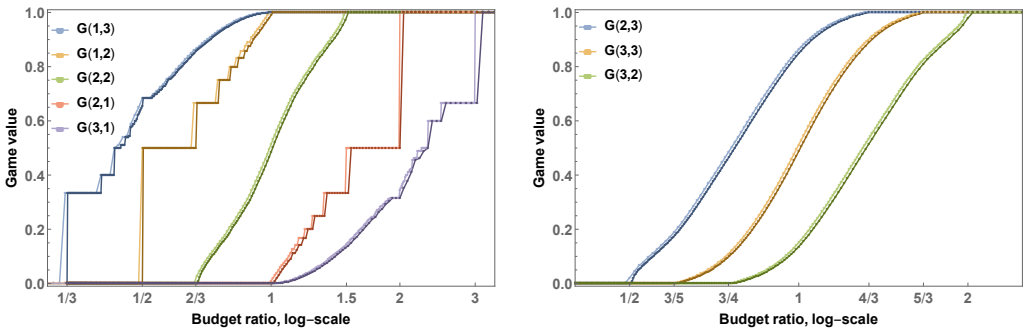


Figure 6 Approximations for the upper- and lower-values for several race games.

Surely-winning threshold ratios. we considering the question of *surely winning* in all-pay bidding games, namely winning with probability 1, and identify a threshold phenomena similar to first-price bidding.

▶ **Theorem 34** ([8]). *Consider a reachability bidding game $\mathcal{G}$ played on a general graph. Each vertex v in $\mathcal{G}$ has a* surely-winning threshold *ratio $ST_{HR}(v)$ such that, when Player 1's budget is B and Player 2's budget is 1:*

- *If $B > ST_{HR}(v)$, Player 1 has a deterministic surely-winning strategy.*
- *If $B < ST_{HR}(v)$, Player 2 wins with positive probability.*

Moreover, when the targets are t_1 and t_2, then $ST_{HR}(t_1) = 0$ and $ST_{HR}(t_2) = \infty$,[4] and for every other vertex v, we have $ST_{HR}(v) = ST_{HR}(v^-) + (1 - ST_{HR}(v^-)/ST_{HR}(v^+))$. Finding the thresholds is in PSPACE in general and in linear time in games played on DAGs.

In DAGs, surely-winning threshold ratios are rational numbers. In general graphs, however, Fig. 8 shows that this not the case. The algorithm for computing surely-winning threshold ratios in DAGs is a simple backwards induction. We implemented it and used it to solve the game tic-tac-toe. The solution of tic-tac-toe under first-price bidding is discussed in the blog post `https://bit.ly/2KUong4`: with first-price Richman bidding, Player 1 can guarantee winning when his ratio exceeds $133/123 \approx 1.0183$ [18] and with first-price poorman bidding, when it exceeds roughly 1.0184. Under all-pay poorman bidding, the surely-winning threshold ratio is $51/31 \approx 1.65$ (see Fig. 7). A reason for the slightly higher threshold in all-pay is that contrary to first-price bidding, there is a gap between the surely-winning thresholds of the two players; namely, in the range $(31/51, 51/31)$ neither player surely wins the game. For more unexpected phenomena that the solution points to, see [8].

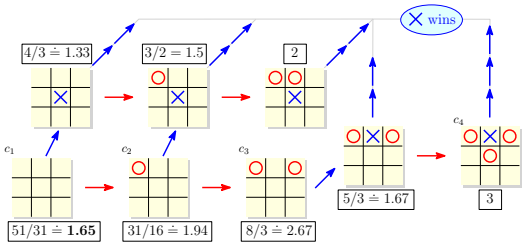


Figure 7 Surely-winning thresholds in all-pay poorman tic-tac-toe. For example, the surely-winning threshold budget in c_4 is 3 since in order to win the game, Player 1 must win three biddings in a row.

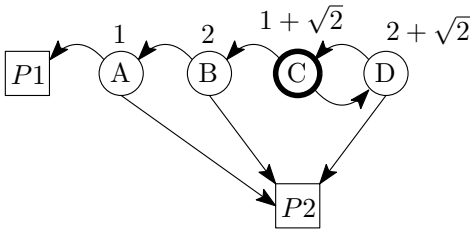


Figure 8 A reachability all-pay poorman game with target Pi for Player i in which surely-winning threshold ratios are irrational.

▶ **Open problem 5.** *We provide a complete picture for the game $G(2, 1)$. Solutions to more complicated games are left open. In particular:*

- *Show that the value exists in general.*
- *Can the value as a function of the budget ratio be a continuous function?*
- *Find closed-form solutions to other race games or other games on DAGs.*

[4] Recall that Player 2's budget is normalized to 1 hence the different values in the targets compared with first-price bidding.

Infinite-duration all-pay bidding games

Recall that even though reachability first-price poorman games do not exhibit an equivalence with random-turn games (Remark 12), an equivalence "pops up" in mean-payoff first-price poorman games (Thm. 20). Inspired by this result, we study the following question:

Are mean-payoff all-pay bidding games equivalent to random-turn games?

This question is studied in [9] and answered positively. Moreover, we find the results particularly surprising. We describe the key properties of mean-payoff games played on strongly connected graphs. Under all-pay Richman bidding, a simple argument shows that deterministic strategies cannot guarantee anything; e.g., in $\mathcal{G}_{\bowtie}$ (Fig. 3), Max cannot deterministically guarantee any positive payoff. On the other hand, with mixed strategies, all-pay and first-price Richman coincide: in both, the initial ratio does not matter, and the optimal *almost-sure* payoff a player can guarantee under all-pay Richman coincides with what he can guarantee deterministically under first-price Richman. In $\mathcal{G}_{\bowtie}$ this is 0.5.

Given these results, it seems safe to guess that poorman all-pay bidding would exhibit similar properties: deterministic strategies are useless and with mixed strategies, first-price and all-pay coincide. Both guesses, however, are wrong. Consider $\mathcal{G}_{\bowtie}$ and suppose Max's ratio is 0.75. As a baseline, recall that under first-price bidding, the mean-payoff value is 0.75. The first surprise is that deterministic strategies are useful under all-pay poorman bidding when the ratio is greater than 0.5; Max can deterministically guarantee a payoff of 2/3 with a ratio of 0.75. The real surprise is with mixed strategies: given a choice between first-price and all-pay bidding, Max strictly prefers all-pay bidding! In $\mathcal{G}_{\bowtie}$, with ratio 0.75, Max can almost-surely guarantee a payoff of 5/6.

5 Discrete Bidding Games

In discrete bidding, the budgets are given in coins, and the minimal positive bid a player can make is one coin. Discrete bidding is more natural with Richman bidding since with poorman bidding, the budgets run out eventually. Motivated by recreational play, reachability discrete bidding games were introduced and studied in [18] (see also [11, 22]). They showed existence of threshold ratios, i.e., an adaption of Thm. 7 to discrete bidding.

Unlike continuous bidding, in discrete bidding, ties in biddings need to be addressed explicitly. We thus assume a bidding game comes equipped with a tie-breaking mechanism. A *configuration* of a bidding game consists of the current vertex the token is placed on, the budgets of the two players, and the state of the tie-breaking mechanism. A game is *determined* if from each configuration exactly one player has a winning strategy. It is not hard to show that bidding games are a sub-class of concurrent games, which are not in general determined. For example, the game "matching pennies" is a non-determined concurrent game: both players concurrently choose a side of a coin, either 1 ("heads") or 0 ("tails"), and Player 1 wins iff the parity of the sum of the players' choices is 0. Matching pennies is not determined since if Player 1 reveals his choice first, Player 2 will choose opposite and win the game, and dually for Player 2.

In [1], we studied the following question:

Which tie-breaking mechanisms admit determinacy of discrete bidding games?

We consider the following classes of tie-breaking mechanisms.

▶ **Definition 35** (Tie-breaking mechanisms)**.**

- ***Transducer-based:*** *Several natural tie-breaking mechanisms can be expressed using a* transducer*. Examples include "Player* 1 *wins all ties", "the players alternate turns in winning ties", "tie breaking depends on the vertex on which the token is placed", "the player with less budget wins ties", and more. Formally, a transducer is* $T = \langle \Sigma, Q, q_0, \Delta, \Gamma \rangle$*, where* Σ *is a set of letters,* Q *is a set of states,* $q_0 \in Q$ *is an initial state,* $\Delta : Q \times \Sigma \to Q$ *is a deterministic transition function, and* $\Gamma : Q \to \{1, 2\}$ *is a labeling of the states. Intuitively,* T *is run in parallel to the bidding game and its state is updated according to the outcomes of the biddings. Whenever a tie occurs and* T *is in state* $s \in Q$*, the winner of the bidding is* $\Gamma(s)$*. A critical point is the information according to which tie-breaking is determined, and this is represented in the alphabet of the transducer.*
- ***Random-based:*** *A tie is resolved by choosing the winner uniformly at random.*
- ***Advantage-based:*** *Exactly one player holds the* advantage*. Suppose Player* i *holds the advantage and a tie occurs. Then Player* i *chooses who wins the bidding. If he calls the other player the winner, Player* i *keeps the advantage, and if he calls himself the winner, the advantage switches to the other player. This is the main tie-breaking mechanism studied in [18].*

The following simple example shows that bidding games are not in general determined with transducer-based tie-breaking. In [1], a more complicated example shows non-determinacy for a particularly simple tie-breaking mechanism: the players alternate turns in winning ties, that is, Player 1 wins the first tie, Player 2 the second tie, Player 1 the third, etc.

▶ **Example 36.** In a *Büchi game*, Player 1 wins a play iff it visits an accepting state infinitely often. Consider the Büchi bidding game that is depicted on the left of Fig. 9, and assume the players' budgets are positive and equal. We claim that the game is not determined when tie-breaking is resolved using the transducer on the right of the figure. That is, if a tie occurs in the first bidding, Player 2 wins all ties for the rest of the game, and otherwise Player 1 wins all ties. First note that, for $i \in \{1, 2\}$, no matter what the budgets are, if Player i wins all ties, he wins the game. A winning strategy for Player 1 always bids 0. Intuitively, Player 2 must invest a unit of budget for winning a bidding and leaving v_1. Thus, if Player 2's initial budget is B, he can leave v_1 only B times, and the game eventually stays in v_1. So, the winner is determined according to the outcome of the first bidding, and the players essentially play a matching-pennies game in that round. ⌟

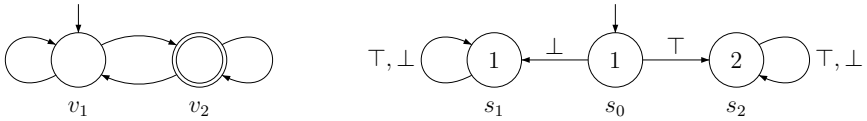


Figure 9 On the left, a Büchi game that is not determined when tie-breaking is determined according to the transducer on the right, where the letters $\top$ and $\bot$ respectively represent "tie" and "no tie".

The statement of the theorem below uses *Müller* objectives, which generalize parity objectives.

▶ **Theorem 37** ([1])**.** *Discrete bidding games are a determined sub-class of concurrent games in the following cases:*

- *Müller bidding games that use a transducer that is not aware of ties are determined.*
- *Reachability bidding games with random tie-breaking are determined.*
- *Müller bidding games with advantage-based tie-breaking are determined.*

Determinacy implies an immediate upper bound on solving discrete bidding games in which the budgets are given in unary.

▶ **Theorem 38** ([1]). *Let $\mathcal{G}$ be a discrete bidding game with objective γ and with either transducer or advantage-based tie-breaking. The complexity of deciding whether Player 1 wins from a given vertex with budgets N_1 and N_2 for the two players, given in unary, matches the complexity of solving a turn-based game with objective γ.*

▶ **Open problem 6.** *Our understanding of infinite-duration discrete bidding games is far from complete. We list specific directions for future work below:*

- *Adapt the theory of infinite-duration continuous bidding games to discrete bidding:*
  - *Recall that the key property of parity continuous-bidding games is that exactly one player wins a strongly-connected game (Lem. 13). This is not true for parity discrete-bidding games. Find a classification for strongly-connected parity discrete-bidding games that can serve a basis for a solution for games played on general graphs.*
  - *Mean-payoff discrete-bidding games were never studied. Can the equivalence with random-turn games in continuous-bidding games be adapted to discrete-bidding?*
- *Discrete bidding is especially suited for practical applications; find concrete applications for this model, e.g., in verifying blockchain technology (see Sec. 4).*
- *Preliminary results on discrete all-pay Richman bidding games were shown in [26] and it is interesting to continue this line of work.*
- *Study the computation complexity of discrete bidding games with budgets given in binary.*

6 Conclusions

We studied three orthogonal dimensions of bidding mechanisms: *who pays* (first-price vs. all-pay), *who is paid* (Richman, poorman, and taxman), and *what types of bids are allowed* (continuous vs. discrete). Our results depict a relatively complete picture, especially for continuous bidding. We believe that there is still ample room for future work on bidding games. In addition to the open problems mentioned throughout the survey, we discuss several orthogonal dimensions to the three mentioned above:

The underlying structure. Bidding games in this survey are always played on graphs. In [6], we established initial results on bidding games that have stochastic behavior: the game is played on an MDP and in each turn, we hold an auction to determine which action is chosen, and the next position of the token is determined according to a probability distribution that depends on the current vertex and the chosen action. Other possible structures to consider include real-time systems or systems with counters.

Objectives. We focused on three objectives: reachability, parity, and mean-payoff. It is interesting to study bidding games with other objectives. For example, in *discounted-sum* bidding games, the players are encouraged to bid higher in the first rounds due to the discounted payoffs. Another example is *quantitative reachability*, which is played on a weighted directed graph, the game ends once a target is reached, and the payoff is the (un-averaged) sum of the traversed weights.

One motivation to study bidding games is that they model ongoing auctions. In auctions, and different from mean-payoff objectives, the budget has a value to the players. We thus find it interesting to incorporate the budget in the payoff. For example, suppose that a play in a quantitative reachability game, reaches the target after traversing weights with a sum of

c, and leaving Player 1 with a budget of B. Then, we can define Player 1's payoff as $c + B$. That is, he gains a utility of c for the items he purchased and a utility of B from his left-over money. A similar definition can be applied to infinite plays like in mean-payoff games.

Non-zero-sum games. To the best of our knowledge, non-zero-sum bidding games were only studied in [25]. They consider two-player games on DAGs in which each leaf is labeled by two (not necessarily contradicting) payoffs that the players gain when the game ends in the leaf. They show existence of *subgame perfect equilibrium* in bidding games with out-degree 2. We believe that their results encourage further study of non-zero-sum bidding games.

Multi-player games. To the best of our knowledge, a combination of bidding games with multiple players has never been considered. While it is tempting to study non-zero-sum multi-player games, we note that it is interesting to study a zero-sum version in the spirit of [2]. For example, consider a reachability game with three players, where Player 1 has a target and Players 2 and 3 win iff Player 1 does not reach his target. We hold a first-price bidding in each round to determine who moves the token (say poorman bidding). The coalition can coordinate their actions, but they are restricted compared with a "traditional" player in a two-player bidding game: For example, if the budgets are $\langle 1.5, 1, 1\rangle$, even though the sum of budgets in the coalition is 2, the highest a member of the coalition can bid is 1, thus Player 1 can guarantee winning the bidding.

We mention a surprising result regarding this model. We say that $\overline{B} = \langle B_1, B_2, B_3\rangle \in \mathbb{R}^3$ is a *winning point* if Player 1 wins when Player i's budget is B_i, for $i \in \{1, 2, 3\}$. Ventsislav Chonev showed that the set $\{\overline{B} : \overline{B} \text{ is a winning point}\}$ is not convex in the game in which the coalition needs to win two biddings and Player 1 needs to win once.

Changes to the bidding mechanism. We list two examples of changes to the bidding mechanism, and there are plenty of other interesting changes one can study. It is appealing, especially in discrete poorman bidding, to add re-charging to the budgets. One way recharging can be defined is by labeling each vertex with a pair $\langle c_1, c_2\rangle \in \mathbb{N}^2$ and when the token lands on a vertex, we add c_i to Player i's budget, for $i \in \{1, 2\}$. So, Player i's available budget at the end of a finite play is his initial budget, minus the sum of bids he pays, plus the sum of recharges to his budget.

A second change to the bidding mechanism relaxes the requirement that the bidding mechanism applies in the same manner to all vertices in the graph. A simple definition would be to partition the vertices into vertices in which we hold a Richman bidding and those in which we hold a poorman bidding. For example, consider the Bowtie game (Fig. 3) and set one of the vertices to be Richman and the other to poorman. It is not clear whether Max prefers his vertex being the poorman vertex, the Richman vertex, or whether he has no preference. More generally, each vertex can be associated with its own taxman parameter.

References

1 M. Aghajohari, G. Avni, and T. A. Henzinger. Determinacy in discrete-bidding infinite-duration games. In *Proc. 30th CONCUR*, volume 140 of *LIPIcs*, pages 20:1–20:17, 2019.

2 R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

3 K.R. Apt and E. Grädel. *Lectures in Game Theory for Computer Scientists*. Cambridge University Press, 2011.

4 G. Avni, T. A. Henzinger, and V. Chonev. Infinite-duration bidding games. *J. ACM*, 66(4):31:1–31:29, 2019.

5 G. Avni, T. A. Henzinger, and R. Ibsen-Jensen. Infinite-duration poorman-bidding games. In *Proc. 14th WINE*, volume 11316 of *LNCS*, pages 21–36. Springer, 2018.

6 G. Avni, T. A. Henzinger, R. Ibsen-Jensen, and P. Novotný. Bidding games on markov decision processes. In *Proc. 13th RP*, pages 1–12, 2019.

7 G. Avni, T. A. Henzinger, and Đ. Žikelić. Bidding mechanisms in graph games. In *In Proc. 44th MFCS*, volume 138 of *LIPIcs*, pages 11:1–11:13, 2019.

8 G. Avni, R. Ibsen-Jensen, and J. Tkadlec. All-pay bidding games on graphs. *Proc. 34th AAAI*, 2020.

9 G. Avni, I. Jecker, and Đ. Žikelić. Infinite-duration all-pay bidding games. *CoRR*, abs/2005.06636, 2020. `arXiv:2005.06636`.

10 M. R. Baye and H. C. Hoppe. The strategic equivalence of rent-seeking, innovation, and patent-race games. *Games and Economic Behavior*, 44(2):217–226, 2003.

11 J. Bhatt and S. Payne. Bidding chess. *Math. Intelligencer*, 31:37–39, 2009.

12 E. Borel. La théorie du jeu les équations intégrales á noyau symétrique. *Comptes Rendus de l'Académie*, 173(1304–1308):58, 1921.

13 T. Brázdil, V. Brozek, K. Etessami, and A. Kucera. Approximating the termination value of one-counter mdps and stochastic games. In *Proc. 38th ICALP*, pages 332–343, 2011.

14 T. Brázdil, V. Brozek, K. Etessami, A. Kucera, and D. Wojtczak. One-counter markov decision processes. In *Proc. 21st SODA*, pages 863–874, 2010.

15 J. F. Canny. Some algebraic and geometric computations in PSPACE. In *Proc. 20th STOC*, pages 460–467, 1988.

16 K. Chatterjee, A. K. Goharshady, and Y. Velner. Quantitative analysis of smart contracts. In *Proc. 27th ESOP*, pages 739–767, 2018.

17 A. Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992.

18 M. Develin and S. Payne. Discrete bidding games. *The Electronic Journal of Combinatorics*, 17(1):R85, 2010.

19 A. E. Emerson, C. S. Jutla, and P. A. Sistla. On model-checking for fragments of μ-calculus. In *Proc. 5th CAV*, pages 385–396, 1993.

20 A. R. Howard. *Dynamic Programming and Markov Processes*. MIT Press, 1960.

21 K. A. Konrad and D. Kovenock. Multi-battle contests. *Games and Economic Behavior*, 66(1):256–274, 2009.

22 U. Larsson and J. Wästlund. Endgames in bidding chess. *Games of No Chance 5*, 70, 2018.

23 A. J. Lazarus, D. E. Loeb, J. G. Propp, W. R. Stromquist, and D. H. Ullman. Combinatorial games under auction play. *Games and Economic Behavior*, 27(2):229–264, 1999.

24 A. J. Lazarus, D. E. Loeb, J. G. Propp, and D. Ullman. Richman games. *Games of No Chance*, 29:439–449, 1996.

25 R. Meir, G. Kalai, and M. Tennenholtz. Bidding games and efficient allocations. *Games and Economic Behavior*, 2018. `doi:10.1016/j.geb.2018.08.005`.

26 M. Menz, J. Wang, and J. Xie. Discrete all-pay bidding games. *CoRR*, abs/1504.02799, 2015.

27 Y. Peres, O. Schramm, S. Sheffield, and D. B. Wilson. Tug-of-war and the infinity laplacian. *J. Amer. Math. Soc.*, 22:167–210, 2009.

28 Y. Peres and Z. Sunic. Biased infinity laplacian boundary problem on finite graphs. *CoRR*, abs/1912.13394, 2019. `arXiv:1912.13394`.

29 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.

30 M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 2005.

31 M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.

32 G. Tullock. *Toward a Theory of the Rent Seeking Society*, chapter Efficient rent seeking, pages 97–112. College Station: Texas A&M Press, 1980.

Safe Reinforcement Learning Using Probabilistic Shields

Nils Jansen
Radboud University, Nijmegen, The Netherlands

Bettina Könighofer
Graz University of Technology, Austria

Sebastian Junges
University of California, Berkeley, CA, USA

Alex Serban
Radboud University, Nijmegen, The Netherlands

Roderick Bloem
Graz University of Technology, Austria

Abstract

This paper concerns the efficient construction of a safety shield for reinforcement learning. We specifically target scenarios that incorporate uncertainty and use Markov decision processes (MDPs) as the underlying model to capture such problems. Reinforcement learning (RL) is a machine learning technique that can determine near-optimal policies in MDPs that may be unknown before exploring the model. However, during exploration, RL is prone to induce behavior that is undesirable or not allowed in safety- or mission-critical contexts. We introduce the concept of a probabilistic shield that enables RL decision-making to adhere to safety constraints with high probability. We employ formal verification to efficiently compute the probabilities of critical decisions within a safety-relevant fragment of the MDP. These results help to realize a shield that, when applied to an RL algorithm, restricts the agent from taking unsafe actions, while optimizing the performance objective. We discuss tradeoffs between sufficient progress in the exploration of the environment and ensuring safety. In our experiments, we demonstrate on the arcade game PAC-MAN and on a case study involving service robots that the learning efficiency increases as the learning needs orders of magnitude fewer episodes.

2012 ACM Subject Classification Computing methodologies → Reinforcement learning; Theory of computation → Verification by model checking; Theory of computation → Reinforcement learning; Computing methodologies → Markov decision processes

Keywords and phrases Safe Reinforcement Learning, Formal Verification, Safe Exploration, Model Checking, Markov Decision Process

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.3

Category Invited Paper

Supplementary Material `http://shieldrl.nilsjansen.org`

1 Introduction

Recent years showed increased use of reinforcement learning (RL) in solving tasks such as complex games [60] or robotic manipulation [67]. In RL, an agent perceives the surrounding environment and acts towards maximizing a long-term reward. A major open challenge for systems employing RL is the *safety* of decision-making [61, 30]. In particular during the exploration phase – when an agent chooses random actions in order to examine its surroundings – it is important to avoid actions that may cause unsafe outcomes. The area of *safe exploration* investigates how RL agents may be forced to adhere to safety requirements during this phase [54, 4].

A suite of methods that deliver theoretical guarantees are so-called *safety-shields* [15]. Shields prevent an agent from taking unsafe actions at runtime. To this end, the performance objective is extended with a constraint specifying that unsafe states should *never* be visited. This new safety objective ensures there are no violations during the exploration phase.

We propose to incorporate constraints that enforce safety violations to occur *only with small probability*. If an action increases the probability of a safety violation by more than a threshold δ with respect to the optimal safety probability, the shield blocks the action. Consequently, an agent augmented with a shield is *guided* to satisfy the safety objective during exploration (or as long as the shield is used). The shield is *adaptive* with respect to δ, as a high value for δ yields a stricter shield, a smaller value a more permissive shield. The value for δ can be changed on-the-fly, and may depend on the individual minimal safety probabilities at each state. Moreover, in case there is not suitable safe action with respect to δ, the shield can always pick the optimal action as a fallback.

We base our formal notion of a probabilistic shield on MDPs, which constitute a popular modeling formalism for decision-making under uncertainty [69] and is widely used in model-based RL. We assess safety by means of probabilistic *temporal logic constraints* [7] that limit, for example, the probability to reach a set of critical states in the MDP.

In order to assess the risk of one action, we (1) construct a behavior model for the environment using model-based RL [26]. By plugging this model into any concrete scenario, we obtain an MDP. To construct the shield, we (2) use a model-based verification technique known as *model checking* [23, 7] that assesses whether a system model satisfies a specification. In particular, we obtain precise *safety probabilities of any possible decision* within the MDP. These probabilities can be looked up efficiently and compared to the threshold δ. The shield then readily (3) augments either model-free or model-based RL.

We identify three key challenges. Firstly, model checking – as any model-based technique – is susceptible to scalability issues. A key advantage of using a separate safety objective is that we may analyze safety on just a fraction of the system, the *safety-critical MDP*. In our experiments, these MDP fragments are at least ten orders of magnitude smaller than a full model of the system, rendering model checking applicable to realistic scenarios. We introduce further optimizations based on problem-specific abstraction techniques.

Secondly, without randomness, all states are either absolutely safe or unsafe. However, in the presence of randomness, safety may be seen as a quantitative measure: in some states certain actions induce a large risk, while others may be considered *relatively* safe. Therefore, we propose an *adaptive* notion of shielding, in which the pre-selection of actions is not based on absolute thresholds.

Lastly, shielding may *restrict* exploration and lead to suboptimal policies. Therefore, it should not be considered in isolation. The trade-off between optimizing the performance objective and the achieved safety is intricate. Intuitively, accepting small short term risks may allow for efficient exploration and limit the risk long-term. To this end, we provide and discuss mechanisms that allow to adjust the shield based on such observations.

We apply shielding to two distinct use cases: the arcade game PAC-MAN and a new case study involving service robots in a warehouse. Shielded RL leads to improved policies for both case studies with fewer safety violations and performance superior to unshielded RL.

Related Work

Safety in RL

Safe RL [33, 54] is concerned with providing safety guarantees for learned agents. Our work focusses on the safe exploration [52], we refer to [33] for other types of safe RL. Using their taxonomy, shielding is an instance of "teacher provides advice" [24], where a teacher with additional information about the system guides the RL agent to pick the right actions. Apprenticeship learning [1] is a closely related variant where the teacher gives (positive) examples and has been used in the context of verification [72]. Some of the work does not assume a model for the environment, making the problem intrinsically harder – and often limiting the safety during exploration. We refer to [52, 31, 21, 32, 37, 36, 38] for some interesting approaches.

Shielding and non-probabilistic specifications

In the remainder of the work, we focus on related work that assumes that the (relevant) dynamics are known. Let us focus on discrete systems first. Exploiting the progress in reactive synthesis [13] provides a shield [44], a maximally permissive policy that contains all actions that will not violate the safety specification. This approach has been shown to be successful in combination with RL [3], and has some noticeable variants. In [68], the temporal specification is extended with an unknown reward function. To enforce complex timing behavior, shields from timed safety properties given as timed automata were considered in [14]. Finally, shielding multi-agent systems is considered in [11], and a shield for almost-sure specifications in partially observable MDPs is introduced in [41].

Shielding and probabilistic specifications

The difference and novel contribution of this paper with the aforementioned papers is rooted in the consideration of stochastic behavior, which is natural to RL. Intuitively, without stochasticities, a learning agent does not take any risk, which is unrealistic in most scenarios. Moreover, often one cannot assume that a 100% (or almost-sure) safety is realizable. A permissive policy for these cases is provided in [29]. Such policies can also be computed from abstract environments [51]. However, as there is no notion of maximally-permissive policies for probabilistic guarentees, multiple permissive policies need to be deployed [42]. Furthermore, the computation of these permissive policies is even more expansive than probabilistic model checking itself, harming scalability. A similar approach to ours was developed independently in [16], but targets a different application area and does not consider scalability issues of formal verification.

Continuous domains

In [71], shielding with qualitative guarantees for continuous domains is discussed. For probabilistic properties and continuous MDPs, additional assumptions are necessary to provide guarantees. Often, these assumption help make some ranking or barrier function [10, 53, 22, 34, 2], and have been extended to work with uncertain specifications [64]. `UPPAAL STRATEGO` provides a number of algorithms combining safety synthesis with optimizing RL for continuous space MDPs [25, 40].

Reinforcement learning in verification

The recent area of programmatic reinforcement learning aims to find (simple) programs rather than policies represented by deep neural networks, such that these programs can be verified [8, 66, 65]. These approaches can be seen as an extension to guided policy synthesis [50]. Such programs have also been used as shields [73]. More generally, Ashok et al. [6, 5] consider *post-processing* controllers into decision trees. Recently, also the verification of recurrent neural network controllers has been investigated [18]. These approaches do not aim to generate permissive shields, but do post-processing to provide guarantees about the final result. Simlarly, methods from reinforcement learning have been successfully employed to improve the scalability of verification methods for MDPs [17, 57, 45] or other areas of formal methods [20, 49].

2 Problem Statement

2.1 Foundations

A *probability distribution* over a countable set X is a function $\mu\colon X \to [0,1]$ with $\sum_{x\in X} \mu(x) = 1$. $\mathit{Distr}(X)$ denotes all distributions on X. The support of $\mu \in \mathit{Distr}(X)$ is $\mathit{supp}(\mu) = \{x \in X \mid \mu(x){>}0\}$.

▶ **Definition 1** (MDP). *A* Markov decision process *(MDP)* $\mathcal{M} = (S, \mathit{Act}, \mathcal{P}, r)$ *has a set* S *of* states*, a finite set* Act *of* actions*, a (partial)* probabilistic transition function $\mathcal{P}\colon S \times \mathit{Act} \to \mathit{Distr}(S)$*, and an* immediate reward function $r\colon S \times \mathit{Act} \to \mathbb{R}_{\geq 0}$*. For all* $s \in S$ *the available actions are* $\mathit{Act}(s) = \{\alpha \in \mathit{Act} \mid \mathcal{P}(s,\alpha) \neq \bot\}$ *and we assume* $|\mathit{Act}(s)| \geq 1$*.*

MDPs operate by means of nondeterministic *choices* of actions at each state. A *policy* for an MDP is a function $\sigma\colon S^* \to \mathit{Distr}(\mathit{Act})$ with $\mathit{supp}(\sigma(s_1 \dots s_n)) \subseteq \mathit{Act}(s_n)$ and S^* a finite sequence of states.

In formal methods, safety properties are often specified as *linear temporal logic* (LTL) properties [55]. For an MDP $\mathcal{M}$, probabilistic model checking [43, 47] employs value iteration or linear programming to compute the probabilities of *all states and actions of the MDP* to satisfy an LTL property φ. Specifically, we compute $\eta^{\max}_{\varphi,\mathcal{M}}\colon S \to [0,1]$ or $\eta^{\min}_{\varphi,\mathcal{M}}\colon S \to [0,1]$, which yields for all states the minimal (or maximal) probability over all possible policies to satisfy φ. For instance, for φ encoding to reach a set of states T, $\eta^{\max}_{\varphi,\mathcal{M}}(s)$ describes the maximal probability to "eventually" reach a state in T.

2.2 Setting

We define a setting where one controllable agent (the *avatar*) and a number of uncontrollable agents (the *adversaries*) operate within an *arena*. The arena is a compact, high-level description of the underlying model. From this arena, the potential states and actions of all agents may be inferred. For safety considerations, the reward structure can be neglected, effectively reducing the state space for our model-based safety computations. Formally, an *arena* is a directed graph $G = (V, E)$ with a finite sets V of nodes and $E \subseteq V \times V$ of edges. The agent's *position* is defined via the current node $v \in V$. The agent *decides* on a new edge $(v, v') \in E$ and determines its next position v'. Some (combinations of) agent positions are safety-critical, as they e.g., correspond to collisions or falling off a cliff. A safety property may describe reaching such positions, or use any other property expressible in (the safety fragment of) temporal logic.

While the underlying model for the arena suffices to specify the safe behavior, it is not sufficiently succinct to model the performance via rewards. Consider an edge that is safety-relevant, but the agent is only rewarded the first time taking this edge. In a flat model with rewards, two different edges are necessary to model this behavior. However, the reward (and thus the difference between these edges) is not needed to assess the safety, and the safety-relevant model may be pruned to an exponentially smaller model. We use a *token* function that implicitly extends the underlying model by a reward structure, enabling a separation of concerns between safety and performance.

Technically, we associate edges with a token function $\circ\colon E \to \{0,1\}$, indicating the status of an edge. Tokens can be (de-) activated and have an associated *reward* earned upon taking edges with an active token.

▶ **Example 2** (Autonomous driving)**.** An autonomous taxi (the avatar) operates within a road network encoded by an arena. The taxi has to visit several points to pick up or drop off passengers [28, 35]. Upon visiting such a point, a corresponding token activates and a reward is earned, afterwards the token is deactivated permanently. Meanwhile, the taxi has to account for other traffic participants or further environmental factors (the adversaries). A sensible safety specification may restrict the probability for collision with other cars to 0.5%. Note that the token structure is not relevant for such a specification.

▶ **Example 3** (Robot logistics in a smart factory)**.** Take a factory floor plan with several corridors with machines. The adversaries are (possibly autonomous) transporters moving parts within the factory. The avatar models a specific service unit moving around and inspecting machines where an issue has been raised (as indicated by a token), while accounting for the behavior of the adversaries. Corridors might be to narrow for multiple (facing) robots, which poses a safety critical situation. The tokens allow to have a *state-dependent* cost, either as long as they are present (indicating the costs of a broken machine) or for removing the tokens (indicating costs for inspecting the machine). A similar scenario has been investigated in [12].

2.3 Problem

Consider an environment described by an arena as above and a safety specification. We assume stochastic behaviors for the adversaries, e.g, obtained using RL [58, 59] in a training environment. In fact, this stochastic behavior determines all actions of the adversaries via probabilities. The underlying model is then an MDP: the avatar executes an action, and upon this execution the next exact positions (the state of the system) are determined stochastically.

We compute a δ-shield that prevents avatar decisions that violate this specification by more than a threshold δ with respect to the optimal safety probability. We evaluate the shield using a model-based or model-free RL avatar that aims to optimize the performance. The shield therefore has to handle an intricate tradeoff between strictly focussing on (short and midterm) safety and performance.

3 Constructing Shields for MDPs

We outline the workflow of our approach in Fig. 1, and describe it below. We employ a separation of concerns between the model-based shield construction and potentially model-free reinforcement learning (RL). First, we construct a *behavior model* for each adversary. Based on this model and a concrete arena, we construct a compact MDP model: the *safety-relevant MDP quotient*. In this MDP, we compute the *shield* which enables safe RL for the full MDP. We now detail the individual technical steps to realize our proposed method.

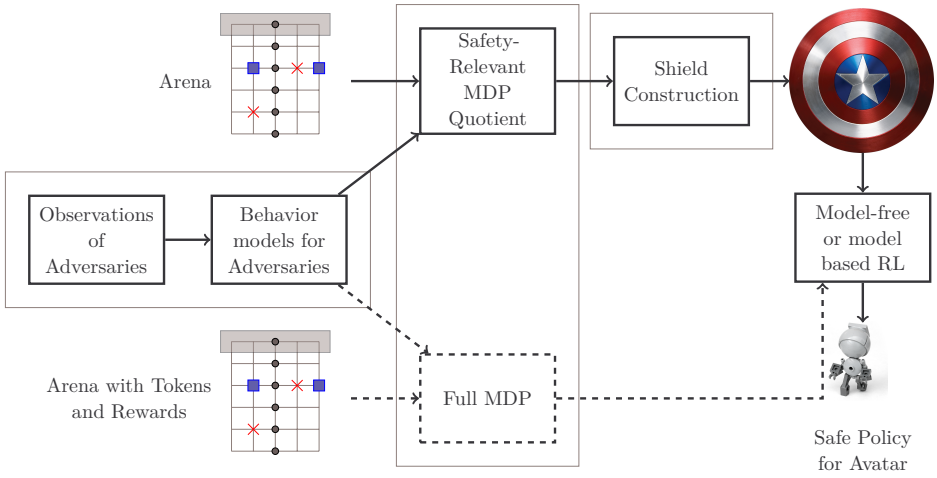


Figure 1 Workflow of the Shield Construction.

3.1 Behavior Models for Adversaries

We learn an adversary model by observing behavior in a set of similar, but smaller arenas, until we gain sufficient confidence that more training data would not change the behavior significantly [58]. To reduce the size of the training set, we devise a data augmentation technique using domain knowledge of the arenas [46, 70]. In particular, we abstract away from the precise configuration of the arena by partitioning the graph into zones that are relative to the view-point of the adversary (e. g., near or far, north or south, east or west). The intuitive assumption is that the specific position of an adversary is not important, but some key information is (e.g., the relation to the position of the avatar). This approach (1) speeds up the learning process and (2) renders the resulting behavior model applicable for varying the concrete instance of the same setting.

Zones are uniquely identified by a coloring with a finite set C of colors. Formally, for an arena $G = (V, E)$, *zones relative to a node* $v \in V$ are given by a function $z_v \colon V \to C$. For nodes $x, y \in V$, with $z_v(x) = z_v(y)$, the assumption is that the adversary in v behaves similarly regardless whether the avatar is in x or y. From our observations, we extract a *histogram* $h \colon E \times C \to \mathbb{N}$, where $h(e, c)$ describes how often the adversary takes an edge $e = (v, v') \in E$ while the avatar is in a node u with $z_v(u) = c$. We translate these likelihoods into distributions over possible edges in the arena.

▶ **Definition 4** (Adversary Behavior). *For an arena $G = (V, E)$, zones $z_u \colon V \to C$ for every $u \in V$, and a histogram $h \colon E \times C \to \mathbb{N}$, the* adversary behavior *is a function $B \colon V \times C \to Distr(E)$ with*

$$B(v, c) = \frac{h\big((v, v'), c\big)}{\sum_{(v,v') \in E} h\big((v, v'), c\big)} \ .$$

While we employ a simple normalization of likelihoods, alternatively one may also utilize, e. g., a softmax function which is adjustable to favor more or less likely decisions [62].

3.2 Safety-Relevant Quotient MDP

The construction of the MDP $\mathcal{M} = (S, Act, \mathcal{P})$ augments an arena by behavior models B_i. First, the *states* $S = V^{m+1} \times \{0, \ldots, m\}$ encode the positions for all agents and whose turn it is. The *decision states* of the safety-relevant MDP $\mathcal{M}$ are $S_d = \{s_d \in S \mid s_d = (\ldots, 0)\}$,

i.e., it's the turn of the avatar. The *actions* $Act = \{\alpha_0\} \cup Act_E$ with $Act_E = \{\alpha_e \mid e \in E\}$ determine the movements of the avatar and the adversaries. For $(v, \dots, 0) = s_d \in S_d$ (the avatar moves next), the available actions are $\alpha_e \in Act(s_d) \subseteq Act_e$, where α_e corresponds to an outgoing edge of v. For $(v, \dots, 0) = s_d \in S_d$, α_e with $e = (v, v')$ leads with probability one to a state $s_e = (v', \dots, 1)$. For $(v, \dots, v_i, \dots, i > 0)$ (an adversary moves next), there is a unique action α_0 where v_i is changed to v_i', randomly determined according to the behavior B_i, which also updates i to $i + 1$ modulo m. These transitions induce the only probabilistic choices in the MDP. A policy only has to choose an action at decision states. At all other states, only the unique action α_0 emanates. Consequently, a policy for $\mathcal{M}$ is a policy for the avatar.

Under the assumption that the reward function is known and not discovered during the exploration of the MDP, one can build the full MDP for the arena (V, E) and the token function $\circ\colon E \to \{0, 1\}$. Then, one can compute the reward-optimal and safe policy without need for further learning techniques. As there are 2^E token configurations, the state space blows up exponentially, which prevents the successful application of model checking or planning techniques for anything but very small examples.

3.3 Shield Construction

For the safety-relevant MDP $\mathcal{M}$, a set of unsafe states $T \subseteq S$ should preferably not be reached from any state. The property $\varphi = \Diamond T$ encodes the **violation** of this safety constraint, that is, eventually reaching T within $\mathcal{M}$. The shield needs to limit the probability to *satisfy* φ. We evaluate all decision states $s_d \in S_d$ with respect to this probability: We compute $\eta^{\min}_{\varphi,\mathcal{M}}(s_e)$, i.e., the minimal probability to satisfy φ from s_e, which is the state reached after taking action $\alpha_e \in Act_e$ in s_d.

▶ **Definition 5** (Action-valuation). *An* action-valuation *for each available action $\alpha_e \in Act_e$ at each decision state $s_d \in S_d$ is given by*

$$val^{\mathcal{M}}_{s_d}\colon Act(s_d) \to [0, 1], \text{ with } val^{\mathcal{M}}_{s_d}(\alpha_e) = \eta^{\min}_{\varphi,\mathcal{M}}(s_e)\ .$$

The optimal action-value *for s_d is $optval^{\mathcal{M}}_{s_d} = \min_{\alpha' \in Act} val^{\mathcal{M}}_{s_d}(\alpha')$, the set of all action-valuations at s_d is $ActVals_{s_d}$.*

We now define a shield for the safety-relevant MDP $\mathcal{M}$ using the action values. Specifically, a *δ-shield* for $\delta \in [0, 1]$ determines a set of actions at each decision state s_d that are δ-optimal for the specification φ. All other actions are "shielded" or "blocked".

▶ **Definition 6** (Shield). *For action-valuation $val^{\mathcal{M}}_{s_d}$ and $\delta \in [0, 1]$, a* δ-shield for state $s_d \in S_d$ *is given by*

$$shield^{s_d}_{\delta}\colon ActVals_{s_d} \to 2^{Act(s_d)}$$

with $shield^{s_d}_{\delta} \mapsto \{\alpha \in Act(s_d) \mid \delta \cdot val^{\mathcal{M}}_{s_d}(\alpha) \leq optval^{\mathcal{M}}_{s_d}\}$.

Intuitively, δ enforces a constraint on actions that are acceptable with respect to the optimal probability. The shield is *adaptive* with respect to δ, as a high value for δ yields a stricter shield, a smaller value a more permissive shield. The shield is stored using a lookup-table, and the value for δ can then be changed on-the-fly. In particularly critical situations, the shield can enforce the decision-maker to resort to (only) the optimal actions w.r.t. the safety objective. A δ-shield for the MDP $\mathcal{M}$ is built by constructing and applying δ-shields to all decision states.

▶ **Definition 7** (Shielded MDP). *The* shielded MDP $\mathcal{M}_{\text{⛉}} = (S, Act\,\mathcal{P}_{\text{⛉}})$ *for a safety-relevant quotient MDP* $\mathcal{M} = (S, Act, \mathcal{P})$ *and a* δ*-shield for all* $s_d \in S_d$ *is given by the transition probability* $\mathcal{P}_{\text{⛉}}$ *with* $\mathcal{P}_{\text{⛉}}(s, \alpha) = \mathcal{P}(s, \alpha)$ *if* $\alpha \in shield_\delta^s(val_s^{\mathcal{M}})$ *and* $\mathcal{P}_{\text{⛉}}(s, \alpha) = \bot$ *otherwise.*

▶ **Lemma 8.** *If MDP* $\mathcal{M}$ *is deadlock-free if and only if the shielded MDP* $\mathcal{M}_{\text{⛉}}$ *is deadlock-free.*

We compute the shield relative to optimal values $optval_{s_d}^{\mathcal{M}}$. Consequently, for $\delta = 1$, only optimal actions are preserved, and for $\delta = 0$ no actions are blocked.

▶ **Theorem 9.** *For an MDP* $\mathcal{M}$ *and a* δ*-shield, it holds for any state* s *that* $val_s^{\mathcal{M}} = val_s^{\mathcal{M}_{\text{⛉}}}$.

As optimal actions for the safety objective are not removed, optimality w.r.t. safety is preserved in the shielded MDP. Thus, during construction of the shield, we compute the action-valuations in fact *for the shielded MDP*. Observe that computing a shield for a state is done *independently* from the application of the shield to other states.

3.4 Guaranteed Safety

A δ-shield ensures that only actions that are δ-optimal with respect to an LTL property φ are allowed. In particular, for each action $\alpha \in Act_e$ at state s_e, we use the *minimal* probability $\eta_{\varphi,\mathcal{M}}^{\min}(s_e)$ to satisfy φ, see Def. 5. Under *optimal* (subsequent) choices, the value $\eta_{\varphi,\mathcal{M}}^{\min}(s_e)$ will be achieved. In contrast, a sequence of bad choices may violate φ with high probability. A more conservative notion would be to use the minimal action value while assuming that in all subsequent states the worst-case decisions corresponding to the maximal probabilities are taken. These values are computable by model checking. Regardless of subsequent choices, at least $val_{s_d}^{\mathcal{M}}(\alpha_e)$ is then guaranteed. A sensible notion to construct a shield would then be to impose a threshold $\lambda \in [0, 1]$ such that only actions with $val_{s_d}^{\mathcal{M}}(\alpha_e) \leq \lambda$ are allowed. A shield with such a guaranteed safety probability may induce a shielded MDP (Def. 7) that is *not deadlock free*. Moreover, the shield may become too restrictive for the agent.

3.5 Scalable Shield Construction

Although we apply model checking only in the safety-relevant MDP, scalability issues for large applications remain. We employ several optimizations towards computational tractability.

Finite Horizon

For infinite horizon properties, the probability to violate safety (in the long run) is often one in our examples. Furthermore, our learned MDP model is inherently an approximation of the real world. Errors originating from this approximation accumulate for growing horizons. Thus, we focus on a finite horizon such that the action values (and consequently, a policy for the avatar) carry only guarantees for the next steps. This assumption also allows us to prune the safety-relevant MDP (see below), increasing the scalability.

Piecewise Construction

Computing a shield for all states in an MDP concurrently yields a large memory footprint. To alleviate this footprint, we compute the shield states independently, in accordance with Theorem 9. The independent computation prunes the relevant part of the MDP, as the number of states reachable within the horizon is drastically reduced. Additionally, the independent computation allows for parallelizing the computation.

Independent Agents

The explosion of state spaces stems mostly from the number of agents. Here, an important observation is that we can consider agents independently. For instance, the probability for the avatar to crash with an adversary is stochastically independent from crashing with the others. Instead of determining the shield for all adversaries at once, we perform computations for each agent individually, and combine them via the inclusion-exclusion principle. Afterwards, the shield is composed from the shields dedicated to individual adversaries.

Abstractions

We observe that for finite horizon properties and piecewise construction, adversaries may be far away – beyond the horizon – without a chance to reach the avatar. We do not need to consider such (positions for) adversaries, as in these states, the shield will not block any actions.

Fewer Decision States. Depending on the setting, there might be only a few critical situations in which the agent requires shielding to ensure safety. The shield can be computed for this critical states only. Consequently, the agent makes shielded decisions in the adapted decision states, and unshielded decisions in all other ones.

Shielding versus Performance. A shield which is *minimally invasive* gives the RL agent the most freedom to optimize the performance objective. We propose two methods to alleviate invasiveness, all of them assume *domain knowledge* of the rationale behind the decision procedure.

Iterative Weakening. During runtime, we may observe that the progress of the avatar regarding the performance objective is not increasing anymore. Then, we weaken the shield by $\delta - \varepsilon$, allowing additional actions. As soon as progress is made, we reset δ to its former value. The adaption of $\mathit{shield}_{\delta}^{s}$ to $\mathit{shield}_{\delta-\varepsilon}^{s}$ can be done on the fly, without new computations.

Adapted Specifications. If the goal of the decision maker is known *and* can be captured in temporal logic, we may adapt the original specification accordingly. There are often natural trade-offs between safety and performance. These trade-offs might be resolved via weights, but this process is often undesirable [56] and similar to reward engineering. Instead, optimizing the conditional performance while assuming to stay sufficiently safe [63], avoids side-effects of attaching some weights to the safety specification.

4 Implementation and Numerical Experiments

4.1 Set-up

We run experiments using an Intel Core i7-4790K CPU with 16 GB of RAM using 4 cores. We give the timing results for a single CPU. Since the shield may be computed in a multi-threaded architecture, this time can be divided by the number of cores available.

The supplementary materials, namely the source code and videos are available online[1].

We demonstrate the applicability of our approach by means of two case studies. For both case studies, we learn the adversary behavior in small arenas, individually for each adversary. These behavior models are applicable to any benchmark instance, as they are independent of concrete positions.

[1] `http://shieldrl.nilsjansen.org`

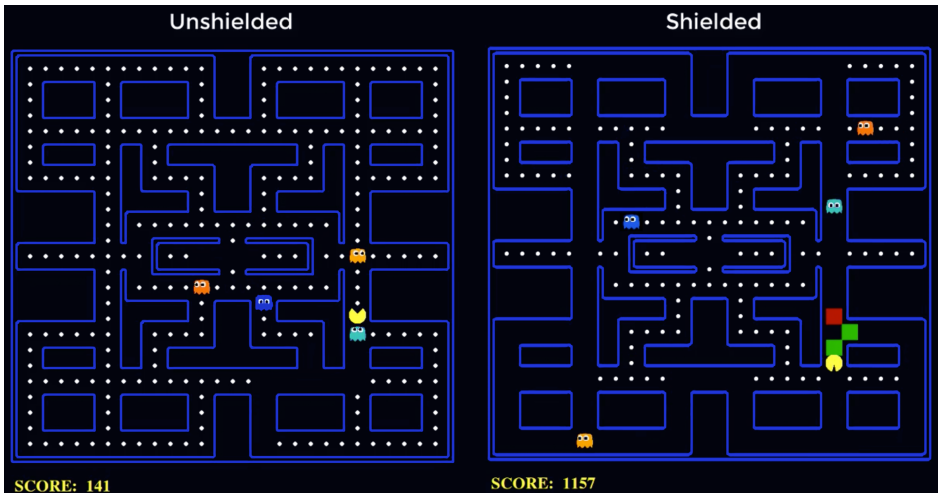


Figure 2 Video still for classic PAC-MAN.

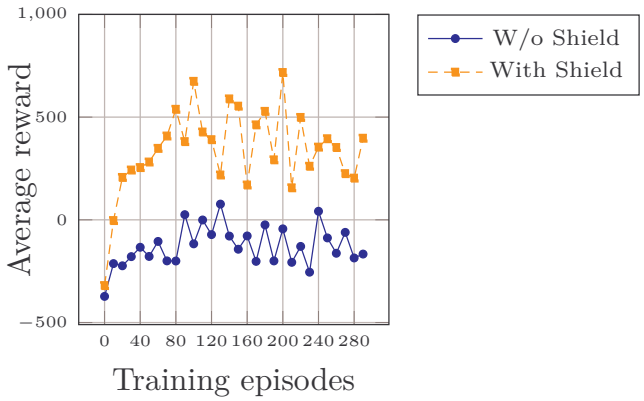


Figure 3 Training scores classic PAC-MAN.

For the arcade game PAC-MAN, PM (the avatar) aims to collect *PAC-dots* in a *maze* and not get caught by *ghosts* (the adversaries). We model various instance of the game (with different sizes) as an arena, where tokens represent the dots at each position in the maze, such that a dot is either present or collected. The score (reward, performance) is positively affected (+10) by collecting a dot and negatively by time (each step: -1). If PM either collects all dots (+500) or is caught (-500), the game is restarted. RL approaches exist [9], but they suffer from the fact that during the exploration phase PM is often caught by the ghosts, achieving very poor scores. The safety specification places a lower bound on the probability of reaching states in the underlying MDP that correspond to being caught.

We also consider a warehouse floor plan with several corridors. A similar scenario has been investigated in [12]. In the arena, nodes describe crossings, the edges the corridors with shelves, and the distances the corridor length. The agents are fork-lift units picking up packages from the shelves and delivering them to the exit; tokens represent the presence of a package at its position. The avatar is a specific (yellow) fork-lift unit that has to account for other units, the adversaries. The performance (reward) is positively affected by loading and delivering packages (+20, respectively) and negatively by time (each step: -1). Delivering all

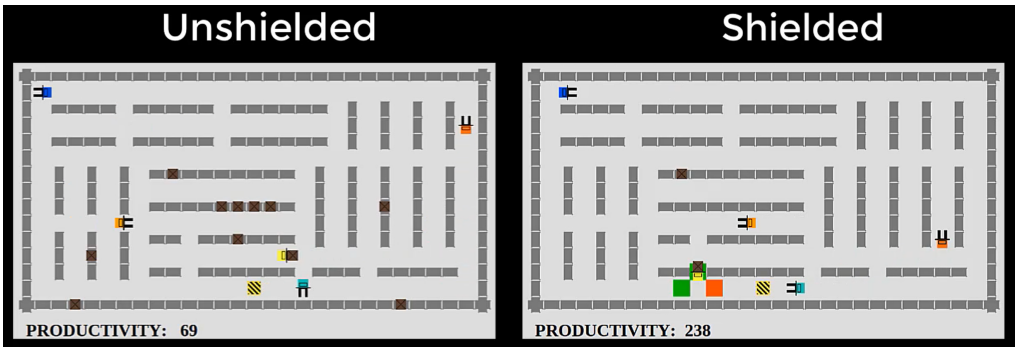


Figure 4 Video still for warehouse.

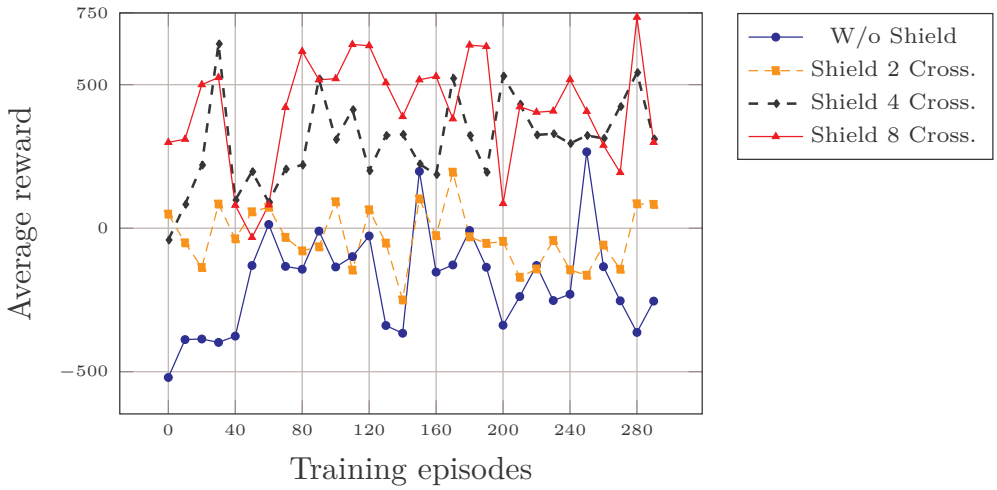


Figure 5 Training scores warehouse.

packages yields a large bonus (+500) and a collision leads to a large punishment (-500), both cases end the scenario. Corridors might be too narrow for multiple (facing) units, which poses a safety-critical situation. Most crucial is the crowded area near the exit, since all units have to deliver the packages to the exit.

Transferring the stochastic adversary behavior to any arena (without tokens) yields a concrete safety-relevant MDP. In particular, we specify an arena with the positions of the avatar and the adversaries as well as their behavior in the high-level `PRISM`-language [48]. We employ a script that automatically generates arenas to enable a broad set of benchmarks. Taking, e.g., the PAC-MAN arena from Fig. 2, the considered MDP has roughly 10^{12} states (compared to 10^{50} for the full MDP). For a safety-relevant MDP, we compute a δ-shield (with iterative weakening) via the model checker `Storm` [27], using a horizon of 10 steps. The immense size even of safety-relevant MDPs requires optimizations such as a piecewise and independent shield construction. Moreover, a multi-threaded architecture lets us construct shields for very large examples. In particular, we perform model checking for (many) MDPs of roughly 10^6 states. The computation time for the largest PM instance takes about 6 hours (single-threaded), while memory is not an issue due to the piecewise shield construction.

Table 1 Average scores and win rates for PM.

Size, #Ghosts	#Model Checking	time (s)	Score w/o Shield	Score w. Shield	Win Rate w/o Shield	Win Rate w. Shield
9x7,1	5912	584	-359,6	535,3	0,04	0,84
17x6,2	5841	1072	-195,6	253,9	0,04	0,4
17x10,3	51732	3681	-220,79	-40,52	0,01	0,07
27x25,4	269426	19941	-129,25	339,89	0,00	0,00

We compare RL to shielded RL on different instances. The key comparison criterion is the performance (detailed above) during learning. Our implementation is based on an existing PAC-MAN environment[2] using an approximate Q-learning agent [62] with the following feature vectors:

- for PAC-MAN: (1) distance to the closest dot, (2) whether a ghost collision is imminent, and (3) whether a ghost is one step away.
- for Warehouse: (1) has the unit loaded or unloaded, (2) the distance to the next package and (3) to the exit, (4) whether another unit is three steps away and (5) one step away.

The results are basic reflex controllers. The Q-learning uses the learning rate $\alpha = 0.2$ and the discount factor $\gamma = 0.8$ for the Q-update and an ϵ-greedy exploration policy with $\epsilon = 0.05$. One episode lasts until either the game is restarted. We describe results for the training phase of RL (300 episodes).

4.2 Results

Figures 2 and 4 show screenshots of a series of recommended videos (available in the supplementary material). Each video compares how RL performs either shielded or unshielded on a instance of the case study. In the shielded version, at each decision state in the underlying MDP, we indicate the risk of decisions from low to high by the colors green, orange, red.

Consider PAC-MAN in detail: Figure 3 depicts the scores obtained during RL. The curves (blue, solid: unshielded, orange, dashed: shielded) show the average scores for every ten training episodes. Table 1 shows results for instances in increasing size. We list the number of model checking calls and the time to construct the shield. We list the scores with and without shield, and the *winning rate* capturing the ratio of successfully ended episodes. For all instances, we see a large difference in scores due to the fact that PM is often rescued by the shield. The winning rates differ for most benchmarks, favoring shielded RL. For three or four ghosts, a shield with a ten-step horizon cannot guide PM to avoid being encircled by the ghosts long enough to successfully end the game. Nevertheless, the shield often safes PM, leading to superior scores. Moreover, the shield helps learning an optimal policy much faster as fewer restarts are needed.

For the warehouse case study, we choose to vary the decision states, i.e., the positions of the avatar for which we compute a shield. We present results for shielding the 2–8 crossings closest to the exit. Figure 5 shows the average score for the different variants, Table 2 summarizes average score and win rate. Unsurprisingly, the score gets better the more states are shielded. Furthermore, we have seen that shielding even more states has only a very limited effect.

[2] `http://ai.berkeley.edu/project_overview.html`

Table 2 Average scores and win rates for warehouse.

Crossings shielded	0	2	4	8
Score	-186	-27.6	303	420
Win Rate	0.16	0.31	0.59	0.71

5 Conclusion and Future Work

We developed the concept of shields for MDPs. Utilizing probabilistic model checking, we maintained probabilistic safety measures during reinforcement learning. We addressed inherent scalability issues and provided means to deal with typical trade-off between safety and performance. Our experiments showed that we improved the state-of-the-art in safe reinforcement learning.

For future work, we will extend the applications to more arcade games and employ deep recurrent neural networks as means of decision-making [39, 19, 18]. Another interesting direction is to explore (possibly model-free) learning of shields, instead of employing model-based model checking.

References

1 Pieter Abbeel and Andrew Y. Ng. Exploration and apprenticeship learning in reinforcement learning. In *ICML*, volume 119 of *ACM International Conference Proceeding Series*, pages 1–8. ACM, 2005.

2 Anayo K. Akametalu, Shahab Kaynama, Jaime F. Fisac, Melanie Nicole Zeilinger, Jeremy H. Gillula, and Claire J. Tomlin. Reachability-based safe learning with gaussian processes. In *CDC*, pages 1424–1431. IEEE, 2014.

3 Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *AAAI*, pages 2669–2678. AAAI Press, 2018.

4 Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *CoRR*, abs/1606.06565, 2016. `arXiv:1606.06565`.

5 Pranav Ashok, Mathias Jackermeier, Pushpak Jagtap, Jan Kretínský, Maximilian Weininger, and Majid Zamani. dtcontrol: decision tree learning algorithms for controller representation. In *HSCC*, pages 30:1–30:2. ACM, 2020.

6 Pranav Ashok, Jan Kretínský, Kim Guldstrand Larsen, Adrien Le Coënt, Jakob Haahr Taankvist, and Maximilian Weininger. SOS: safe, optimal and small strategies for hybrid markov decision processes. In *QEST*, volume 11785 of *Lecture Notes in Computer Science*, pages 147–164. Springer, 2019.

7 Christel Baier and J.P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

8 Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *NeurIPS*, pages 2499–2509, 2018.

9 UC Berkeley. Intro to AI – Reinforcement Learning , 2018. URL: `http://ai.berkeley.edu/reinforcement.html`.

10 Felix Berkenkamp, Matteo Turchetta, Angela Schoellig, and Andreas Krause. Safe model-based reinforcement learning with stability guarantees. In *NIPS*, pages 908–919, 2017.

11 Suda Bharadwaj, Roderick Bloem, Rayna Dimitrova, Bettina Könighofer, and Ufuk Topcu. Synthesis of minimum-cost shields for multi-agent systems. In *ACC*, pages 1048–1055. IEEE, 2019.

12 Arthur Bit-Monnot, Francesco Leofante, Luca Pulina, Erika Ábrahám, and Armando Tacchella. Smartplan: a task planner for smart factories. *CoRR*, abs/1806.07135, 2018. `arXiv:1806.07135`.

13 Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. Graph games and reactive synthesis. In *Handbook of Model Checking*, pages 921–962. Springer, 2018.

14 Roderick Bloem, Peter Gjøl Jensen, Bettina Könighofer, Kim Guldstrand Larsen, Florian Lorber, and Alexander Palmisano. It's time to play safe: Shield synthesis for timed systems. *CoRR*, abs/2006.16688, 2020. `arXiv:2006.16688`.

15 Roderick Bloem, Bettina Könighofer, Robert Könighofer, and Chao Wang. Shield synthesis: - runtime enforcement for reactive systems. In *TACAS*, volume 9035 of *LNCS*, pages 533–548. Springer, 2015.

16 Maxime Bouton, Jesper Karlsson, Alireza Nakhaei, Kikuo Fujimura, Mykel J. Kochenderfer, and Jana Tumova. Reinforcement learning with probabilistic guarantees for autonomous driving. *CoRR*, abs/1904.07189, 2019. `arXiv:1904.07189`.

17 Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Vojtěch Forejt, Jan Křetínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of Markov decision processes using learning algorithms. In *ATVA*, 2014.

18 Steven Carr, Nils Jansen, and Ufuk Topcu. Verifiable rnn-based policies for pomdps under temporal logic constraints. *CoRR*, abs/2002.05615, 2020. `arXiv:2002.05615`.

19 Steven Carr, Nils Jansen, Ralf Wimmer, Alexandru Constantin Serban, Bernd Becker, and Ufuk Topcu. Counterexample-guided strategy improvement for pomdps using recurrent neural networks. In *IJCAI*, pages 5532–5539. ijcai.org, 2019.

20 Jia Chen, Jiayi Wei, Yu Feng, Osbert Bastani, and Isil Dillig. Relational verification using reinforcement learning. *Proc. ACM Program. Lang.*, 3(OOPSLA):141:1–141:30, 2019.

21 Richard Cheng, Gábor Orosz, Richard M Murray, and Joel W Burdick. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. *AAAI*, 2019.

22 Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh. A Lyapunov-based approach to safe reinforcement learning. In *NIPS*, pages 8103–8112, 2018.

23 Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. MIT Press, 2001.

24 Jeffery A. Clouse and Paul E. Utgoff. A teaching method for reinforcement learning. In *ML*, pages 92–110. Morgan Kaufmann, 1992.

25 Alexandre David, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marius Mikucionis, and Jakob Haahr Taankvist. Uppaal stratego. In *TACAS*, volume 9035 of *LNCS*, pages 206–211. Springer, 2015.

26 Peter Dayan and Yael Niv. Reinforcement learning: the good, the bad and the ugly. *Current opinion in neurobiology*, 18(2):185–196, 2008.

27 Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *CAV (2)*, volume 10427 of *LNCS*, pages 592–600. Springer, 2017.

28 Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

29 Klaus Dräger, Vojtech Forejt, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Permissive controller synthesis for probabilistic systems. *Log. Methods Comput. Sci.*, 11(2), 2015.

30 Richard G Freedman and Shlomo Zilberstein. Safety in AI-HRI: Challenges complementing user experience quality. In *AAAI Fall Symposium Series*, 2016.

31 Jie Fu and Ufuk Topcu. Probably approximately correct mdp learning and control with temporal logic constraints. In *RSS*, 2014.

32 Nathan Fulton and André Platzer. *Verifiably Safe Off-Model Reinforcement Learning*, volume 11427 of *Lecture Notes in Computer Science*, pages 413–430. Springer, 2019.

33 Javier Garcıa and Fernando Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

34 Javier García and Fernando Fernández. Probabilistic policy reuse for safe reinforcement learning. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 13(3):14, 2019.

35 OpenAI Gym. Taxi-v2, 2018. URL: https://gym.openai.com/envs/Taxi-v2/.

36 Ernst Moritz Hahn, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi, and Dominik Wojtczak. Omega-regular objectives in model-free reinforcement learning. In *TACAS (1)*, volume 11427 of *LNCS*, pages 395–412. Springer, 2019.

37 Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-correct reinforcement learning. *CoRR*, abs/1801.08099, 2018. arXiv:1801.08099.

38 Mohammadhosein Hasanbeig, Yiannis Kantaros, Alessandro Abate, Daniel Kroening, George J. Pappas, and Insup Lee. Reinforcement learning for temporal logic control synthesis with probabilistic satisfaction guarantees. In *CDC*, pages 5338–5343. IEEE, 2019.

39 Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *CoRR, abs/1507.06527*, 2015. arXiv:1507.06527.

40 Manfred Jaeger, Peter Gjøl Jensen, Kim Guldstrand Larsen, Axel Legay, Sean Sedwards, and Jakob Haahr Taankvist. Teaching stratego to play ball: Optimal synthesis for continuous space mdps. In *ATVA*, volume 11781 of *Lecture Notes in Computer Science*, pages 81–97. Springer, 2019.

41 Sebastian Junges, Nils Jansen, and Sanjit A. Seshia. Enforcing almost-sure reachability in pomdps. *CoRR*, abs/2007.00085, 2020. arXiv:2007.00085.

42 Sebastian Junges, Nils Jansen, Christian Dehnert, Ufuk Topcu, and Joost-Pieter Katoen. Safety-constrained reinforcement learning for MDPs. In *TACAS*, volume 9636 of *LNCS*, pages 130–146. Springer, 2016.

43 Joost-Pieter Katoen. The probabilistic model checking landscape. In *LICS*, pages 31–45. ACM, 2016.

44 Bettina Könighofer, Mohammed Alshiekh, Roderick Bloem, Laura R. Humphrey, Robert Könighofer, Ufuk Topcu, and Chao Wang. Shield synthesis. *Formal Methods Syst. Des.*, 51(2):332–361, 2017.

45 Jan Kretínský, Guillermo A. Pérez, and Jean-François Raskin. Learning-based mean-payoff optimization in an unknown MDP under omega-regular constraints. In *CONCUR*, volume 118 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

46 Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

47 Marta Z. Kwiatkowska. Model checking for probability and time: from theory to practice. In *LICS*, page 351. IEEE CS, 2003.

48 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

49 Gil Lederman, Markus N. Rabe, Sanjit Seshia, and Edward A. Lee. Learning heuristics for quantified boolean formulas through reinforcement learning. In *ICLR*. OpenReview.net, 2020.

50 Sergey Levine and Vladlen Koltun. Guided policy search. In *ICML (3)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1–9. JMLR.org, 2013.

51 George Mason, Radu Calinescu, Daniel Kudenko, and Alec Banks. Assured reinforcement learning with formally verified abstract policies. In *ICAART (2)*, pages 105–117. SciTePress, 2017.

52 Teodor Mihai Moldovan and Pieter Abbeel. Safe exploration in markov decision processes. In *ICML*. icml.cc / Omnipress, 2012.

53 M. Ohnishi, L. Wang, G. Notomista, and M. Egerstedt. Barrier-certified adaptive reinforcement learning with applications to brushbot navigation. *IEEE Transactions on Robotics*, pages 1–20, 2019.

54 Martin Pecka and Tomas Svoboda. Safe exploration techniques for reinforcement learning–an overview. In *MESAS*, pages 357–375. Springer, 2014.

55 Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science*, pages 46–57. IEEE, 1977.

56 Diederik M. Roijers, Peter Vamplew, Shimon Whiteson, and Richard Dazeley. A survey of multi-objective sequential decision-making. *J. Artif. Intell. Res.*, 48:67–113, 2013.

57 Dorsa Sadigh, Eric S Kim, Samuel Coogan, S Shankar Sastry, and Sanjit A Seshia. A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. In *CDC*, pages 1091–1096. IEEE, 2014.

58 Dorsa Sadigh, Nick Landolfi, Shankar S Sastry, Sanjit A Seshia, and Anca D Dragan. Planning for cars that coordinate with people: leveraging effects on human actions for planning and active information gathering over human internal state. *Autonomous Robots*, 42(7):1405–1426, 2018.

59 Dorsa Sadigh, Shankar Sastry, Sanjit A Seshia, and Anca D Dragan. Planning for autonomous cars that leverage effects on human actions. In *Robotics: Science and Systems*, 2016.

60 David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nat.*, 529(7587):484–489, 2016.

61 Ion Stoica, Dawn Song, Raluca Ada Popa, David Patterson, Michael W Mahoney, Randy Katz, Anthony D Joseph, Michael Jordan, Joseph M Hellerstein, Joseph E Gonzalez, et al. A berkeley view of systems challenges for AI. *CoRR*, abs/1712.05855, 2017. `arXiv:1712.05855`.

62 Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

63 Florent Teichteil-Königsbuch. Stochastic safest and shortest path problems. In *AAAI*. AAAI Press, 2012.

64 Matteo Turchetta, Felix Berkenkamp, and Andreas Krause. Safe exploration for interactive machine learning. In *NeurIPS*, pages 2887–2897, 2019.

65 Abhinav Verma, Hoang Minh Le, Yisong Yue, and Swarat Chaudhuri. Imitation-projected programmatic reinforcement learning. In *NeurIPS*, pages 15726–15737, 2019.

66 Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 5052–5061. PMLR, 2018.

67 Angelina Wang, Thanard Kurutach, Kara Liu, Pieter Abbeel, and Aviv Tamar. Learning robotic manipulation through visual planning and acting. arXiv preprint, 2019. `arXiv:1905.04411`.

68 Min Wen, Rüdiger Ehlers, and Ufuk Topcu. Correct-by-synthesis reinforcement learning with temporal logic constraints. In *IROS*, 2015.

69 Douglas J White. Real applications of Markov decision processes. *Interfaces*, 15(6):73–83, 1985.

70 Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

71 Meng Wu, Jingbo Wang, Jyotirmoy Deshmukh, and Chao Wang. Shield synthesis for real: Enforcing safety in cyber-physical systems. In *FMCAD*, pages 129–137. IEEE, 2019.

72 Weichao Zhou and Wenchao Li. Safety-aware apprenticeship learning. In *CAV (1)*, volume 10981 of *Lecture Notes in Computer Science*, pages 662–680. Springer, 2018.

73 He Zhu, Zikang Xiong, Stephen Magill, and Suresh Jagannathan. An inductive synthesis framework for verifiable reinforcement learning. In *PLDI*, pages 686–701. ACM, 2019.

Modern Applications of Game-Theoretic Principles

Catuscia Palamidessi
Inria, Palaiseau, France
LIX, Ecole Polytechnique, Institut Polytechnique de Paris, Palaiseau, France
https://www.lix.polytechnique.fr/~catuscia/

Marco Romanelli
Inria, Palaiseau, France
LIX, Ecole Polytechnique, Institut Polytechnique de Paris, Palaiseau. France
University of Siena, Italy
http://www.lix.polytechnique.fr/Labo/Marco.Romanelli/

Abstract

Game theory is the study of the strategic behavior of rational decision makers who are aware that their decisions affect one another. Its simple but universal principles have found applications in the most diverse disciplines, including economics, social sciences, evolutionary biology, as well as logic, system science and computer science. Despite its long-standing tradition and its many advances, game theory is still a young and developing science. In this paper, we describe some recent and exciting applications in the fields of machine learning and privacy.

2012 ACM Subject Classification Security and privacy → Formal security models; Security and privacy → Privacy-preserving protocols; Security and privacy → Information flow control

Keywords and phrases Game theory, machine learning, privacy, security

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.4

Category Invited Paper

Funding This work was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme. Grant agreement № 835294.

1 Introduction

Game theory is concerned with situations in which one agent's best action depends on the expectation about what one or more other agents will do, and viceversa. Long before being investigated as a mathematical discipline, game-theoretic insights go back to ancient times, and influenced the strategies of political and military leaders. For example, when landing in Mexico with a small force, and surrounded by a far more numerous Atzecs army, Cortez burnt the ships on which he and his soldiers had landed, and took care of doing so very visibly, so that the Atzecs would see it. His action had a discouraging effect on the Aztecs, who must have thought: Any commander who is so confident as to willfully destroy his own option to save himself must have good reasons to be so optimist; it wouldn't be wise to attack an opponent who is so sure (whatever, exactly, the reason might be) that he can't lose. Thus the Atzecs retreated and missed the occasion – perhaps the best they ever had, to stop the Spanish invasion.

The initial formulation of games in mathematical terms is usually attributed to Emile Borel. However, it was John von Neumann who first solved the problem of the two-player zero-sum games, with the minimax theorem [14]. Formally, this theorem states that, if $f : X \times Y \to \mathbb{R}$ is a continuous function that is concave in the first argument and convex in the second one, then we have that:

$$\max_{x \in X} \min_{y \in Y} f(x, y) = \min_{y \in Y} \max_{x \in X} f(x, y).$$

The importance of this theorem consists in the fact that it states the existence of a *saddle point*, which in a sense represents the best possible pair of choices of the two adversaries – having to take each other strategy into account.

Von Neumann's work culminated in the fundamental book on game theory titled *Theory of Games and Economic Behavior*, written in collaboration with Oskar Morgenstern [20]. The mathematical foundations established in this book, and later on in the work of Nash, who generalized the notion of saddle point with that of Nash equilibrium [10], found a wide range of applications, especially in economics, social sciences, and biology.

In the past several decades, the framework has been deepened and generalized, and new variants and refinements are still being made. In logic, the principles of game theory have inspired the *game semantics*, with the purpose of giving an interpretation to intuitionistic logic [13] and to linear logic [5]. These ideas were further developed by Samson Abramsky, Radhakrishnan Jagadeesan, Pasquale Malacaria and independently Martin Hyland and Luke Ong, who focused in particular on defining strategies compositionally and inductively on the syntax. In this way, they succeeded to define a fully abstract semantics for PCF, which had been a long-standing problem [3, 9]. Following this line, game semantics has been applied to a variety of programming languages, and has led to semantic-based methods of software verification by model checking [2],

In concurrency theory, games were used for the first time by Colin Stirling to give an elegant characterization of *strong bisimulation* [19]. These are infinite-duration games played by two adversaries, called *Spoiler* and *Duplicator*. The game is played on pairs of states (s, t) in a labelled transition system, where Duplicator tries to prove that states s and t are strongly bisimilar, whereas Spoiler tries to disprove this. Spoiler plays first, and challenges Duplicator with a move from either s or t, while Duplicator is required to match this move. At the next turn, the game continues from the pair of the new states reached after the transitions. If the game is infinite (i.e., the play continues forever), Duplicator wins the game. If on the other hand the game is finite, namely one of the players gets stuck, then the game is won by the other player. Note that if it is Duplicator that gets stuck, it means that Spoiler found a way to distinguish the states. Viceversa, if it is Spoiler that gets stuck, or if the games goes on indefinitely, then it means that Duplicator is able to match all Spoiler's challenges. Hence, the two states are deemed equivalent if and only if Duplicator wins.

In recent times, game theory has found additional exciting applications in new fields: machine learning (ML), privacy, and computer security. We are going to give an overview of the first two applications in the next sections.

2 Game theory and machine learning

The principles of game theory have inspired Ian Goodfellow and other researchers at the University of Montreal (including Yoshua Bengio) to propose the so-called *generative adversarial networks* (GANs) [8], which has been called by Facebook's AI research director Yann LeCun "the most interesting idea in the last 10 years in machine learning".

To fully understand the power of this idea, we should explain the difference between two main categories of learning algorithms: the *discriminative* and the *generative* ones. The first category aims at building classifiers. For example, a machine that takes in input a picture of an animal and outputs a label representing the kind of animal: a cat, a dog, etc. To

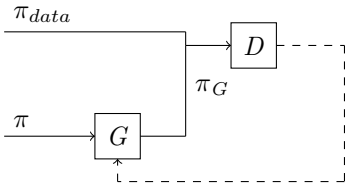


Figure 1 Scheme of a GANs, where π_{data} is the distribution of the real data, and π is the random number generator.

construct such model, typically the learning phase consists in feeding the machine with a huge amount of pairs of the form (animal, label), called training examples. Processing these examples brings to the tuning of the nodes of the net in such a way that, later on, when the machine is given a new picture (possibly never seen in training phase), it will be able to associate to it the most likely label.

In contrast, a generative algorithm should be able to *generate new examples*. For instance, the generative counterpart of the model above, should be able to generate a picture of a new dog (never seen in training phase) each time it is requested.

In 2014, while the research on discriminative algorithms had made enormous progress in the last decade and the corresponding models were being deployed successfully, generative algorithms were still an open challenge. In his seminal paper, Goodfellow and his colleagues had the idea of using an architecture consisting of two neural networks, pitting one against the other (thus the "adversarial"). The two nets are called respectively *generator* G and *discriminator* D. The purpose of G is to learn to transform a latent space of random numbers into data of the target domain, trying to imitate the (unknown) distribution of the data (the random generator in input is necessary because neural nets are essentially deterministic). The discriminator, on the other hand, tries to distinguish between real data sampled from the distribution and the synthetic data generated by the generator. The output of the discriminator is fed back to the generator, that in this way can learn if it has been successful or not in fooling the adversary. The goal of the generative network, indeed, is to trick the discriminator into thinking that the novel data produced is coming from true data distribution; this way, it increases the discriminator's error rate. The goal of the discriminator, of course, is to minimize its own error rate. Thus this game is in fact a two-player zero-sum game. The GANs architecture is represented in Figure 1. The interplay between the two networks is regulated by the following minimax problem:

$$\min_D \max_G V(D,G) \quad \text{where} \quad V(D,G) = \mathbb{E}_{x\sim\pi_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim\pi(z)}[\log(1 - D(G(z)))]$$

GANs are used widely in image generation, video generation and voice generation, and their results are impressive. As an example, Figure 2 shows the outcome of a GANs model developed in [11]. GANs are also used to crete artistic works according to a given style, see for instance Figure 3.

3 Game theory and privacy

The huge amount of data collected by digital service providers and the development of powerful technologies to perform analytics, in particular ML, have exacerbated the risks for privacy. See for example the model inversion attacks [7] and the membership inference attacks [17].

Figure 2 Synthetic images representing human faces (1024 × 1024 pixels) generated using the CELEBA-HQ dataset [11].

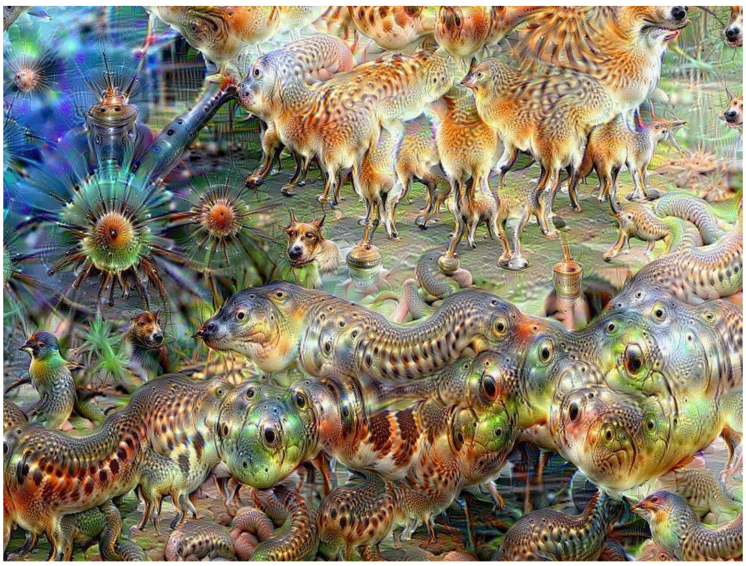


Figure 3 An artwork created using DeepDream, which researchers at Google developed in 2015.

However, if ML can be a threat to privacy, it can also be exploited as a powerful means to build "good" privacy-protection mechanisms, i.e. mechanisms that are robust and which preserve a good utility. In particular, we consider mechanisms that achieve privacy by adding controlled noise to the original data. Following the approach of [18], we aim at maximizing the privacy protection while ensuring the desired quality of service (QoS) that the user receives when providing obfuscated data. In this paper we focus on *location privacy* and on the risk of the *re-identification* of a user from his location, but the framework we develop is general and can be applied to any scenario in which there is sensitive information that we wish to hide, correlated to information that we need to make public.

In the context of location privacy, the QoS constraint (aka *utility* constraint) is typically expressed as a bound on the expected distance between the real location and the obfuscated one (cfr. [18, 4]). This is a linear constraint. If also privacy is measured by a linear expression, then in principle it is possible to use linear programming to compute a mechanism that provides the optimal privacy-utility trade-off (cfr. [18, 6, 15]). The limitation of this approach, however, is that it does not scale to large datasets. The problem is that the linear program

needs one variable for every pair (w, z) of real and obfuscated locations. Such variables represent the probability of producing the obfuscated location z when the real one is w. For a 50×50 grid this is more than six million variables, which is already at the limit of what modern solvers can do. For a 260×260 grid, the program has 4.5 billion variables, making it completely intractable (we could not even launch such a program due to huge memory requirements).

In order to overcome this limitation, in [16] we have explored an approach based on ML. Inspired by the GANs paradigm [8], we constructed a system of two adversarial neural networks: a *generator* G and *classifier* C. G generates noise so to confuse the adversary as much as possible, within the boundaries of the utility constraints, while C takes the noisy locations produced by G as inputs and tries to re-identify (classify) the corresponding user. Note that there are considerable differences with the standard GANs paradigm: in the latter the generator tries to reproduce a known target data distribution, while in our setting the target distribution that optimizes the data obfuscation is unknown and G has to "invent" it.

The interplay between G and C can be seen as an instance of a zero-sum Stackelberg game [18], where G is the *leader*, and C is the *follower*, and the payoff function f is the privacy loss. Finding the optimal point of equilibrium between G and C corresponds to solving a minimax problem on f with G being the minimizer and C the maximizer.

The definition of f was particularly critical in our setting. A first idea would be to measure the privacy in terms of C's capability of re-identifying users given their locations (i.e. classification accuracy). However this would be a poor choice. In fact, consider the following scenario: two users, A and B, are respectively in two locations a and b. C is trained on this dataset and learns that a corresponds to A and b to B. At the next iteration, G will maximize C's misclassification error by swapping the locations so that a corresponds now to B and b to A. But at the next iteration C is trained again and learns the new correct classification. G would then keep swapping the locations without ever reaching the equilibrium point. A similar example was independently pointed out in [1].

In order to address this issue we adopted a different payoff function f, one which is less sensitive to the particular labeling strategy of C and takes into account not just the *precision* of the classification, but, rather, the *information* contained in it. A way to formalize this intuition was to use the *mutual information* $I(X;Y)$, where X, Y are respectively the random variables associated to the *true ids*, and to the ids resulting from the classification (*predicted ids*). We recall that $I(X;Y) = H(X) - H(X|Y)$, where $H(X)$ is the entropy of X and $H(X|Y)$ is the residual entropy of X given Y.

The nets G and C tries to minimize/maximize $I(X;Y)$ according to the following minimax game formulation:

$$\min_G \max_C I(X;Y).$$

where the minimization by G is on the mechanisms producing the noisy points Z and which satisfy the utility constraint, while the maximization by C is on the classifications that map Z into Y.

The adversarial networks are organized as follows: G takes in input a pair $(x, w) \sim (X, W)$ consisting of an identifier x and an associated location w, and two random seeds. (The latter are necessary because a neural network by itself is deterministic.) The goal of the classifier it to transform the random inputs in "clever" probabilistic noise to add to the two coordinates of the location, taking into account the utility consraint. The noisy location produced by G is given in input to C, which tries to re-identify it. Note that C is helped in this by the fact that X and W are correlated, and that Z is also correlated to W and therefore to X, because of the utility constraint. The result of C is then fed back to G, which uses it to refine the mechanism.

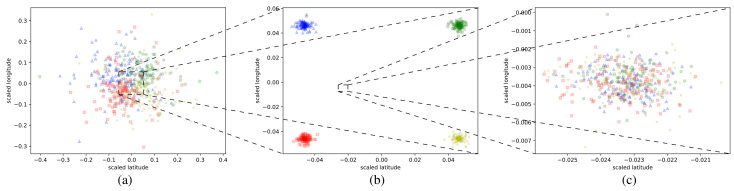


Figure 4 Synthetic testing data. From left to right: Laplace noise, no noise, our noise. $L = 270$m.

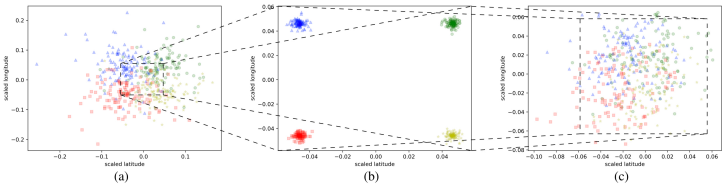


Figure 5 Synthetic testing data. From left to right: Laplace noise, no noise, our noise. $L = 173$m.

In the next section we evaluate the performance of our mechanism with respect to others used in location privacy. We fix the level of utility and we compare the resulting Bayes error of X given Z, which is defined as $B(X \mid Z) = \sum_z P_Z(z)(1 - \max_x P_{X|Z}(x \mid z))$. The use of this metric is justified by the fact that it can be proved that $B(X \mid Z)$ is a lower bound for $B(X \mid Y)$ for any possible classifier that derives Y from Z. It is indeed the error of the "ideal" Bayesian classifier, which is optimal, and represents, therefore, the strongest possible adversary for the given Z.

The other two mechanisms under comparison are the (planar) Laplace [4] and the optimal mechanism, namely the mechanism that induces the highest $B(X \mid Z)$ for the given utility loss.

4 Experiments

We perform four different experiments, over synthetic and Gowalla [12] data, and with two different utility bounds.

The first experiment is on synthetic data with a loose utility constraint. We set the domain of W and Z to be a square of side $6500m$. We create a dataset of 600 synthetic check-ins (locations) for each one of 4 users (classes). We use 480 check-ins for training and validation, and 120 for testing. We place the check-ins randomly in clouds of $50m$ max around the four vertices of a square of side 300m (each user corresponds to a vertex). We set the bound on utility, namely the maximum expected distance L between Z and W (aka *distortion*), to be $270m$. Note that such L is enough for each check-in to be remapped into the center of the square, that is what the optimal mechanism would do. In fact, if all noisy locations coincide then they give no information about the identity of the corresponding user.

We first apply a Laplace mechanism to the test data (cfr. Figure 4(a)). For the sake of the comparison, we have set the parameters of the Laplace to be such that the average distortion is at least $270m$. In fact, it results to be $\approx 298.40m$. Then, we train our mechanism and we apply it to the test data (cfr. Figure 4(c)), obtaining an average distortion of $\approx 219.60m$. Note that, while the Laplace tends to "spread out" the obfuscated check-ins, our method tends to concentrate them into a small area in the center, thus approximating the optimal mechanism. Remapping all check-ins into the same location may not be possible in general,

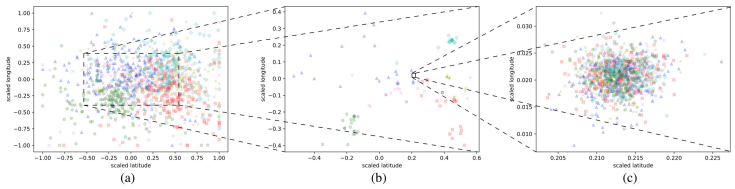


Figure 6 Gowalla testing data. From left to right: Laplace noise, no noise, our noise. $L = 1150$m.

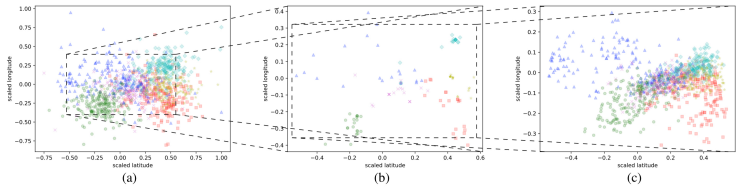


Figure 7 Gowalla testing data. From left to right: Laplace noise, no noise, our noise. $L = 518$m.

as it depends on the utility constraint. Nevertheless, we can expect that our mechanism will tend to overlap the check-ins of different classes as much as allowed by the utility constraint. With the Laplace, on the contrary, the zones of the various classes tend to remain separated and the noise is injected in a symmetrical fashion and remaps the check-ins into zones that remain separate and hence do not contribute to increase privacy.

We now evaluate $B(X \mid Z)$. To this aim we need to discretize Z, and we do so by using a 260×260 cells grid where each cell has $25m$ long side, i.e. $6500/260$. We apply the mechanisms on each check-in in the test set for 500 times with different random seeds thus obtaining $4 \times 120 \times 500$ obfuscated check-ins. In a real situation this would correspond to a scenario in which a user is in a certain location multiple times and each time reports different noisy locations. Figure 8(a) represents the Bayes errors for the three mechanisms. Note that, since the optimal mechanism in this case leaks no information about X (identifiers), its Bayes error is the same as that of randomly guessing the identifier. Since there are 4 identifiers, and as classes they are balanced, $B(X \mid Z) = 1 - 1/4 = 0.75$.

The second experiment is the same as the first, except now the utility bound is $173m$. This bound is not enough to map all check-ins to the center of the square, but it is enough to map the check-ins of two adjacent vertices into the same point in the middle of the corresponding side. Hence one of the possible optimal mechanisms is the one that induces a partition of the identifiers in two sets, where each set contains two indistinguishable identifiers. The Bayes error of an optimal mechanism is therefore $B(X \mid Z) = 1 - 1/2 = 0.50$. The Laplace method and ours produce the distribution in Figures 5(a) and 5(c). The Bayes error is reported in Figure 8(b).

The third experiment is on data extracted from the *Gowalla* dataset [12]. We set the domain of W and Z to be a squared region of with side 4500m and centered in 5, Boulevard de Sébastopol, Paris. We select the 6 users who checked in the region most frequently. (We limit the classes to 6 for visualization sake, but our method can handle an unbound number of classes.) For each location we create 10 repetitions with different random seeds so to increase the dataset. Again for the sake of visualization, we filter the check-ins so obtained to reduce the overlapping of those belonging to different classes. We divide the resulting check-ins into 4920 for training and validation, and 1200 for testing. Fig. 6(b) shows the test data. We set the utility threshold to $L = 1150m$, which is enough to map all check-ins into a

Synthetic data, low utility		
Laplace	Ours	Optimal
0.39	0.74	0.75

(a)

Synthetic data, high utility		
Laplace	Ours	Optimal
0.23	0.42	0.50

(b)

Figure 8 Bayes error on synthetic data, for the Laplace mechanism, our mechanism, and the optimal mechanism, on a grid of 260×260 cells.

Gowalla data, low utility		
Laplace	Ours	Optimal
0.33	0.80	0.83

(a)

Gowalla data, high utility		
Laplace	Ours	Optimal
0.28	0.38	?

(b)

Figure 9 Bayes error on Gowalla data, for the Laplace mechanism, our mechanism, and the optimal mechanism, on a grid of 260×260 cells. In the last table the Bayes error of the optimal mechanism is unknown: the linear program contains 4.5 billion variables, making it intractable in practice.

unique location. Therefore the Bayes error of the optimal mechanisms is the same as that of the random guess, i.e., $B(X \mid Z) = 1 - 1/6 \approx 0.83$. The distribution for Laplace and ours are in Figure 6(a) and 6(c) respectively, and the corresponding Bayes errors are reported in Figure 9(a).

Finally, the setting of the fourth experiment is like the third one, except we now set a stricter threshold, $L = 518m$. We obtain the data distribution in Figure 7(a) for Laplace and Figure 7(c) for our mechanism. The corresponding Bayes errors are reported in Figure 9(b). In this case we do not know the optimal mechanism: we cannot figure it theoretically and we cannot compute it due to the large number of variables.

References

1 Martín Abadi and David G. Andersen. Learning to protect communications with adversarial neural cryptography. *CoRR*, abs/1610.06918, 2016. `arXiv:1610.06918`.

2 Samson Abramsky, Dan R. Ghica, Andrzej S. Murawski, and C. H. Luke Ong. Applying game semantics to compositional software modeling and verification. In Kurt Jensen and Andreas Podelski, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 421–435, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

3 Samson Abramsky, Radha Jagadeesan, and Pasquale Malacaria. Full abstraction for PCF. *Information and Computation*, 163(2):409–470, 2000. `doi:10.1006/inco.2000.2930`.

4 Miguel E. Andrés, Nicolás E. Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Geo-indistinguishability: differential privacy for location-based systems. In *Proceedings of the 20th ACM Conference on Computer and Communications Security (CCS 2013)*, pages 901–914. ACM, 2013. `doi:10.1145/2508859.2516735`.

5 Andreas Blass. A game semantics for linear logic. *Annals of Pure and Applied Logic*, 56(1–3):183–220, 29 April 1992.

6 Nicolás E. Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Optimal geo-indistinguishable mechanisms for location privacy. In *Proceedings of the 21th ACM Conference on Computer and Communications Security (CCS 2014)*, 2014.

7 Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 1322–1333, New York, NY, USA, 2015. ACM. `doi:10.1145/2810103.2813677`.

8 Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014. URL: `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

9 J. M. E. Hyland and C.-H. Luke Ong. On full abstraction for PCF: i, ii, and III. *Information and Computation*, 163(2):285–408, 2000. `doi:10.1006/inco.2000.2917`.

10 John Forbes Nash Jr. Non-cooperative games. *Annals of Mathematics*, 2(54):286–295, 1951.

11 Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *6th International Conference on Learning Representations, (ICLR)*. OpenReview.net, 2018.

12 Jure Leskovec and Andrej Krevl. The Gowalla dataset (Part of the SNAP collection). `https://snap.stanford.edu/data/loc-gowalla.html`.

13 Paul Lorenzen and Kuno Lorenz. *Dialogische Logik*. Kurztitelaufnahme der Deutschen Bibliothek. Wissenschaftliche Buchgesellschaft, [Abt. Verlag], 1978.

14 John Von Neumann. Zur theorie der gesellshaftsspiele. *Mathematical Annals*, 100:295–320, 1928.

15 Simon Oya, Carmela Troncoso, and Fernando Pérez-González. Back to the drawing board: Revisiting the design of optimal location privacy-preserving mechanisms. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1959–1972. ACM, 2017. `doi:10.1145/3133956.3134004`.

16 Marco Romanelli, Catuscia Palamidessi, and Konstantinos Chatzikokolakis. Generating optimal privacy-protection mechanisms via machine learning. In *Proceedings of the IEEE International Symposium on Computer Security Foundations (CSF)*, 2020. `arXiv:1904.01059`.

17 Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 3–18. IEEE Computer Society, 2017. `doi:10.1109/SP.2017.41`.

18 Reza Shokri, George Theodorakopoulos, and Carmela Troncoso. Privacy games along location traces: A game-theoretic framework for optimizing location privacy. *ACM Transactions on Privacy and Security*, 19(4):11:1–11:31, 2017. `doi:10.1145/3009908`.

19 Colin Stirling. Bisimulation, modal logic and model checking games. *Logic Journal of the IGPL*, 7(1):103–124, 1999. `doi:10.1093/jigpal/7.1.103`.

20 John von Neumann and Oskar Morgenstern. *Theory of games and economic behavior*. Princeton University Press, Princeton, 1944.

CONCUR Test-Of-Time Award 2020 Announcement

Luca Aceto
ICE-TCS, Department of Computer Science, Reykjavik University, Iceland
Gran Sasso Science Institute, L'Aquila, Italy

Jos Baeten
CWI Amsterdam, The Netherlands
University of Amsterdam, The Netherlands

Patricia Bouyer-Decitre
LSV, CNRS, Gif-sur-Yvette, France
ENS Paris-Saclay, Gif-sur-Yvette, France

Holger Hermanns
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

Alexandra Silva
University College London, UK

Abstract

This short article announces the recipients of the CONCUR Test-of-Time Award 2020.

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases Concurrency, CONCUR Test-of-Time Award

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.5

Category Invited Paper

Acknowledgements We thank Javier Esparza (chair of the CONCUR Steering Committee), Ilaria Castellani and Mohammad Reza Mousavi (chair and secretary of the IFIP Working Group 1.8 on Concurrency Theory), and Igor Konnov and Laura Kovacs (chairs of the CONCUR 2020 Program Committee) for their assistance throughout our work.

1 Introduction

The International Conference on Concurrency Theory (CONCUR) and the IFIP Working Group 1.8 on Concurrency Theory have established the CONCUR Test-of-Time Award to recognize important achievements in Concurrency Theory that were published at the CONCUR conference and have stood the test of time. All papers published at CONCUR between 1990 and 1995 were eligible for the first installment of the award, which was presented at the 31st International Conference on Concurrency Theory (CONCUR 2020). The conference was held on line from Vienna, Austria, in the period 1–4 September 2020, with Igor Konnov and Laura Kovacs as chairs of the program committee.

We had the great honour to serve as members of the first CONCUR Test-of-Time Award Jury, and were asked by the CONCUR Steering Committee to select one or two awardees for the periods 1990–1993 and 1992–1995.

After having made a shortlist of candidate award recipients for each of the above-mentioned periods and having thoroughly discussed their relative merits and impact on the CONCUR research community and beyond, the Jury unanimously selected the four articles mentioned below for the award out of a slate of many excellent candidates.

2 The Award Winning Contributions

2.1 Period 1990–1993

- Rob van Glabbeek. "The Linear Time-Branching Time Spectrum".
  Citation: The companion papers on "The Linear Time-Branching Time Spectrum", published by Rob van Glabbeek at CONCUR 1990 and 1993, jointly receive one award for offering a highly influential taxonomy of the menagerie of process semantics, both in a setting where every system action is observable and in the presence of silent moves. Each of the plethora of studied semantics comes equipped with a variety of elegant characterisations in terms of modal logics, testing scenarios, relations, and complete axiomatisations. The encyclopedic nature of the above-mentioned papers has made them a must read for researchers in concurrency theory for nearly 30 years.
- Søren Christensen, Hans Hüttel and Colin Stirling. "Bisimulation Equivalence is Decidable for all Context-Free Processes".
  Citation: The paper "Bisimulation Equivalence is Decidable for all Context-Free Processes", published by Søren Christensen, Hans Hüttel and Colin Stirling at CONCUR 1992, receives one award for extending and simplifying the seminal result by Baeten, Bergstra and Klop, who proved the decidability of bisimilarity over normed context-free processes. The CONCUR 1992 paper has paved the way to further decidability and complexity results for a variety of classes of infinite-state processes. This includes the 2-EXPTIME algorithm for bisimilarity over BPA presented by Burkart, Caucal and Steffen in a paper published at MFCS 1995, and the work by Senizergues in papers at FOCS 1998 and in the SIAM Journal on Computing in 2005, presenting decidability results for all "equational graphs" with finite out-degree.

2.2 Period 1992–1995

- Roberto Segala and Nancy Lynch. "Probabilistic Simulations for Probabilistic Processes".
  Citation: The paper "Probabilistic Simulations for Probabilistic Processes", published by Roberto Segala and Nancy Lynch at CONCUR 1994, receives one award for introducing the "simple" probabilistic automata model. Unlike earlier attempts to embrace probabilities, transition targets here are probability distributions over states, and this makes it possible to lift core process algebraic results in a very elegant manner. Probabilistic automata have quickly been recognised as the pivotal link between classical concurrency theory and the theory of discrete-state Markov processes. They have become the central subjects of probabilistic model checking, and are echoed in a range of very influential modelling formalisms including probabilistic timed automata, probabilistic hybrid automata, and Markov automata.
- Davide Sangiorgi. "A Theory of Bisimulation for the pi-Calculus".
  Citation: The paper "A Theory of Bisimulation for the pi-Calculus", published by Davide Sangiorgi at CONCUR 1993, receives one award for introducing the notion of open bisimilarity, which, unlike early and late bisimilarity, is a congruence for the pi-calculus. Open bisimilarity makes it possible to view most names as uninstantiated variables, and this allows for the development of efficient tools based on a kind of symbolic state-space exploration. Open bisimilarity and tools based on it have, for instance, played an important role in research on cryptographic protocols modelled using extensions of the pi-calculus. For a recent example, Horne has used open bisimilarity as the appropriate way to model the capabilities of an attacker trying to get confidential information, with a real-world application to finding and fixing a privacy flaw in e-passports presented by Filimonov et al. at ESORICS 2019.

3 Concluding Remarks

Interviews with the award recipients, which give some information on the historical context that led them to develop their award-winning work and on their research philosophy, may be found in four blog posts that are accessible from `https://processalgebra.blogspot.com/`, and are collected under one roof in a contribution to the June 2020 issue of the Bulletin of the EATCS [1].

We hope that researchers in Concurrency Theory will read or re-read the award-winning papers and the others that were presented at the early editions of the CONCUR conference, which are a veritable treasure trove of information about our field's intellectual heritage and of inspiration for future work.

References

1 Luca Aceto. Interviews with the 2020 CONCUR Test-of-Time Award recipients. *Bulletin of the EATCS*, 131:66–84, 2020. URL: `http://bulletin.eatcs.org/index.php/beatcs/article/view/626`.

Reactive Bisimulation Semantics for a Process Algebra with Time-Outs

Rob van Glabbeek
Data61, CSIRO, Sydney, Australia
UNSW, Sydney, Australia
rvg@cs.stanford.edu

Abstract

This paper introduces the counterpart of strong bisimilarity for labelled transition systems extended with time-out transitions. It supports this concept through a modal characterisation, congruence results for a standard process algebra with recursion, and a complete axiomatisation.

2012 ACM Subject Classification Theory of computation → Process calculi

Keywords and phrases Process algebra, time-outs, labelled transition systems, reactive bisimulation semantics, Hennessy-Milner logic, modal characterisations, recursion, complete axiomatisations

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.6

Related Version Like many conference papers, this is an extended abstract, with omitted proofs. The full version of this paper is available at `http://rvg.web.cse.unsw.edu.au/pub/reactive.pdf`.

Acknowledgements My thanks to the referees for helpful feedback.

1 Introduction

This is a contribution to classic untimed non-probabilistic process algebra, modelling systems that move from state to state by performing discrete, uninterpreted actions. A system is modelled as a process-algebraic expression, whose standard semantics is a state in a labelled transition system (LTS). An LTS consists of a set of states, with action-labelled transitions between them. The execution of an action is assumed to be instantaneous, so when any time elapses the system must be in one of its states. With "untimed" I mean that I will refrain from quantifying the passage of time; however, whether a system can pause in some state or not will be part of my model.

Following [29], I consider *reactive* systems that interact with their environments through the synchronous execution of visible actions a, b, c, ... taken from an alphabet A. At any time, the environment *allows* a set of actions $X \subseteq A$, while *blocking* all other actions. At discrete moments the environment can change the set of actions it allows. In a metaphor from [29], the environment of a system can be seen as a user interacting with it. This user has a button for each action $a \in A$, on which it can exercise pressure. When the user exercises pressure *and* the system is in a state where it can perform action a, the action occurs. For the system this involves taking an a-labelled transition to a following state; for the environment it entails the button going down, thus making the action occurrence observable. This can trigger the user to alter the set of buttons on which it exercises pressure.

The current paper considers two special actions that can occur as transition labels: the traditional *hidden action* τ [29], modelling the occurrence of an instantaneous action from which we abstract, and the *time-out* action t, modelling the end of a time-consuming activity from which we abstract. The latter was introduced in [16] and constitutes the main novelty of the present paper with respect to [29] and forty years of process algebra. Both special actions are unobservable, in the sense that their occurrence cannot trigger any state-change in the environment. Conversely, the environment cannot cause or block their occurrence.

Following [16], I model the passage of time in the following way. When a system arrives in a state P, and at that time X is the set of actions allowed by the environment, there are two possibilities. If P has an outgoing transition $P \xrightarrow{\alpha} Q$ with $\alpha \in X \cup \{\tau\}$, the system immediately takes one of the outgoing transitions $P \xrightarrow{\alpha} Q$ with $\alpha \in X \cup \{\tau\}$, without spending any time in state P. The choice between these actions is entirely nondeterministic. The system cannot immediately take a transition $\xrightarrow{b}$ with $b \in A \backslash X$, because the action b is blocked by the environment. Neither can it immediately take a transition $P \xrightarrow{\mathrm{t}} Q$, because such transitions model the end of an activity with a finite but positive duration that started when reaching state P.

In case P has no outgoing transition $P \xrightarrow{\alpha} Q$ with $\alpha \in X \cup \{\tau\}$, the system idles in state P for a positive amount of time. This idling can end in two possible ways. Either one of the time-out transitions $P \xrightarrow{\mathrm{t}} Q$ occurs, or the environment spontaneously changes the set of actions it allows into a different set Y with the property that $P \xrightarrow{a} Q$ for some $a \in Y$. In the latter case a transition $P \xrightarrow{a} Q$ occurs, with $a \in Y$. The choice between the various ways to end a period of idling is entirely nondeterministic. It is possible to stay forever in state P only if there are no outgoing time-out transitions $P \xrightarrow{\mathrm{t}} Q$.

The addition of time-outs enhances the expressive power of LTSs and process algebras. The process $a.P + \mathrm{t}.b.Q$, for instance, models a choice between $a.P$ and $b.Q$ where the former has priority. In an environment where a is allowed it will always choose $a.P$ and never $b.Q$; but in an environment that blocks a the process will, after some delay, proceed with $b.Q$. Such a priority mechanism cannot be modelled in standard process algebras without time-outs, such as CCS [29], CSP [4, 24] and ACP [1, 8]. Additionally, mutual exclusion cannot be correctly modelled in any of these standard process algebras [17], but adding time-outs makes it possible – see Section 11 for a more precise statement.

In [16] I characterised the coarsest reasonable semantic equivalence on LTSs with time-outs – the one induced by *may testing*, as proposed by De Nicola & Hennessy [6]. In the absence of time-outs, may testing yields *weak trace equivalence*, where two processes are defined equivalent iff they have the same *weak traces*: sequence of actions the system can perform, while eliding hidden actions. In the presence of time-outs weak trace equivalence fails to be a congruence for common process algebraic operators, and may testing yields its congruence closure, characterised in [16] as *(rooted) failure trace equivalence*.

The present paper aims to characterise one of the finest reasonable semantic equivalences on LTSs with time-outs – the counterpart of strong bisimilarity for LTSs without time-outs. Naturally, strong bisimilarity can be applied verbatim to LTSs with time-outs – and has been in [16] – by treating t exactly like any visible action. Here, however, I aim to take into account the essence of time-outs, and propose an equivalence that satisfies some natural laws discussed in [16], such as $\tau.P + \mathrm{t}.Q = \tau.P$ and $a.P + \mathrm{t}.(Q + \tau.R + a.S) = a.P + \mathrm{t}.(Q + \tau.R)$. To motivate the last law, note that the time-out transition $a.P + \mathrm{t}.(Q + \tau.R + a.S) \xrightarrow{\mathrm{t}} Q + \tau.R + a.S$ can occur only in an environment that blocks the action a, for otherwise a would have taken place before the time-out went off. The occurrence of this transition is not observable by the environment, so right afterwards the state of the environment is unchanged, and the action a is still blocked. Therefore, the process $Q + \tau.R + a.S$ will, without further ado, proceed with the τ-transition to R, or any action from Q, just as if the $a.S$ summand were not present.

Standard process algebras and LTSs without time-outs can model systems whose behaviour is triggered by input signals from the environment in which they operate. This is why they are called "reactive systems". By means of time-outs one can additionally model systems whose behaviour is triggered by the *absence* of input signals from the environment, during a sufficiently long period. This creates a greater symmetry between a system and its environment, as it has always been understood that the environment or user of a system can

change its behaviour as a result of sustained inactivity of the system it is interacting with. Hence one could say that process algebras and LTSs enriched with time-outs form a more faithful model of reactivity. It is for this reason that I use the name *reactive bisimilarity* for the appropriate form of bisimilarity on systems modelled in this fashion.

Section 2 introduces strong reactive bisimilarity as the proper counterpart of strong bisimilarity in the presence of time-out transitions. Naturally, it coincides with strong bisimilarity when there are no time-out transitions. Section 3 derives a modal characterisation; a reactive variant of the Hennessy-Milner logic. Section 4 offers an alternative characterisation of strong reactive bisimilarity that will be more convenient in proofs, although it is lacks the intuitive appeal to be used as the initial definition.

Section 5 recalls the process algebra CCSP, a common mix of CCS and CSP, and adds the time-out action, as well as two auxiliary operators that will be used in the forthcoming axiomatisation. Section 6 states that in this process algebra one can express all countably branching transition systems, and only those, or all and only the finitely branching ones when restricting to guarded recursion.

Section 7 recalls the concept of a congruence, focusing on the congruence property for the recursion operator, which is commonly the hardest to establish. It then shows that the simple *initials equivalence*, as well as Milner's strong bisimilarity, are congruences. Due to the presence of negative premises in the operational rules for the auxiliary operators, these proofs are not entirely trivial. Using these results as a stepping stone, Section 8 shows that strong reactive bisimilarity is a congruence for my extension of CCSP. Here the congruence property for one of the auxiliary operators with negative premises is needed in establishing the result for the common CCSP operators, such as parallel composition.

Section 9 shows that guarded recursive specifications have unique solutions up to strong reactive bisimilarity. Using this, Section 10 provides a sound and complete axiomatisation for processes with guarded recursion. My completeness proof combines three innovations in establishing completeness of process algebraic axiomatisations. First of all, following [19], it applies to *all* processes in a Turing powerful language like guarded CCSP, rather than the more common fragment merely employing finite sets of recursion equations featuring only choice and action prefixing. Secondly, instead of the classic technique of *merging guarded recursive equations* [27, 28, 35, 9, 26], which in essence proves two bisimilar systems P and Q equivalent by equating both to an intermediate variant that is essentially a *product* of P and Q, I employ the novel method of *canonical solutions* [25], which equates both P and Q to a canonical representative within the bisimulation equivalence class of P and Q – one that has only one reachable state for each bisimulation equivalence class of states of P and Q. In fact I tried so hard, and in vain, to apply the traditional technique of merging guarded recursive equations, that I came to believe that it fundamentally does not work for this axiomatisation. The third innovation is the use of the axiom of choice [36] in defining the transition relation on my canonical representative, in order to keep this process finitely branching.

Section 11 describes a worthwhile gain in expressiveness caused by the addition of time-outs, and presents an agenda for future work.

Due to lack of space, all proofs have been deleted. However, some appear in the appendix – their theorems are marked @ – and all can be found in the accompanying technical report.

2 Reactive bisimilarity

A *labelled transition system* (LTS) is a triple $(\mathbb{P}, Act, \rightarrow)$ with $\mathbb{P}$ a set (of *states* or *processes*), Act a set (of *actions*) and $\rightarrow \in \mathbb{P} \times Act \times \mathbb{P}$. In this paper I consider LTSs with $Act := A \uplus \{\tau, \mathrm{t}\}$, where A is a set of *visible actions*, τ is the *hidden action*, and t the *time-out* action. The set of *initial* actions of a process $P \in \mathbb{P}$ is $\mathcal{I}(P) := \{\alpha \in A \cup \{\tau\} \mid P \xrightarrow{\alpha}\}$. Here $P \xrightarrow{\alpha}$ means that there is a Q with $P \xrightarrow{\alpha} Q$.

▶ **Definition 1.** *A* strong reactive bisimulation *is a symmetric relation* $\mathscr{R} \subseteq (\mathbb{P} \times \mathscr{P}(A) \times \mathbb{P}) \cup (\mathbb{P} \times \mathbb{P})$ *(meaning that* $(P, X, Q) \in \mathscr{R} \Leftrightarrow (Q, X, P) \in \mathscr{R}$ *and* $(P, Q) \in \mathscr{R} \Leftrightarrow (Q, P) \in \mathscr{R}$*), such that,*

- *if* $(P, Q) \in \mathscr{R}$ *and* $P \xrightarrow{\tau} P'$*, then there exists a* Q' *such that* $Q \xrightarrow{\tau} Q'$ *and* $(P', Q') \in \mathscr{R}$*,*
- *if* $(P, Q) \in \mathscr{R}$ *then* $(P, X, Q) \in \mathscr{R}$ *for all* $X \subseteq A$*,*

and for all $(P, X, Q) \in \mathscr{R}$*,*

- *if* $P \xrightarrow{a} P'$ *with* $a \in X$*, then there exists a* Q' *such that* $Q \xrightarrow{a} Q'$ *and* $(P', Q') \in \mathscr{R}$*,*
- *if* $P \xrightarrow{\tau} P'$*, then there exists a* Q' *such that* $Q \xrightarrow{\tau} Q'$ *and* $(P', X, Q') \in \mathscr{R}$*,*
- *if* $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$*, then* $(P, Q) \in \mathscr{R}$*, and*
- *if* $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ *and* $P \xrightarrow{\mathrm{t}} P'$*, then* $\exists Q'$ *such that* $Q \xrightarrow{\mathrm{t}} Q'$ *and* $(P', X, Q') \in \mathscr{R}$*.*

Processes $P, Q \in \mathbb{P}$ *are* strongly X-bisimilar*, denoted* $P \leftrightarrows_r^X Q$*, if* $(P, X, Q) \in \mathscr{R}$ *for some strong reactive bisimulation* $\mathscr{R}$*. They are* strongly reactive bisimilar*, denoted* $P \leftrightarrows_r Q$*, if* $(P, Q) \in \mathscr{R}$ *for some strong reactive bisimulation* $\mathscr{R}$*.*

Intuitively, $(P, X, Q) \in \mathscr{R}$ says that processes P and Q behave the same way, as witnessed by the relation $\mathscr{R}$, when placed in the environment X – meaning any environment that allows exactly the actions in X to occur – whereas $(P, Q) \in \mathscr{R}$ says they behave the same way in an environment that has just been triggered to change. An environment can be thought of as an unknown process placed in parallel with P and Q, using the operator $\|_A$, enforcing synchronisation on all visible actions. The environment X can be seen as a process $\sum_{i \in I} a_i.R_i + \mathrm{t}.R$ where $\{a_i \mid i \in I\} = X$. A triggered environment, on the other hand, can execute a sequence of instantaneous hidden actions before stabilising as an environment Y, for $Y \subseteq A$. During this execution, actions can be blocked and allowed in rapid succession. Since the environment is unknown, the bisimulation should be robust under any such environment.

The first clause for $(P, X, Q) \in \mathscr{R}$ is like the common transfer property of strong bisimilarity [29]: a visible a-transition of P can be matched by one of Q, such that the resulting processes P' and Q' are related again. However, I require it only for actions $a \in X$, because actions $b \in A \backslash X$ cannot happen at all in the environment X, and thus need not be matched by Q. Since the occurrence of a is observable by the environment, this can trigger the environment to change the set of actions it allows, so P' and Q' ought to be related in a triggered environment.

The second clause is the transfer property for τ-transitions. Since these are not observable by the environment, they cannot trigger a change in the set of actions allowed by it, so the resulting processes P' and Q' should be related only in the same environment X.

The first clause for $(P, Q) \in \mathscr{R}$ expresses the transfer property for τ-transitions in a triggered environment. Here it may happen that the τ-transition occurs before the environment stabilises, and hence P' and Q' will still be related in a triggered environment. A similar transfer property for a-transitions is already implied by the next two clauses.

The second clause allows a triggered environment to stabilise into any environment X.

The first two clauses for $(P, X, Q) \in \mathscr{R}$ imply that if $(P, X, Q) \in \mathscr{R}$ then $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \mathcal{I}(Q) \cap (X \cup \{\tau\})$. So $P \leftrightarrows_r Q$ implies $\mathcal{I}(P) = \mathcal{I}(Q)$. The condition $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ is necessary and sufficient for the system to remain a positive amount of time in state P when

X is the set of allowed actions. The next clause says that during this time the environment may be triggered to change the set of actions it allows by an event outside our model, that is, by a time-out in the environment. So P and Q should be related in a triggered environment.

The last clause says that also a t-transition of P should be matched by one of Q. Naturally, the t-transition of P can be taken only when the system is idling in P, i.e., when $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$. The resulting processes P' and Q' should be related again, but only in the same environment allowing X.

▶ **Proposition 2** (@). *$\leftrightarrows_r$ and $\leftrightarrows_r^X$ for $X \subseteq A$ are equivalence relations.*

Note that the union of arbitrarily many strong reactive bisimulations is itself a strong reactive bisimulation. Consequently, the family of relations $\leftrightarrows_r$, $\leftrightarrows_r^X$ for $X \subseteq A$ can be seen as a strong reactive bisimulation.

To get a firm grasp on strong reactive bisimilarity, the reader is invited to check the two laws mentioned in the introduction, and then to construct a strong reactive bisimulation between the two systems depicted on Page 14.

2.1 A more general form of reactive bisimulation

The following notion of a *generalised strong reactive bisimulation* (gsrb) generalises that of a strong reactive bisimulation; yet it induces the same concept of strong reactive bisimilarity. This makes the relation convenient to use for further analysis. I did not introduce it as the original definition, because it lacks a strong motivation.

▶ **Definition 3.** *A* gsrb *is a symmetric relation $\mathscr{R} \subseteq (\mathbb{P} \times \mathscr{P}(A) \times \mathbb{P}) \cup (\mathbb{P} \times \mathbb{P})$ such that, for all $(P, Q) \in \mathscr{R}$,*

- *if $P \xrightarrow{\alpha} P'$ with $a \in A \cup \{\tau\}$, then there exists a Q' such that $Q \xrightarrow{\alpha} Q'$ and $(P', Q') \in \mathscr{R}$,*
- *if $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ with $X \subseteq A$ and $P \xrightarrow{\mathrm{t}} P'$, then there exists a Q' with $Q \xrightarrow{\mathrm{t}} Q'$ and $(P', X, Q') \in \mathscr{R}$,*

and for all $(P, Y, Q) \in \mathscr{R}$,

- *if $P \xrightarrow{a} P'$ with either $a \in Y$ or $\mathcal{I}(P) \cap (Y \cup \{\tau\}) = \emptyset$, then there exists a Q' with $Q \xrightarrow{a} Q'$ and $(P', Q') \in \mathscr{R}$,*
- *if $P \xrightarrow{\tau} P'$, then there exists a Q' such that $Q \xrightarrow{\tau} Q'$ and $(P', Y, Q') \in \mathscr{R}$,*
- *if $\mathcal{I}(P) \cap (X \cup Y \cup \{\tau\}) = \emptyset$ with $X \subseteq A$ and $P \xrightarrow{\mathrm{t}} P'$ then there exists a Q' with $Q \xrightarrow{\mathrm{t}} Q'$ and $(P', X, Q') \in \mathscr{R}$.*

▶ **Proposition 4** (@). *$P \leftrightarrows_r Q$ iff there exists a gsrb $\mathscr{R}$ with $(P, Q) \in \mathscr{R}$.*
Likewise, $P \leftrightarrows_r^X Q$ iff there exists a gsrb $\mathscr{R}$ with $(P, X, Q) \in \mathscr{R}$.

Unlike Definition 1, a gsrb needs the triples (P, X, Q) only after encountering a t-transition; two systems without t-transitions can be related without using these triples at all.

3 A modal characterisation of strong reactive bisimilarity

The Hennessy-Milner logic [23] expresses properties on the behaviour of processes in an LTS.

▶ **Definition 5.** *The class $\mathbb{O}$ of* infinitary HML formulas *is defined as follows, where I ranges over all index sets and α over $A \cup \{\tau\}$:*

$$\varphi \quad ::= \quad \bigwedge_{i \in I} \varphi_i \quad | \quad \neg\varphi \quad | \quad \langle\alpha\rangle\varphi$$

$\top$ abbreviates the empty conjunction, and $\varphi_1 \wedge \varphi_2$ stands for $\bigwedge_{i \in \{1,2\}} \varphi_i$.

$P \models \varphi$ denotes that process P satisfies formula φ. The first two operators represent standard Boolean operators. By definition, $P \models \langle\alpha\rangle\varphi$ iff $P \xrightarrow{\alpha} P'$ for some P' with $P' \models \varphi$.

A famous result stemming from [23] states that

$$P \leftrightarrows Q \quad \Leftrightarrow \quad \forall \varphi \in \mathbb{O}.\ (P \models \varphi \Leftrightarrow Q \models \varphi)$$

where $\leftrightarrows$ denotes strong bisimilarity [29, 23], formally defined in Section 7. It states that the Hennessy-Milner logic yields a *modal characterisation* of strong bisimilarity. I will now adapt this into a modal characterisation of strong reactive bisimilarity.

To this end I extend the Hennessy-Milner logic with a new modality $\langle X\rangle$, for $X \subseteq A$, and auxiliary satisfaction relations $\models_X \subseteq \mathbb{P} \times \mathbb{O}$ for each $X \subseteq A$. The formula $P \models \langle X\rangle\varphi$ says that in an environment X, allowing exactly the actions in X, process P can perform a time-out transition to a process that satisfies φ. $P \models_X \varphi$ says that if P satisfies φ when placed in environment X. The relations $\models$ and $\models_X$ are the smallest ones satisfying:

$P \models \bigwedge_{i\in I} \varphi_i$	if	$\forall i \in I.\ P \models \varphi_i$
$P \models \neg\varphi$	if	$P \not\models \varphi$
$P \models \langle\alpha\rangle\varphi$ with $\alpha \in A \cup \{\tau\}$	if	$\exists P'.\ P \xrightarrow{\alpha} P' \wedge P' \models \varphi$
$P \models \langle X\rangle\varphi$ with $X \subseteq A$	if	$\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset \wedge \exists P'.\ P \xrightarrow{\mathrm{t}} P' \wedge P' \models_X \varphi$
$P \models_X \bigwedge_{i\in I} \varphi_i$	if	$\forall i \in I.\ P \models_X \varphi_i$
$P \models_X \neg\varphi$	if	$P \not\models_X \varphi$
$P \models_X \langle a\rangle\varphi$ with $a \in A$	if	$a \in X \wedge \exists P'.\ P \xrightarrow{a} P' \wedge P' \models \varphi$
$P \models_X \langle\tau\rangle\varphi$	if	$\exists P'.\ P \xrightarrow{\tau} P' \wedge P' \models_X \varphi$
$P \models_X \varphi$	if	$\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset \wedge P \models \varphi$

Note that a formula $\langle a\rangle\varphi$ is less often true under $\models_X$ then under $\models$, due to the side condition $a \in X$. This reflects the fact that a cannot happen in an environment that blocks it. The last clause in the above definition reflects the fifth clause of Definition 1. If $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$, then process P, operating in environment X, idles for a while, during which the environment can change. This ends the blocking of actions $a \notin X$ and makes any formula valid under $\models$ also valid under $\models_X$.

▶ **Example 6.** Both systems depicted on Page 14 satisfy $\langle\{a,b\}\rangle\langle\tau\rangle\langle b\rangle\top \wedge \langle\{a,b\}\rangle\langle\tau\rangle\neg\langle b\rangle\top \wedge \langle\{b\}\rangle\langle a\rangle\top \wedge \langle\{b\}\rangle\neg\langle a\rangle\top$ and neither satisfies $\langle\{a,b\}\rangle(\langle a\rangle \wedge \langle\tau\rangle\langle b\rangle)$ or $\langle\{b\}\rangle(\langle a\rangle \wedge \langle\tau\rangle\langle b\rangle)$.

▶ **Theorem 7** (@). *Let $P, Q \in \mathbb{P}$ and $X \subseteq A$. Then $P \leftrightarrows_r Q \Leftrightarrow \forall \varphi \in \mathbb{O}.\ (P \models \varphi \Leftrightarrow Q \models \varphi)$ and $P \leftrightarrows_r^X Q \Leftrightarrow \forall \varphi \in \mathbb{O}.\ (P \models_X \varphi \Leftrightarrow Q \models_X \varphi)$.*

4 Time-out bisimulations

I will now present a characterisation of strong reactive bisimilarity in terms of a binary relation $\mathscr{B}$ on processes – a *strong time-out bisimulation* – not parametrised by the set of allowed actions X. To this end I need a family of unary operators θ_X on processes, for $X \subseteq A$. These *environment* operators place a process in an environment that allows exactly the actions in X to occur. They are defined by the following structural operational rules.

$$\frac{x \xrightarrow{\tau} y}{\theta_X(x) \xrightarrow{\tau} \theta_X(y)} \qquad \frac{x \xrightarrow{a} y}{\theta_X(x) \xrightarrow{a} y}\ (a \in X) \qquad \frac{x \xrightarrow{\alpha} y \quad x \not\xrightarrow{\beta} \text{ for all } \beta \in X \cup \{\tau\}}{\theta_X(x) \xrightarrow{\alpha} y}\ (\alpha \in A \cup \{\mathrm{t}\})$$

The operator θ_X modifies its argument by inhibiting all initial transitions (here including also those that occur after a τ-transition) that cannot occur in the specified environment. When an observable transition does occur, the environment may be triggered to change, and the inhibiting effect of the θ_X-operator comes to an end. The premises $x \stackrel{\beta}{\not\longrightarrow}$ for all $\beta \in X \cup \{\tau\}$ in the third rule guarantee that the process x will idle for a positive amount of time in its current state. During this time, the environment may be triggered to change, and again the inhibiting effect of the θ_X-operator comes to an end.

Below I assume that $\mathbb{P}$ is closed under θ, that is, if $P \in \mathbb{P}$ and $X \subseteq A$ then $\theta_X(P) \in \mathbb{P}$.

▶ **Definition 8.** *A* strong time-out bisimulation *is a symmetric relation $\mathscr{B} \subseteq \mathbb{P} \times \mathbb{P}$, such that, for $P \mathscr{B} Q$,*

- *if $P \xrightarrow{\alpha} P'$ with $\alpha \in A \cup \{\tau\}$, then $\exists Q'$ such that $Q \xrightarrow{\alpha} Q'$ and $P' \mathscr{B} Q'$,*
- *if $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{\mathrm{t}} P'$, then $\exists Q'$ such that $Q \xrightarrow{\mathrm{t}} Q'$ and $\theta_X(P') \mathscr{B} \theta_X(Q')$.*

▶ **Proposition 9** (@). *$P \leftrightarrows_r Q$ iff there exists a strong time-out bisimulation $\mathscr{B}$ with $P \mathscr{B} Q$.*

Note that the union of arbitrarily many strong time-out bisimulations is itself a strong time-out bisimulation. Consequently, the relation $\leftrightarrows_r$ is a strong time-out bisimulation.

5 The process algebra $\mathrm{CCSP}^{\theta}_{\mathrm{t}}$

Let A be a set of *visible actions* and *Var* an infinite set of *variables*. The syntax of $\mathrm{CCSP}^{\theta}_{\mathrm{t}}$ is given by

$$E ::= 0 \mid \alpha.E \mid E{+}E \mid E\|_S E \mid \tau_I(E) \mid \mathcal{R}(E) \mid \theta^U_L(E) \mid \psi_X(E) \mid x \mid \langle x|\mathcal{S}\rangle \text{ (with } x{\in}V_{\mathcal{S}})$$

with $\alpha \in Act := A \uplus \{\tau, \mathrm{t}\}$, $S, I, U, L, X \subseteq A$, $L \subseteq U$, $\mathcal{R} \subseteq A \times A$, $x \in Var$ and $\mathcal{S}$ a *recursive specification*: a set of equations $\{y = \mathcal{S}_y \mid y \in V_{\mathcal{S}}\}$ with $V_{\mathcal{S}} \subseteq Var$ (the *bound variables* of $\mathcal{S}$) and each $\mathcal{S}_y$ a $\mathrm{CCSP}^{\theta}_{\mathrm{t}}$ expression. I require that all sets $\{b \mid (a,b) \in \mathcal{R}\}$ are finite.

The fragment of $\mathrm{CCSP}^{\theta}_{\mathrm{t}}$ without θ^U_L and ψ_X is called $\mathrm{CCSP}_{\mathrm{t}}$. Omitting t._ yields CCSP.

The constant 0 represents a process that is unable to perform any action. The process $\alpha.E$ first performs the action α and then proceeds as E. The process $E + F$ behaves as either E or F. $\|_S$ is a partially synchronous parallel composition operator; actions $a \in S$ must synchronise – they can occur only when both arguments are ready to perform them – whereas actions $\alpha \notin S$ from both arguments are interleaved. τ_I is an abstraction operator; it conceals the actions in I by renaming them into the hidden action τ. The operator $\mathcal{R}$ is a relational renaming: it renames a given action $a \in A$ into a choice between all actions b with $(a,b) \in \mathcal{R}$. The *environment operators* θ^U_L and ψ_X are new in this paper and explained below. Finally, $\langle x|\mathcal{S}\rangle$ represents the x-component of a solution of the system of recursive equations $\mathcal{S}$.

The language CCSP is a common mix of the process algebras CCS [29] and CSP [4, 24]. It first appeared in [30], where it was named following a suggestion by M. Nielsen. The family of parallel composition operators $\|_S$ stems from [31], and incorporates the two CSP parallel composition operators from [4]. The relation renaming operators $\mathcal{R}(\_)$ stem from [34]; they combine both the (functional) renaming operators that are common to CCS and CSP, and the inverse image operators of CSP. The choice operator + stems from CCS, and the abstraction operator from CSP, while the inaction constant 0, action prefixing operators $a.\_$ for $a \in A$, and the recursion construct are common to CCS and CSP. The time-out prefixing operator t._ was added by me in [16]. The syntactic form of inaction 0, action prefixing $\alpha.E$ and choice $E + F$ follows CCS, whereas the syntax of abstraction $\tau_I(\_)$ and recursion $\langle x|\mathcal{S}\rangle$ follows ACP [1, 8].

Table 1 Structural operational interleaving semantics of $\mathrm{CCSP}_{\mathrm{t}}^{\theta}$.

$$\alpha.x \xrightarrow{\alpha} x \qquad \frac{x \xrightarrow{\alpha} x'}{x+y \xrightarrow{\alpha} x'} \qquad \frac{y \xrightarrow{\alpha} y'}{x+y \xrightarrow{\alpha} y'} \qquad \frac{x \xrightarrow{\alpha} x'}{\mathcal{R}(x) \xrightarrow{\beta} \mathcal{R}(x')} \begin{pmatrix} \alpha{=}\beta{=}\tau \\ \vee\ \alpha{=}\beta{=}\mathrm{t} \\ \vee\ (\alpha,\beta)\in\mathcal{R} \end{pmatrix}$$

$$\frac{x \xrightarrow{\alpha} x'}{x \parallel_S y \xrightarrow{\alpha} x' \parallel_S y}\ (\alpha \notin S) \qquad \frac{x \xrightarrow{a} x' \quad y \xrightarrow{a} y'}{x \parallel_S y \xrightarrow{a} x' \parallel_S y'}\ (a \in S) \qquad \frac{y \xrightarrow{\alpha} y'}{x \parallel_S y \xrightarrow{\alpha} x \parallel_S y'}\ (\alpha \notin S)$$

$$\frac{x \xrightarrow{\alpha} x'}{\tau_I(x) \xrightarrow{\alpha} \tau_I(x')}\ (\alpha \notin I) \qquad \frac{x \xrightarrow{a} x'}{\tau_I(x) \xrightarrow{\tau} \tau_I(x')}\ (a \in I) \qquad \frac{\langle \mathcal{S}_x | \mathcal{S} \rangle \xrightarrow{\alpha} y}{\langle x | \mathcal{S} \rangle \xrightarrow{\alpha} y}$$

$$\frac{x \xrightarrow{\tau} y}{\theta_L^U(x) \xrightarrow{\tau} \theta_L^U(y)} \qquad \frac{x \xrightarrow{a} y}{\theta_L^U(x) \xrightarrow{a} y}\ (a \in U) \qquad \frac{x \xrightarrow{\alpha} y \quad x \xrightarrow{\beta}\!\!\!\!\!\!/\ \text{for all } \beta \in L \cup \{\tau\}}{\theta_L^U(x) \xrightarrow{\alpha} y}\ (\alpha \in A \cup \{\mathrm{t}\})$$

$$\frac{x \xrightarrow{\alpha} y}{\psi_X(x) \xrightarrow{\alpha} y}\ (\alpha \in A \cup \{\tau\}) \qquad \frac{x \xrightarrow{\mathrm{t}} y \quad x \xrightarrow{\beta}\!\!\!\!\!\!/\ \text{for all } \beta \in X \cup \{\tau\}}{\psi_X(x) \xrightarrow{\mathrm{t}} \theta_X(y)}$$

An occurrence of a variable x in a $\mathrm{CCSP}_{\mathrm{t}}^{\theta}$ expression E is *bound* iff it occurs in a subexpression $\langle y|\mathcal{S}\rangle$ of E with $x \in V_{\mathcal{S}}$; otherwise it is *free*. Here each $\mathcal{S}_y$ for $y \in V_{\mathcal{S}}$ counts as a subexpression of $\langle x|\mathcal{S}\rangle$. An expression E is *invalid* if it has a subexpression $\theta_L^U(F)$ or $\psi_X(F)$ such that a variable occurrence in F is free in F but bound in E. Let $\mathbb{E}$ be the set of valid $\mathrm{CCSP}_{\mathrm{t}}^{\theta}$ expressions. Furthermore, $\mathbb{P} \subseteq \mathbb{E}$ is the set of *closed* valid $\mathrm{CCSP}_{\mathrm{t}}^{\theta}$ expressions, or *processes*; those in which every variable occurrence is bound.

A substitution is a partial function $\rho\colon Var \rightharpoonup \mathbb{E}$. The application $E[\rho]$ of a substitution ρ to an expression $E \in \mathbb{E}$ is the result of simultaneous replacement, for all $x \in \mathrm{dom}(\rho)$, of each free occurrence of x in E by the expression $\rho(x)$, while renaming bound variables in E if necessary to present name clashes.

The semantics of $\mathrm{CCSP}_{\mathrm{t}}^{\theta}$ is given by the labelled transition relation $\rightarrow \subseteq \mathbb{P} \times Act \times \mathbb{P}$, where the transitions $P \xrightarrow{\alpha} Q$ are derived from the rules of Table 1. Here $\langle E|\mathcal{S}\rangle$ for $E \in \mathbb{E}$ and $\mathcal{S}$ a recursive specification denotes the result of substituting $\langle y|\mathcal{S}\rangle$ for y in E, for all $y \in V_{\mathcal{S}}$. Even though negative premises occur in Table 1, the meaning of this transition system specification is well-defined, for instance by the method of *stratification* [21, 13]. This is explained in the appendix (@).

The auxiliary operators θ_L^U and ψ_X are added here to facilitate complete axiomatisation, similar to the left merge and communication merge of ACP [1, 8]. The operator θ_X^X is the same as what was called θ_X in Section 4. It inhibits those transitions of its argument that are blocked in the environment X, allowing only the actions from $X \subseteq A$. It stops inhibiting as soon as the system performs a visible action or takes a break, as this may trigger a change in the environment. The operator θ_L^U preserves those transitions that are allowed in some environment X with $L \subseteq X \subseteq U$. The letters L and U stand for *lower* and *upper*bound. The operator ψ_X places a process in the environment X when a time-out transition occurs; it is inert if any other transition occurs. If $P \xrightarrow{\beta}$ for $\beta \in A \cup \{\tau\}$, then a time-out transition $P \xrightarrow{\mathrm{t}} Q$ cannot occur in an environment that allows β. Thus the transition $P \xrightarrow{\mathrm{t}} Q$ survives only when considering an environments that blocks β, meaning $\beta \notin X \cup \{\tau\}$. Taking the contrapositive, $\beta \in X \cup \{\tau\}$ implies $P \xrightarrow{\beta}\!\!\!\!\!\!/$.

The operator $\theta_{\emptyset}^U$ features in the forthcoming law (L3), which is a convenient addition to my axiomatisation, although only ψ_X and θ_X $(= \theta_X^X)$ are necessary for completeness.

6 Guarded recursion and finitely branching processes

In many process algebraic treatments, only guarded recursive specifications are allowed.

▶ **Definition 10.** *An occurrence of a variable x in an expression E is* guarded *if x occurs in a subexpression $\alpha.F$ of E, with $\alpha \in Act$. An expression E is* guarded *if all free occurrences of variables in E are guarded. A recursive specification $\mathcal{S}$ is* manifestly guarded *if all expressions $\mathcal{S}_y$ for $y \in V_{\mathcal{S}}$ are guarded. It is* guarded *if it can be converted into a manifestly guarded recursive specification by repeated substitution of expressions $\mathcal{S}_y$ for variables $y \in V_{\mathcal{S}}$ occurring in the expressions $\mathcal{S}_z$ for $z \in V_{\mathcal{S}}$. Let* guarded $CCSP^{\theta}_{\rm t}$ *be the fragment of $CCSP^{\theta}_{\rm t}$ allowing only guarded recursion.*

▶ **Definition 11.** *The set of processes* reachable *from a process P is inductively defined by*
(i) *P is reachable from P, and*
(ii) *if Q is reachable from P and $Q \xrightarrow{\alpha} R$ for some $\alpha \in Act$ then R is reachable from P.*
A process P is finitely branching *if for all $Q \in \mathbb{P}$ reachable from P there are only finitely many processes R such that $Q \xrightarrow{\alpha} R$ for some $\alpha \in Act$. Likewise, P is* countably branching *if there are countably many.*

▶ **Proposition 12** (@). *Each $CCSP^{\theta}_{\rm t}$ process is countably branching.*

▶ **Proposition 13** (@). *Each $CCSP^{\theta}_{\rm t}$ process with guarded recursion is finitely branching.*

▶ **Proposition 14** ([11], @). *Each finitely branching processes in an LTS can be denoted by a $CCSP_{\rm t}$ expression with guarded recursion, only using the operations 0, $\alpha.\_$ and $+$.*

▶ **Proposition 15** ([11], @). *Each countably branching processes in an LTS can be denoted by a $CCSP_{\rm t}$ expression. Again I only need the $CCSP_{\rm t}$ operations 0, $\alpha.\_$, $+$ and recursion.*

7 Congruence

Given an arbitrary process algebra with a collection of operators f, each with an arity n, and a recursion construct $\langle x|\mathcal{S}\rangle$ as in Section 5, let $\mathbb{P}$ and $\mathbb{E}$ be the sets of [closed] valid expressions, and let a substitution instance $E[\rho] \in \mathbb{E}$ for $E \in \mathbb{E}$ and $\rho : Var \rightharpoonup \mathbb{E}$ be defined as in Section 5. Any semantic equivalence $\sim \subseteq \mathbb{P} \times \mathbb{P}$ extends to $\sim \subseteq \mathbb{E} \times \mathbb{E}$ by defining $E \sim F$ iff $E[\rho] \sim F[\rho]$ for each closed substitution $\rho : Var \to \mathbb{P}$. It extends to substitutions $\rho, \nu : Var \rightharpoonup \mathbb{E}$ by $\rho \sim \nu$ iff $\rho(x) \sim \nu(x)$ for each $x \in \mathrm{dom}(\rho)$.

▶ **Definition 16** ([14]). A semantic equivalence $\sim$ is a *lean congruence* if $E[\rho] \sim E[\nu]$ for any expression $E \in \mathbb{E}$ and any substitutions ρ and ν with $\rho \sim \nu$. It is a *full congruence* if

$$P_i \sim Q_i \text{ for all } i = 1, ..., n \quad \Rightarrow \quad f(P_1, ..., P_n) \sim f(Q_1, ..., Q_n) \tag{1}$$

$$\mathcal{S}_y \sim \mathcal{S}'_y \text{ for all } y \in V_{\mathcal{S}} \quad \Rightarrow \quad \langle x|\mathcal{S}\rangle \sim \langle x|\mathcal{S}'\rangle \tag{2}$$

for all functions f of arity n, processes $P_i, Q_i \in \mathbb{P}$, and recursive specifications $\mathcal{S}, \mathcal{S}'$ with $x \in V_{\mathcal{S}} = V_{\mathcal{S}'}$ and $\langle x|\mathcal{S}\rangle, \langle x|\mathcal{S}'\rangle \in \mathbb{P}$.

Clearly, each full congruence is also a lean congruence, and each lean congruence satisfies (1) above. Both implications are strict, as illustrated in [14].

A main result of the present paper will be that strong reactive bisimilarity is a full congruence for the process algebra $\mathrm{CCSP}^{\theta}_{\rm t}$. To achieve it I need to establish first that strong bisimilarity [29], $\leftrightarrows$, and initials equivalence [12, Section 16], $=_{\mathcal{I}}$, are full congruences.

▶ **Definition 17.** *$CCSP^\theta_{\mathrm{t}}$ processes P and Q are* initials equivalent*, $P =_{\mathcal{I}} Q$, if $\mathcal{I}(P) = \mathcal{I}(Q)$.*

▶ **Theorem 18.** *Initials equivalence is a full congruence for $CCSP^\theta_{\mathrm{t}}$.*

▶ **Definition 19.** *A* strong bisimulation *is a symmetric relation $\mathscr{B}$, such that, when $P \mathscr{B} Q$,*
- *if $P \xrightarrow{\alpha} P'$ with $\alpha \in Act$ then $Q \xrightarrow{\alpha} Q'$ for some Q' with $P' \mathscr{B} Q'$.*

Two processes $P, Q \in \mathbb{P}$ are strongly bisimilar*, $P \leftrightarrows Q$, if $P \mathscr{R} Q$ for a strong bisimulation $\mathscr{B}$.*

Contrary to reactive bisimilarity, strong bisimilarity treats the time-out action t, as well as the hidden action τ, just like any visible action. In the absence of time-out actions, there is no difference between a strong bisimulation and a time-out bisimulation, so $\leftrightarrows_r$ and $\leftrightarrows$ coincide. In general, strong bisimulation is a finer equivalence relation: $P \leftrightarrows Q \Rightarrow P \leftrightarrows_r Q \Rightarrow P =_{\mathcal{I}} Q$, and both implications are strict.

▶ **Theorem 20** (@)**.** *Strong bisimilarity is a full congruence for $CCSP^\theta_{\mathrm{t}}$.*

8 Strong reactive bisimilarity is a full congruence for $\mathbf{CCSP}^\theta_{\mathbf{t}}$

The proofs showing that $\leftrightarrows_r$ is a full congruence for $\mathrm{CCSP}^\theta_{\mathrm{t}}$ follow the lines of Milner [29], but are more complicated due to the nature of reactive bisimilarity. A crucial tool is Milner's notion of *bisimilarity up-to*. Even if we would not be interested in the operators θ^U_L and ψ_X, the proof needs to take the operator θ_X $(= \theta^X_X)$ along in order to deal with the other operators. This is a consequence of the occurrence of θ_X in Definition 8.

▶ **Definition 21.** *Given a relation $\sim \subseteq \mathbb{P} \times \mathbb{P}$, a* strong time-out bisimulation up to $\sim$ *is a symmetric relation $\mathscr{B} \subseteq \mathbb{P} \times \mathbb{P}$, such that, for $P \mathscr{B} Q$,*
- *if $P \xrightarrow{\alpha} P'$ with $\alpha \in A \cup \{\tau\}$, then $\exists Q'$ such that $Q \xrightarrow{\alpha} Q'$ and $P' \sim \mathscr{B} \sim Q'$,*
- *if $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{\mathrm{t}} P'$, then $\exists Q'$ with $Q \xrightarrow{\mathrm{t}} Q'$ and $\theta_X(P') \sim \mathscr{B} \sim \theta_X(Q')$.*

Here $\sim \mathscr{B} \sim := \{(R, T) \mid \exists R', T'.\ R \sim R' \mathscr{B} T' \sim T\}$.

▶ **Proposition 22.** *If $P \mathscr{B} Q$ for a strong time-out bisimulation $\mathscr{B}$ up to $\leftrightarrows$, then $P \leftrightarrows_r Q$.*

▶ **Theorem 23.** *Strong reactive bisimilarity is a lean congruence for $CCSP^\theta_{\mathrm{t}}$. In other words, if $\rho, \nu : Var \rightharpoonup \mathbb{E}$ are substitutions with $\rho \leftrightarrows_r \nu$, then $E[\rho] \leftrightarrows_r E[\nu]$ for any expression $E \in \mathbb{E}$.*

▶ **Proposition 24.** *If $P \mathscr{B} Q$ for a strong time-out bisimulation $\mathscr{B}$ up to $\leftrightarrows_r$, then $P \leftrightarrows_r Q$.*

▶ **Theorem 25.** *Strong reactive bisimilarity is a full congruence for $CCSP^\theta_{\mathrm{t}}$.*

9 The Recursive Specification Principle

For $W \subseteq Var$ a set of variables, a W-tuple of expressions is a function $\vec{E} \in \mathbb{E}^W$. It has a component $\vec{E}(x)$ for each variable $x \in W$. Note that a W-tuple of expressions is nothing else than a substitution. Let id_W be the identity function, given by $id_W(x) = x$ for all $x \in W$. If $G \in \mathbb{E}$ and $\vec{E} \in \mathbb{E}^W$ then $G[\vec{E}]$ denotes the result of simultaneous substitution of $\vec{E}(x)$ for x in G, for all $x \in W$. Likewise, if $\vec{G} \in \mathbb{E}^V$ and $\vec{E} \in \mathbb{E}^W$ then $\vec{G}[\vec{E}] \in \mathbb{E}^V$ denotes the V-tuple with components $G(y)[\vec{E}]$ for $y \in V$. Henceforth, I regard a recursive specification $\mathcal{S}$ as a $V_{\mathcal{S}}$-tuple with components $\mathcal{S}(y) = \mathcal{S}_y$ for $y \in V_{\mathcal{S}}$. If $\vec{E} \in \mathbb{E}^W$ and $\mathcal{S} \in \mathbb{E}^V$, then $\langle \vec{E} | \mathcal{S} \rangle \in \mathbb{E}^W$ is the W-tuple with components $\langle \vec{E}(x) | \mathcal{S} \rangle \in \mathbb{E}^W$ for $x \in W$.

For $\mathcal{S}$ a recursive specification and $\vec{E} \in \mathbb{E}^{V_{\mathcal{S}}}$ a $V_{\mathcal{S}}$-tuple of expressions, $\vec{E} \leftrightarrows_r \mathcal{S}[\vec{E}]$ states that $\vec{E}$ is a *solution* of $\mathcal{S}$, up to strong reactive bisimilarity. The tuple $\langle id_{V_{\mathcal{S}}} | \mathcal{S} \rangle \in \mathbb{E}^{V_{\mathcal{S}}}$ is called the *default solution*.

Table 2 A complete axiomatisation of strong bisimilarity on guarded $\mathrm{CCSP_t}$.

$$\begin{array}{lll}
x+(y+z)=(x+y)+z & \tau_I(x+y)=\tau_I(x)+\tau_I(y) & \mathcal{R}(x+y)=\mathcal{R}(x)+\mathcal{R}(y)\\
x+y=y+x & \tau_I(\alpha.x)=\alpha.\tau_I(x) \quad \text{if } \alpha\notin I & \mathcal{R}(\tau.x)=\tau.\mathcal{R}(x)\\
x+x=x & \tau_I(\alpha.x)=\tau.\tau_I(x) \quad \text{if } \alpha\in I & \mathcal{R}(\mathrm{t}.x)=\mathrm{t}.\mathcal{R}(x)\\
x+0=0 & \langle x|\mathcal{S}\rangle=\langle\mathcal{S}_x|\mathcal{S}\rangle & \mathcal{R}(a.x)=\sum_{\{b|(a,b)\in\mathcal{R}\}} b.\mathcal{R}(x)
\end{array}$$

If $P=\sum_{i\in I}\alpha_i.P_i$ and $Q=\sum_{j\in J}\beta_j.Q_j$ then

$$P\parallel_S Q=\sum_{i\in I,\ \alpha_i\notin S}(\alpha_i.P_i\parallel_S Q)+\sum_{j\in J,\ \beta_j\notin S}(P\parallel_S \beta_j.Q_j)+\sum_{i\in I,\ j\in J,\ \alpha_i=\beta_j\in S}\alpha_i.(P_i\parallel_S Q_j)$$

Recursive Specification Principle (RSP) $\quad \mathcal{S}\Rightarrow x=\langle x|\mathcal{S}\rangle$ $\quad$ ($\mathcal{S}$ guarded)

In [1, 8] two requirements occur for process algebras with recursion. The *recursive definition principle* (RDP) says that each recursive specification must have a solution, and the *recursive specification principle* (RSP) says that guarded recursive specifications have at most one solution. When dealing with process algebras where the meaning of a closed expression is a semantic equivalence class of processes, these principles become requirements on the semantic equivalence employed.

▶ **Proposition 26.** *Let $\mathcal{S}$ be a guarded recursive specification, and $x\in V_{\mathcal{S}}$. Then $\langle x|\mathcal{S}\rangle \leftrightarrows_r \langle\mathcal{S}_x|\mathcal{S}\rangle$.*

Proposition 26 says that the recursive definition principle holds for strong reactive bisimulation semantics. The "default solution" of a recursive specification is in fact a solution. Note that the conclusion of Proposition 26 can be restated as $\langle id_{V_{\mathcal{S}}}|\mathcal{S}\rangle \leftrightarrows_r \langle\mathcal{S}|\mathcal{S}\rangle$, and that $\mathcal{S}[\langle id_{V_{\mathcal{S}}}|\mathcal{S}\rangle]=\langle\mathcal{S}|\mathcal{S}\rangle$.

The following theorem establishes the recursive specification principle for strong reactive bisimulation semantics.

▶ **Theorem 27.** *Let $\mathcal{S}$ be a guarded recursive specification. If $\vec{E}\leftrightarrows_r\mathcal{S}[\vec{E}]$ and $\vec{F}\leftrightarrows_r\mathcal{S}[\vec{F}]$ with $\vec{E},\vec{F}\in\mathbb{E}^{V_{\mathcal{S}}}$, then $\vec{E}\leftrightarrows_r\vec{F}$.*

10 A complete axiomatisation for processes with guarded recursion

The well-known axioms of Table 2 are *sound* for strong bisimilarity, meaning that writing $\leftrightarrows$ for $=$, and substituting arbitrary expressions for the free variables x,y,z, or the meta-variables P_i and Q_j, turns them into true statements. In these axioms α,β range over Act and a,b over A. All axioms involving variables are equations. The axiom involving P and Q is a template that stands for a family of equations, one for each fitting choice of P and Q. This is the $\mathrm{CCSP_t}$ version of the *expansion law* from [29]. The axiom $\langle x|\mathcal{S}\rangle=\langle\mathcal{S}_x|\mathcal{S}\rangle$ says that recursively defined processes $\langle x|\mathcal{S}\rangle$ satisfy their set of defining equations $\mathcal{S}$. As discussed in the previous section, this entails that each recursive specification has a solution. The axiom RSP [1, 8] is a conditional equation, with as antecedents the equations of a guarded recursive specification $\mathcal{S}$. It says that the x-component of any solution of $\mathcal{S}$ – a vector of processes substituted for the variables $V_{\mathcal{S}}$ – equals $\langle x|\mathcal{S}\rangle$. In other words, each solution of $\mathcal{S}$ equals the default solution. This is a compact way of saying that solutions of guarded recursive specifications are unique.

▶ **Theorem 28.** *For $\mathrm{CCSP_t}$ processes $P,Q\in\mathbb{P}$ with guarded recursion, one has $P\leftrightarrows Q$, that is, P and Q are strongly bisimilar, iff $P=Q$ is derivable from the axioms of Table 2.*

Table 3 A complete axiomatisation of strong bisimilarity on guarded $\mathrm{CCSP}^{\theta}_{\mathrm{t}}$.

$\theta^U_L(\sum_{i\in I}\alpha_i.x_i)=\sum_{i\in I}\alpha_i.x_i$	$(\alpha_i\notin L\cup\{\tau\}$ for all $i\in I)$
$\theta^U_L(x+\alpha.y+\beta.z)=\theta^U_L(x+\alpha.y)$	$(\alpha\in L\cup\{\tau\}\wedge\beta\notin U\cup\{\tau\})$
$\theta^U_L(x+\alpha.y+\beta.z)=\theta^U_L(x+\alpha.y)+\theta^U_L(\beta.z)$	$(\alpha\in L\cup\{\tau\}\wedge\beta\in U\cup\{\tau\})$
$\theta^U_L(\beta.x)=\beta.x$	$(\beta\neq\tau)$
$\theta^U_L(\tau.x)=\tau.\theta^U_L(x)$	
$\psi_X(x+\alpha.z)=\psi_X(x)+\alpha.z$	$(\alpha\notin X\cup\{\tau,\mathrm{t}\})$
$\psi_X(x+\alpha.y+\mathrm{t}.z)=\psi_X(x+\alpha.y)$	$(\alpha\in X\cup\{\tau\})$
$\psi_X(x+\alpha.y+\beta.z)=\psi_X(x+\alpha.y)+\beta.z$	$(\alpha,\beta\in X\cup\{\tau\})$
$\psi_X(\alpha.x)=\alpha.x$	$(\alpha\neq\mathrm{t})$
$\psi_X(\sum_{j\in I}\mathrm{t}.y_i)=\sum_{j\in I}\mathrm{t}.\theta_X(y_j)$	

In this theorem, "if", the *soundness* of the axiomatisation of Table 2, is an immediate consequence of the soundness of the individual axioms. "Only if" states the *completeness* of the axiomatisation.

A crucial tool in its proof is the simple observation that the axioms from the first box of Table 2 allow any $\mathrm{CCSP}_{\mathrm{t}}$ process with guarded recursion to be brought in the form $\sum_{i\in I}\alpha_i.P_i$ – a *head normal form*. Using this, the rest of the proof is a standard argument employing RSP, independent of the choice of the specific process algebra. It can be found in [29], [1], [8] and many other places. However, in the literature this completeness theorem was always stated and proved for a small fragment of the process algebra, allowing only guarded recursive specifications with a finite number of equations, and whose right-hand sides $\mathcal{S}_y$ involve only the basic operators inaction, action prefixing and choice. Since the set of true statements $P\leftrightarrows Q$, with P and Q processes in a process algebra like guarded $\mathrm{CCSP}_{\mathrm{t}}$, is well-known to be undecidable, and even not recursively enumerable, is was widely believed that no sound and complete axiomatisation of strong bisimilarity could exist. Only in March 2017, Kees Middelburg observed (in the setting of the process algebra ACP [1, 8]) that the standard proof applies almost verbatim to arbitrary processes with guarded recursion, although one has to be a bit careful in dealing with the infinite nature of recursive specifications. The argument has been carefully documented in [19], in the setting of the process algebra ACP. This result does not contradict the non-enumerability of the set of true statements $P\leftrightarrows Q$, due to the fact that RSP is a proof rule with infinitely many premises.

Table 3 extends Table 2 with axioms for the auxiliary operators θ^U_L and ψ_X. With Table 1 it is straightforward to check the soundness of these axioms. The fourth axiom, for instance, follows from the second or third rule for θ^U_L in Table 1, depending on whether $\beta\in L\cup\{\mathrm{t}\}$. Moreover, a straightforward induction shows that these axioms suffice to convert each process into head normal form. Using this, we immediately obtain:

▶ **Theorem 29.** *For $\mathrm{CCSP}^{\theta}_{\mathrm{t}}$ processes $P,Q\in\mathbb{P}$ with guarded recursion, one has $P\leftrightarrows Q$ iff $P=Q$ is derivable from the axioms of Tables 2 and 3.*

A law that turns out to be particularly useful in verifications modulo strong reactive bisimilarity is

$$\theta^V_K(\theta^U_L(x))\leftrightarrows\theta^{V\cap U}_{K\cup L}(x)\qquad\text{provided } U=V \text{ or } K=L \text{ or } K\subseteq L\subseteq U\subseteq V \text{ or } L\subseteq K\subseteq V\subseteq U\qquad\text{(L1)}.$$

Table 4 A complete axiomatisation of strong reactive bisimilarity on guarded $\mathrm{CCSP}^\theta_{\mathrm{t}}$.

$$\frac{\psi_X(x)=\psi_X(y) \text{ for all } X\subseteq A}{x=y} \qquad \text{(RA)}$$

Note that the right-hand side only exists if $(K\cup L)\subseteq(V\cap U)$. This law is sound for strong bisimilarity, as demonstrated by the following proposition. Yet it is not needed to add it to Table 3, as all its closed instances are derivable. In fact, this is a consequence of the above completeness theorem.

▶ **Proposition 30.** *$\theta^V_K(\theta^U_L(P))\leftrightarroweq\theta^{V\cap U}_{K\cup L}(P)$, provided $(K\cup L)\subseteq(V\cap U)$ and either $U=V$ or $K=L$ or $K\subseteq L\subseteq U\subseteq V$ or $L\subseteq K\subseteq V\subseteq U$.*

To obtain a sound and complete axiomatisation of strong reactive bisimilarity for $\mathrm{CCSP}^\theta_{\mathrm{t}}$ with guarded recursion, one needs to combine the axioms of Tables 2, 3 and 4. These axioms are useful only in combination with the full congruence property of strong reactive bisimilarity, Theorem 25. This is what allows us to apply these axioms within subexpressions of a given expression. Since $\leftrightarroweq\subseteq\leftrightarroweq_r$, the soundness of all equational axioms for strong reactive bisimilarity follows from their soundness for strong bisimilarity. The soundness of RSP has been established as Theorem 27. The soundness of (RA), the *reactive approximation axiom*, is contributed by the following proposition.

▶ **Proposition 31.** *Let $P,Q\in\mathbb{P}$. If $\psi_X(P)\leftrightarroweq_r\psi_X(Q)$ for all $X\subseteq A$, then $P\leftrightarroweq_r Q$.*

The completeness of this axiomatisation is documented in the technical report accompanying this paper. Instead of the classic technique of *merging guarded recursive equations* [27, 28, 35, 9, 26], which in essence proves two bisimilar systems P and Q equivalent by equating both to an intermediate variant that is essentially a *product* of P and Q, it employs the novel method of *canonical solutions* [25], which equates both P and Q to a canonical representative within the bisimulation equivalence class of P and Q – one that has only one reachable state for each bisimulation equivalence class of states of P and Q.

Moreover, my proof employs the axiom of choice [36] in defining the transition relation on my canonical representative, in order to keep this process finitely branching.

At first sight it appears that axiom (RA) is not very handy, as the number of premises to verify is exponential in the size of the alphabet A of visible actions. In case A is infinite, there are even uncountably many premises. However, in practical verifications this is hardly an issue, as one uses a partition of the premises into a small number of equivalence classes, each of which requires only one common proof. This technique will be illustrated on three examples below. Furthermore, one could calculate the set of visible actions $\mathcal{J}(P)$ of a process P that can be encountered as initial actions after one t-transition followed by a sequence of τ-transitions. For large classes of processes, $\mathcal{J}(P)$ will be a finite set. Now axiom (RA) can be modified by changing $X\subseteq A$ into $X\subseteq\mathcal{J}(P)\cup\mathcal{J}(Q)$. This preserves the soundness of the axiom, because only the actions in $\mathcal{J}(P)$ play any rôle in evaluating $\psi_X(P)$.

A crucial property of strong reactive bisimilarity was mentioned in the introduction:

$$\tau.P+\mathrm{t}.Q=\tau.P \qquad \text{(L2)}\,.$$

It is an immediate consequence of (RA), since $\psi_X(\tau.P+\mathrm{t}.Q)=\psi_X(\tau.P)$ for any $X\subseteq A$, by Table 3. Another useful law in verifications modulo strong reactive bisimilarity is

$$\textstyle\sum_{i\in I}a_i.x_i+\mathrm{t}.y=\sum_{i\in I}a_i.x_i+\mathrm{t}.\theta^{A\setminus In}_\emptyset(y),\ \text{where } In=\{a_i\mid i\in I\}. \qquad \text{(L3)}$$

Its soundness is intuitively obvious: the t-transition to y will be taken only in an environment X with $X \cap In = \emptyset$. Hence one can just as well restrict the behaviour of y to those transitions that are allowed in one such environment. This law was one of the prime reasons for extending the family of operators θ_X $(= \theta^X_X)$, which were needed to establish the key theorems of this paper, to the larger family θ^U_L. Law (L3) for finite I is effortlessly derivable from its simple instance

$$\boxed{a.x + \mathrm{t}.y = a.x + \mathrm{t}.\theta^{A\setminus\{a\}}_{\emptyset}(y).} \qquad \text{(L3}'\text{)}$$

in combination with (L1). I now show how to derive (L3) from (RA). For this proof I need to partition the set of premises of (RA) in only two equivalence classes.

First let $X \cap In \neq \emptyset$. Then $\psi_X(\sum_{i\in I} a_i.x_i + \mathrm{t}.y) = \sum_{i\in I} a_i.x_i = \psi_X(\sum_{i\in I} a_i.x_i + \mathrm{t}.\theta^{A\setminus In}_{\emptyset}(y))$.

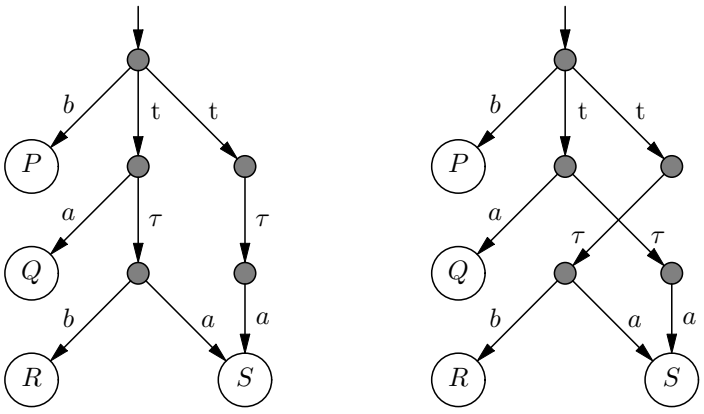
Next let $X \cap In = \emptyset$. Then
$$\begin{aligned}\psi_X(\textstyle\sum_{i\in I} a_i.x_i + \mathrm{t}.y) &= \textstyle\sum_{i\in I} a_i.x_i + \mathrm{t}.\theta_X(y)\\ &= \textstyle\sum_{i\in I} a_i.x_i + \mathrm{t}.\theta_X(\theta^{A\setminus In}_{\emptyset}(y))\\ &= \psi_X(\textstyle\sum_{i\in I} a_i.x_i + \mathrm{t}.\theta^{A\setminus In}_{\emptyset}(y))\ ,\end{aligned}$$
where the second step is an application of (L1).

As an application of (L3′) one obtains the law that was justified in the introduction:

$$\begin{aligned}a.P + \mathrm{t}.(Q + \tau.R + a.S) &= a.P + \mathrm{t}.\theta^{A\setminus\{a\}}_{\emptyset}(Q + \tau.R + a.S)\\ &= a.P + \mathrm{t}.\theta^{A\setminus\{a\}}_{\emptyset}(Q + \tau.R)\\ &= a.P + \mathrm{t}.(Q + \tau.R)\ .\end{aligned}$$

As a third illustration of the use of (RA) I derive an equational law that does not follow from (L1), (L2) and (L3), namely

$$b.P + \mathrm{t}.(a.Q + \tau.(b.R + a.S)) + \mathrm{t}.\tau.a.S = b.P + \mathrm{t}.(a.Q + \tau.a.S) + \mathrm{t}.\tau.(b.R + a.S)$$



These system are surely not strongly bisimilar. Moreover, (L3) does not help in proving them equivalent, as applying $\theta^{A\setminus\{b\}}_{\emptyset}$ to any of the four targets of a t-transition does not kill any of the transitions of those processes. In particular, $\theta^{A\setminus\{b\}}_{\emptyset}(b.R + a.S) = b.R + a.S$. To derive this law from (RA), I partition $\mathscr{P}(A)$ into three equivalence classes.

First let $b \in X$. Then
$$\begin{aligned}&\psi_X(b.P + \mathrm{t}.(a.Q + \tau.(b.R + a.S)) + \mathrm{t}.\tau.a.S)\\ &= b.P\\ &= \psi_X(b.P + \mathrm{t}.(a.Q + \tau.a.S) + \mathrm{t}.\tau.(b.R + a.S)).\end{aligned}$$

Next let $b \notin X$ and $a \in X$. Then

$$\begin{aligned}
&\psi_X\big(b.P + \mathrm{t}.(a.Q + \tau.(b.R + a.S)) + \mathrm{t}.\tau.a.S\big)\\
&= b.P + \mathrm{t}.\theta_X\big(a.Q + \tau.(b.R + a.S)\big) + \mathrm{t}.\theta_X\big(\tau.a.S\big)\\
&= b.P + \mathrm{t}.(a.Q + \tau.\theta_X(b.R + a.S)) + \mathrm{t}.\tau.\theta_X(a.S)\\
&= b.P + \mathrm{t}.(a.Q + \tau.a.S) + \mathrm{t}.\tau.a.S\\
&= b.P + \mathrm{t}.(a.Q + \tau.\theta_X(a.S)) + \mathrm{t}.\tau.\theta_X\big(b.R + a.S\big)\\
&= b.P + \mathrm{t}.\theta_X\big(a.Q + \tau.a.S\big) + \mathrm{t}.\theta_X\big(\tau.(b.R + a.S)\big)\\
&= \psi_X\big(b.P + \mathrm{t}.(a.Q + \tau.a.S) + \mathrm{t}.\tau.(b.R + a.S)\big)\ .
\end{aligned}$$

Finally let $a, b \notin X$. Then

$$\begin{aligned}
&\psi_X\big(b.P + \mathrm{t}.(a.Q + \tau.(b.R + a.S)) + \mathrm{t}.\tau.a.S\big)\\
&= b.P + \mathrm{t}.\theta_X\big(a.Q + \tau.(b.R + a.S)\big) + \mathrm{t}.\theta_X\big(\tau.a.S\big)\\
&= b.P + \mathrm{t}.\tau.\theta_X(b.R + a.S) + \mathrm{t}.\tau.\theta_X(a.S)\\
&= b.P + \mathrm{t}.\tau.(b.R + a.S) + \mathrm{t}.\tau.a.S\\
&= b.P + \mathrm{t}.\tau.a.S + \mathrm{t}.\tau.(b.R + a.S)\\
&= b.P + \mathrm{t}.\tau.\theta_X(a.S) + \mathrm{t}.\tau.\theta_X\big(b.R + a.S\big)\\
&= b.P + \mathrm{t}.\theta_X\big(a.Q + \tau.a.S\big) + \mathrm{t}.\theta_X\big(\tau.(b.R + a.S)\big)\\
&= \psi_X\big(b.P + \mathrm{t}.(a.Q + \tau.a.S) + \mathrm{t}.\tau.(b.R + a.S)\big)\ .
\end{aligned}$$

In words, the 4 processes that are targets of t-transitions always run in an environment that blocks b. In an environment that allows a, the branch $b.R$ disappears, so that the left branch of the first process can matched with the left branch of the second process, and similarly for the two right branches. In an environment that blocks a, this matching won't fly, as the branch $b.R$ now survives. However, the branches $a.Q$ will disappear, so that the left branch of the first process can be matched with the right branch of the second, and vice versa.

11 Concluding remarks

This paper laid the foundations of the proper analogue of strong bisimulation semantics for a process algebra with time-outs. This makes it possible to specify systems in this setting and verify their correctness properties. The addition of time-outs comes with considerable gains in expressive power. An illustration of this is mutual exclusion.

As shown in [17], it is fundamentally impossible to correctly specify mutual exclusion protocols in standard process algebras, such as CCS [29], CSP [4, 24], ACP [1, 8] or CCSP, unless the correctness of the specified protocol hinges on a fairness assumption. The latter, in the view of [17], does not provide an adequate solution, as fairness assumptions are in many situations unwarranted and lead to false conclusions. In [7] a correct process-algebraic rendering of mutual exclusion is given, but only after making two important modifications to standard process algebra. The first involves making a justness assumption. Here *justness* [18] is an alternative to fairness, in some sense a much weaker form of fairness – meaning weaker than weak fairness. Unlike (strong or weak) fairness, its use typically is warranted and does not lead to false conclusions. The second modification is the addition of a new construct – *signals* – to CCS, or any other standard process algebra. Interestingly, both modifications are necessary; just using justness, or just adding signals, is insufficient. Bouwman [2, 3] points out that since the justness requirement was fairly new, and needed to be carefully defined to describe its interaction with signals anyway, it is possible to specify mutual exclusion without adding signals to the language at all, instead reformulating the justness requirement in such a way that it effectively turns some actions into signals. Yet justness is essential in all these

approaches. This may be seen as problematic, because large parts of the foundations of process algebra are incompatible with justness, and hence need to be thoroughly reformulated in a justness-friendly way. This is pointed out in [15].

The addition of time-outs to standard process algebra makes it possible to specify mutual exclusion without assuming justness! Instead, one should make the assumption called *progress* in [18], which is weaker than justness, uncontroversial, unproblematic, and made (explicitly or implicitly) in virtually all papers dealing with issues like mutual exclusion. I will defer substantiation of this claim to a future occasion.

Besides applications to protocol verification, future work includes adapting the work done here to a form of reactive bisimilarity that abstracts from hidden actions, that is, to provide a counterpart for process algebras with time-outs of, for instance, branching bisimilarity [20], weak bisimilarity [29] or coupled similarity [32, 10]. Other topics worth exploring are the extension to probabilistic processes, and especially the relations with timed process algebras. Davies & Schneider in [5], for instance, added a construct with a quantified time-out to the process algebra CSP [4, 24], elaborating the timed model of CSP presented by Reed & Roscoe in [33].

References

1 J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science 18. Cambridge University Press, 1990. doi:10.1017/CBO9780511624193.

2 M.S. Bouwman. Liveness analysis in process algebra: simpler techniques to model mutex algorithms. Technical report, Eindhoven University of Technology, 2018. URL: http://www.win.tue.nl/~timw/downloads/bouwman_seminar.pdf.

3 M.S. Bouwman, B. Luttik, and T.A.C. Willemse. Off-the-shelf automated analysis of liveness properties for just paths. *Acta Informatica*, 57(3-5):551–590, 2020. doi:10.1007/s00236-020-00371-w.

4 S.D. Brookes, C.A.R. Hoare, and A.W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984. doi:10.1145/828.833.

5 J. Davies and S. Schneider. Recursion induction for real-time processes. *Formal Aspects of Computing*, 5(6):530–553, 1993. doi:10.1007/BF01211248.

6 R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984. doi:10.1016/0304-3975(84)90113-0.

7 V. Dyseryn, R.J. van Glabbeek, and P. Höfner. Analysing mutual exclusion using process algebra with signals. In K. Peters and S. Tini, editors, Proceedings Combined 24th International Workshop on *Expressiveness in Concurrency* and 14th Workshop on *Structural Operational Semantics,* Berlin, Germany, 4th September 2017, volume 255 of *Electronic Proceedings in Theoretical Computer Science*, pages 18–34. Open Publishing Association, 2017. doi:10.4204/EPTCS.255.2.

8 W. J. Fokkink. *Introduction to Process Algebra*. Texts in Theoretical Computer Science, An EATCS Series. Springer, 2000. doi:10.1007/978-3-662-04293-9.

9 R.J. van Glabbeek. A complete axiomatization for branching bisimulation congruence of finite-state behaviours. In A.M. Borzyszkowski and S. Sokołowski, editors, Proceedings 18^{th} International Symposium on *Mathematical Foundations of Computer Science,* MFCS '93, Gdansk, Poland, August/September 1993, volume 711 of LNCS, pages 473–484. Springer, 1993. doi:10.1007/3-540-57182-5_39.

10 R.J. van Glabbeek. The linear time – branching time spectrum II; the semantics of sequential systems with silent moves. In E. Best, editor, Proceedings *CONCUR'93,* 4^{th} International Conference on *Concurrency Theory,* Hildesheim, Germany, August 1993, volume 715 of LNCS, pages 66–81. Springer, 1993. doi:10.1007/3-540-57208-2_6.

11 R.J. van Glabbeek. On the expressiveness of ACP (extended abstract). In A. Ponse, C. Verhoef, and S.F.M. van Vlijmen, editors, Proceedings First Workshop on the *Algebra of Communicating Processes,* ACP'94, Utrecht, The Netherlands, May 1994, Workshops in Computing, pages 188–217. Springer, 1994. doi:10.1007/978-1-4471-2120-6_8.

12 R.J. van Glabbeek. The linear time – branching time spectrum I; the semantics of concrete, sequential processes. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, chapter 1, pages 3–99. Elsevier, 2001. doi:10.1016/B978-044482830-9/50019-9.

13 R.J. van Glabbeek. The meaning of negative premises in transition system specifications II. *Journal of Logic and Algebraic Programming*, 60–61:229–258, 2004. doi:10.1016/j.jlap.2004.03.007.

14 R.J. van Glabbeek. Lean and full congruence formats for recursion. In Proceedings 32^{nd} Annual ACM/IEEE Symposium on *Logic in Computer Science,* LICS'17, Reykjavik, Iceland, June 2017. IEEE Computer Society Press, 2017. doi:10.1109/LICS.2017.8005142.

15 R.J. van Glabbeek. Ensuring liveness properties of distributed systems: Open problems. *Journal of Logical and Algebraic Methods in Programming*, 109, 2019. doi:10.1016/j.jlamp.2019.100480.

16 R.J. van Glabbeek. Failure trace semantics for a process algebra with time-outs, 2020. arXiv:2002.10814.

17 R.J. van Glabbeek and P. Höfner. CCS: it's not fair! Fair schedulers cannot be implemented in CCS-like languages even under progress and certain fairness assumptions. *Acta Informatica*, 52(2-3):175–205, 2015. doi:10.1007/s00236-015-0221-6.

18 R.J. van Glabbeek and P. Höfner. Progress, justness and fairness. *ACM Computing Surveys*, 52(4), August 2019. doi:10.1145/3329125.

19 R.J. van Glabbeek and C.A. Middelburg. On infinite guarded recursive specifications in process algebra, 2020. arXiv:2005.00746.

20 R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996. doi:10.1145/233551.233556.

21 J.F. Groote. Transition system specifications with negative premises. *Theoretical Computer Science*, 118:263–299, 1993. doi:10.1016/0304-3975(93)90111-6.

22 J.F. Groote and F.W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, 100(2):202–260, October 1992. doi:10.1016/0890-5401(92)90013-6.

23 M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, 1985. doi:10.1145/2455.2460.

24 C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, Englewood Cliffs, 1985.

25 X. Liu and T.Yu. Canonical solutions to recursive equations and completeness of equational axiomatisations. In I. Konnov and L. Kovacs, editors, Proceedings 31st International Conference on *Concurrency Theory* (CONCUR 2020), Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. To appear.

26 M. Lohrey, P.R. D'Argenio, and H. Hermanns. Axiomatising divergence. *Information and Computation*, 203(2):115–144, 2005. doi:10.1016/j.ic.2005.05.007.

27 R. Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28:439–466, 1984. doi:10.1016/0022-0000(84)90023-0.

28 R. Milner. A complete axiomatisation for observational congruence of finite-state behaviors. *Information and Computation*, 81(2):227–247, 1989. doi:10.1016/0890-5401(89)90070-9.

29 R. Milner. Operational and algebraic semantics of concurrent processes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 19, pages 1201–1242. Elsevier Science Publishers B.V. (North-Holland), 1990. Alternatively see *Communication and Concurrency*, Prentice-Hall, Englewood Cliffs, 1989, of which an earlier version appeared as *A Calculus of Communicating Systems*, LNCS 92, Springer, 1980, doi:10.1007/3-540-10235-3.

30 E.-R. Olderog. Operational Petri net semantics for CCSP. In G. Rozenberg, editor, *Advances in Petri Nets 1987*, volume 266 of LNCS, pages 196–223. Springer, 1987. doi:10.1007/3-540-18086-9_27.

31 E.-R. Olderog and C.A.R. Hoare. Specification-oriented semantics for communicating processes. *Acta Informatica*, 23:9–66, 1986. doi:10.1007/BF00268075.

32 J. Parrow and P. Sjödin. Multiway synchronization verified with coupled simulation. In W.R. Cleaveland, editor, *Proceedings CONCUR 92,* Stony Brook, NY, USA, volume 630 of LNCS, pages 518–533. Springer, 1992. doi:10.1007/BFb0084813.

33 G.M. Reed and A.W. Roscoe. A timed model for communicating sequential processes. *Theoretical Computer Science*, 58:249–261, 1988. doi:10.1016/0304-3975(88)90030-8.

34 F.W. Vaandrager. Expressiveness results for process algebras. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Proceedings REX Workshop on Semantics: Foundations and Applications,* Beekbergen, The Netherlands, June 1992, volume 666 of LNCS, pages 609–638. Springer, 1993. doi:10.1007/3-540-56596-5_49.

35 D.J. Walker. Bisimulation and divergence. *Information and Computation*, 85(2):202–241, 1990. doi:10.1016/0890-5401(90)90048-M.

36 Ernst Zermelo. Untersuchungen über die Grundlagen der Mengenlehre I. *Mathematische Annalen*, 65(2):261–281, 1908. doi:10.1007/bf01449999.

A Proofs

▶ **Proposition 2.** $\leftrightarrows_r$ and $\leftrightarrows_r^X$ for $X \subseteq A$ are equivalence relations.

Proof. $\leftrightarrows_r^X$, $\leftrightarrows_r$ are reflexive, as $\{(P,X,P),(P,P) \mid P \in \mathbb{P} \wedge X \subseteq A\}$ is a strong reactive bisimulation.
$\leftrightarrows_r^X$ and $\leftrightarrows_r$ are symmetric, since strong reactive bisimulations are symmetric by definition.
$\leftrightarrows_r^X$ and $\leftrightarrows_r$ are transitive, for if $\mathscr{R}$ and $\mathscr{S}$ are strong reactive bisimulations, then so is

$$\mathscr{R};\mathscr{S} = \{(P,X,R) \mid (P,X,Q) \in \mathscr{R} \wedge (Q,X,R) \in \mathscr{S}\} \cup \{(P,R) \mid (P,Q) \in \mathscr{R} \wedge (Q,R) \in \mathscr{S}\}.$$ ◀

▶ **Proposition 4.** $P \leftrightarrows_r Q$ iff there exists a gsrb $\mathscr{R}$ with $(P,Q) \in \mathscr{R}$.
Likewise, $P \leftrightarrows_r^X Q$ iff there exists a gsrb $\mathscr{R}$ with $(P,X,Q) \in \mathscr{R}$.

Proof. Clearly, each strong reactive bisimulation satisfies the five clauses of Definition 3 and thus is a gsrb. In the other direction, given a gsrb $\mathscr{B}$, let

$$\begin{aligned}\mathscr{R} := \mathscr{B} &\cup \{(P,X,Q) \mid (P,Q) \in \mathscr{B} \wedge X \subseteq A\} \\ &\cup \{(P,Q),(P,X,Q) \mid (P,Y,Q) \in \mathscr{B} \wedge \mathcal{I}(P) \cap (Y \cup \{\tau\}) = \emptyset\}.\end{aligned}$$

It is straightforward to check that $\mathscr{R}$ satisfies the six clauses of Definition 1. ◀

▶ **Theorem 7.** Let $P,Q \in \mathbb{P}$ and $X \subseteq A$. Then $P \leftrightarrows_r Q \;\Leftrightarrow\; \forall \varphi \in \mathbb{O}.\ (P \models \varphi \Leftrightarrow Q \models \varphi)$
and $P \leftrightarrows_r^X Q \;\Leftrightarrow\; \forall \varphi \in \mathbb{O}.\ (P \models_X \varphi \Leftrightarrow Q \models_X \varphi)$.

Proof. "⇒": I prove by simultaneous structural induction on $\varphi \in \mathbb{O}$ that, for all $P,Q \in \mathbb{P}$ and $X \subseteq A$, $P \leftrightarrows_r Q \wedge P \models \varphi \;\Rightarrow\; Q \models \varphi$ and $P \leftrightarrows_r^X Q \wedge P \models_X \varphi \;\Rightarrow\; Q \models_X \varphi$. The converse implications ($Q \models \varphi \Rightarrow P \models \varphi$ and $Q \models_X \varphi \Rightarrow P \models_X \varphi$) follow by symmetry.

- Let $P \leftrightarrows_r Q \wedge P \models \varphi$.
  - Let $\varphi = \bigwedge_{i \in I} \varphi_i$. Then $P \models \varphi_i$ for all $i \in I$. By induction $Q \models \varphi_i$ for all i, so $q \models \bigwedge_{i \in I} \varphi_i$.
  - Let $\varphi = \neg\psi$. Then $P \not\models \psi$. By induction $Q \not\models \psi$, so $Q \models \neg\psi$.
  - Let $\varphi = \langle\alpha\rangle\psi$ with $\alpha \in A \cup \{\tau\}$. Then $P \xrightarrow{\alpha} P'$ for some P' with $P' \models \psi$. By Definition 3, $Q \xrightarrow{\alpha} Q'$ for some Q' with $P' \leftrightarrows_r Q'$. So by induction $Q' \models \psi$, and thus $Q \models \langle\alpha\rangle\psi$.

- Let $\varphi = \langle X\rangle\psi$ for some $X \subseteq A$. Then $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{t} P'$ for some P' with $P' \models_X \psi$. By Definition 3, $Q \xrightarrow{t} Q'$ for some Q' with $P' \leftrightarrows_r^X Q'$. So by induction $Q' \models_X \psi$. Moreover, $\mathcal{I}(Q) = \mathcal{I}(P)$, as $P \leftrightarrows_r Q$, so $\mathcal{I}(Q) \cap (X \cup \{\tau\}) = \emptyset$. Thus $Q \models \langle X\rangle\psi$.

- Let $P \leftrightarrows_r^X Q \wedge P \models_X \varphi$.
  - Let $\varphi = \bigwedge_{i\in I} \varphi_i$, and $P \models_X \varphi_i$ for all $i \in I$. By induction $Q \models_X \varphi_i$ for all $i \in I$, so $q \models_X \bigwedge_{i\in I} \varphi_i$.
  - Let $\varphi = \neg\psi$, and $P \not\models_X \psi$. By induction $Q \not\models_X \psi$, so $Q \models_X \neg\psi$.
  - Let $\varphi = \langle a\rangle\psi$ with $a \in X$ and $P \xrightarrow{a} P'$ for some P' with $P' \models \psi$. By Definition 1, $Q \xrightarrow{a} Q'$ for some Q' with $P' \leftrightarrows_r Q'$. By induction $Q' \models \psi$, so $Q \models_X \langle a\rangle\psi$.
  - Let $\varphi = \langle\tau\rangle\psi$, and $P \xrightarrow{\tau} P'$ for some P' with $P' \models_X \psi$. By Definition 1, $Q \xrightarrow{\tau} Q'$ for some Q' with $P' \leftrightarrows_r^X Q'$. By induction $Q' \models_X \psi$, so $Q \models_X \langle\tau\rangle\psi$.
  - Let $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \models \varphi$. By the fifth clause of Definition 1, $P \leftrightarrows_r Q$. Hence, by the previous case in this proof, $Q \models \varphi$. Moreover, $\mathcal{I}(Q) \cap (X \cup \{\tau\}) = \mathcal{I}(P) \cap (X \cup \{\tau\})$, since $P \leftrightarrows_r^X Q$. Thus $Q \models_X \varphi$.

"$\Leftarrow$": Write $P \equiv Q$ for $\forall\varphi \in \mathbb{O}.\ (P \models \varphi \Leftrightarrow Q \models \varphi)$, and $P \equiv_X Q$ for $\forall\varphi \in \mathbb{O}.\ (P \models_X \varphi \Leftrightarrow Q \models_X \varphi)$. I show that the family of relations $\equiv$, $\equiv_X$ for $X \subseteq A$ constitutes a gsrb.

- Suppose $P \equiv Q$ and $P \xrightarrow{\alpha} P'$ with $\alpha \in A \cup \{\tau\}$. Let $\mathcal{Q}^\dagger := \{Q^\dagger \in \mathbb{P} \mid Q \xrightarrow{\alpha} Q^\dagger \wedge P' \not\equiv Q^\dagger\}$. For each $Q^\dagger \in \mathcal{Q}^\dagger$, let $\varphi_{Q^\dagger} \in \mathbb{O}$ be a formula such that $P' \models \varphi_{Q^\dagger}$ and $Q^\dagger \not\models \varphi_{Q^\dagger}$. (Such a formula always exists because $\mathbb{O}$ is closed under negation.)
  Define $\varphi := \bigwedge_{Q^\dagger\in\mathcal{Q}^\dagger} \varphi_{Q^\dagger}$. Then $P' \models \varphi$, so $P \models \langle a\rangle\varphi$. Consequently, also $Q \models \langle a\rangle\varphi$. Hence there is a Q' with $Q \xrightarrow{\alpha} Q'$ and $Q' \models \varphi$. Since none of the $Q^\dagger \in \mathcal{Q}^\dagger$ satisfies φ, one obtains $Q' \notin \mathcal{Q}^\dagger$ and thus $P' \equiv Q'$.
- Let $X \subseteq A$ and suppose $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{t} P'$. Let

  $$\mathcal{Q}^\dagger := \{Q^\dagger \in \mathbb{P} \mid Q \xrightarrow{t} Q^\dagger \wedge P' \not\equiv_X Q^\dagger\}.$$

  For each $Q^\dagger \in \mathcal{Q}^\dagger$, let $\varphi_{Q^\dagger} \in \mathbb{O}$ be a formula such that $P' \models_X \varphi_{Q^\dagger}$ and $Q^\dagger \not\models_X \varphi_{Q^\dagger}$. Define $\varphi := \bigwedge_{Q^\dagger\in\mathcal{Q}^\dagger} \varphi_{Q^\dagger}$. Then $P' \models_X \varphi$, so $P \models \langle X\rangle\varphi$. Consequently, also $Q \models \langle X\rangle\varphi$. Hence there is a Q' with $Q \xrightarrow{t} Q'$ and $Q' \models_X \varphi$. Again $Q' \notin \mathcal{Q}^\dagger$ and thus $P' \equiv_X Q'$.
- Suppose $P \equiv_Y Q$ and $P \xrightarrow{\alpha} P'$ with $a \in A$ and either $a \in Y$ or $\mathcal{I}(P) \cap (Y \cup \{\tau\}) = \emptyset$.
  Let $\mathcal{Q}^\dagger := \{Q^\dagger \in \mathbb{P} \mid Q \xrightarrow{\alpha} Q^\dagger \wedge P' \not\equiv Q^\dagger\}$. For each $Q^\dagger \in \mathcal{Q}^\dagger$, let $\varphi_{Q^\dagger} \in \mathbb{O}$ be a formula such that $P' \models \varphi_{Q^\dagger}$ and $Q^\dagger \not\models \varphi_{Q^\dagger}$. Define $\varphi := \bigwedge_{Q^\dagger\in\mathcal{Q}^\dagger} \varphi_{Q^\dagger}$. Then $P' \models \varphi$, so $P \models \langle a\rangle\varphi$, and also $P \models_Y \langle a\rangle\varphi$, using either the third or last clause in the definition of $\models_X$. Hence also $Q \models_Y \langle a\rangle\varphi$. Therefore there is a Q' with $Q \xrightarrow{\alpha} Q'$ and $Q' \models \varphi$, using the third clause of either $\models_X$ or $\models$. Since none of the $Q^\dagger \in \mathcal{Q}^\dagger$ satisfies φ, one obtains $Q' \notin \mathcal{Q}^\dagger$ and thus $P' \equiv Q'$.
- The fourth clause of Definition 3 is obtained exactly like the first, but using $\models_Y$ instead of $\models$.
- Suppose $P \equiv_Y Q$, $P \xrightarrow{t} P'$ and $\mathcal{I}(P) \cap (X \cup Y \cup \{\tau\}) = \emptyset$, with $X \subseteq A$. Let

  $$\mathcal{Q}^\dagger := \{Q^\dagger \in \mathbb{P} \mid Q \xrightarrow{t} Q^\dagger \wedge P' \not\equiv_X Q^\dagger\}.$$

  For each $Q^\dagger \in \mathcal{Q}^\dagger$, let $\varphi_{Q^\dagger} \in \mathbb{O}$ be a formula such that $P' \models_X \varphi_{Q^\dagger}$ and $Q^\dagger \not\models_X \varphi_{Q^\dagger}$. Define $\varphi := \bigwedge_{Q^\dagger\in\mathcal{Q}^\dagger} \varphi_{Q^\dagger}$. Then $P' \models_X \varphi$, so $P \models \langle X\rangle\varphi$, and thus $P \models_Y \langle X\rangle\varphi$. Consequently, also $Q \models_Y \langle X\rangle\varphi$ and therefore $Q \models \langle X\rangle\varphi$. Hence there is a Q' with $Q \xrightarrow{t} Q'$ and $Q' \models_X \varphi$. Again $Q' \notin \mathcal{Q}^\dagger$ and thus $P' \equiv_X Q'$. ◀

▶ **Proposition 9.** $P \leftrightarrows_r Q$ iff there exists a strong time-out bisimulation $\mathscr{B}$ with $P \mathscr{B} Q$.

Proof. Let $\mathscr{R}$ be a gsrb on $\mathbb{P}$. Define $\mathscr{B} \subseteq \mathbb{P} \times \mathbb{P}$ by $P \mathscr{B} Q$ iff either $(P,Q) \in \mathscr{R}$ or $P = \theta_X(P^\dagger)$, $Q = \theta_X(Q^\dagger)$ and $(P^\dagger, X, Q^\dagger) \in \mathscr{R}$. I show that $\mathscr{B}$ is a strong time-out bisimulation.

- Let $P \mathscr{B} Q$ and $P \xrightarrow{a} P'$ with $a \in A$. First suppose $(P,Q) \in \mathscr{R}$. Then, by the first clause of Definition 3, there exists a Q' such that $Q \xrightarrow{a} Q'$ and $(P',Q') \in \mathscr{R}$. So $P' \mathscr{B} Q'$.
  Next suppose $P = \theta_X(P^\dagger)$, $Q = \theta_X(Q^\dagger)$ and $(P^\dagger, X, Q^\dagger) \in \mathscr{R}$. Since $\theta_X(P^\dagger) \xrightarrow{a} P'$ it must be that $P^\dagger \xrightarrow{a} P'$ and either $a \in X$ or $P^\dagger \xrightarrow{\beta}\!\!\!\!\!/\;$ for all $\beta \in X \cup \{\tau\}$. Hence there exists a Q' such that $Q^\dagger \xrightarrow{a} Q'$ and $(P',Q') \in \mathscr{R}$, using the third clause of Definition 3. Recall that $P^\dagger \leftrightarrows_r^X Q^\dagger$ implies $I(P^\dagger) \cap (X \cup \{\tau\}) = I(Q^\dagger) \cap (X \cup \{\tau\})$, and thus either $a \in X$ or $Q^\dagger \xrightarrow{\beta}\!\!\!\!\!/\;$ for all $\beta \in X \cup \{\tau\}$. It follows that $Q = \theta_X(Q^\dagger) \xrightarrow{a} Q'$ and $P' \mathscr{B} Q'$.
- Let $P \mathscr{B} Q$ and $P \xrightarrow{\tau} P'$. First suppose $(P,Q) \in \mathscr{R}$. Then, using the first clause of Definition 3, there is a Q' with $Q \xrightarrow{\tau} Q'$ and $(P',Q') \in \mathscr{R}$. So $P' \mathscr{B} Q'$.
  Next suppose $P = \theta_X(P^\dagger)$, $Q = \theta_X(Q^\dagger)$ and $(P^\dagger, X, Q^\dagger) \in \mathscr{R}$. Since $\theta_X(P^\dagger) \xrightarrow{\tau} P'$, it must be that P' has the form $\theta_X(P^\ddagger)$, and $P^\dagger \xrightarrow{\tau} P^\ddagger$. Thus, by the fourth clause of Definition 3, there is a $Q^\ddagger$ with $Q^\dagger \xrightarrow{\tau} Q^\ddagger$ and $(P^\ddagger, X, Q^\ddagger) \in \mathscr{R}$. Now $Q = \theta_X(Q^\dagger) \xrightarrow{\tau} \theta_X(Q^\ddagger) =: Q'$ and $P' \mathscr{B} Q'$.
- Let $P \mathscr{B} Q$, $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{\mathrm{t}} P'$. First suppose $(P,Q) \in \mathscr{R}$. Then, by the second clause of Definition 3, there is a Q' with $Q \xrightarrow{\mathrm{t}} Q'$ and $(P',X,Q') \in \mathscr{R}$. So $\theta_X(P') \mathscr{B} \theta_X(Q')$.
  Next suppose $P = \theta_Y(P^\dagger)$, $Q = \theta_Y(Q^\dagger)$ and $(P^\dagger, Y, Q^\dagger) \in \mathscr{R}$. Since $\theta_Y(P^\dagger) \xrightarrow{\mathrm{t}} P'$, it must be that $P^\dagger \xrightarrow{\mathrm{t}} P'$ and $P^\dagger \xrightarrow{\beta}\!\!\!\!\!/\;$ for all $\beta \in Y \cup \{\tau\}$. Consequently, $\mathcal{I}(P^\dagger) = \mathcal{I}(P)$ and thus $\mathcal{I}(P^\dagger) \cap (X \cup Y \cup \{\tau\}) = \emptyset$. By the last clause of Definition 3 there is a Q' such that $Q^\dagger \xrightarrow{\mathrm{t}} Q'$ and $(P,X,Q') \in \mathscr{R}$. So $\theta_X(P') \mathscr{B} \theta_X(Q')$. From $(P^\dagger, Y, Q^\dagger) \in \mathscr{R}$ and $\mathcal{I}(P^\dagger) \cap (Y \cup \{\tau\}) = \emptyset$, I infer $\mathcal{I}(Q^\dagger) \cap (Y \cup \{\tau\}) = \emptyset$. So $Q^\dagger \xrightarrow{\beta}\!\!\!\!\!/\;$ for all $\beta \in Y \cup \{\tau\}$. This yields $Q = \theta_Y(Q^\dagger) \xrightarrow{\mathrm{t}} Q'$.

Now let $\mathscr{B}$ be a time-out bisimulation. Define $\mathscr{R} \subseteq \mathbb{P} \times \mathscr{P}(A) \times \mathbb{P}$ by $(P,Q) \in \mathscr{R}$ iff $P \mathscr{B} Q$, and $(P,X,Q) \in \mathscr{R}$ iff $\theta_X(P) \mathscr{B} \theta_X(Q)$. I need to show that $\mathscr{R}$ is a gsrb.

- Suppose $(P,Q) \in \mathscr{R}$ and $P \xrightarrow{\alpha} P'$ with $\alpha \in A \cup \{\tau\}$. Then $P \mathscr{B} Q$, so there is a Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \mathscr{B} Q'$. Hence $(P',Q') \in \mathscr{R}$.
- Suppose $(P,Q) \in \mathscr{R}$, $X \subseteq A$, $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{\mathrm{t}} P'$. Then $P \mathscr{B} Q$, so $\exists Q'$ such that $Q \xrightarrow{\mathrm{t}} Q'$ and $\theta_X(P') \mathscr{B} \theta_X(Q')$. Thus $(P',X,Q') \in \mathscr{R}$.
- Suppose $(P,X,Q) \in \mathscr{R}$ and $P \xrightarrow{a} P'$ with either $a \in X$ or $\mathcal{I}(P) \cap (X \cup \{\tau\}) = \emptyset$. Then $\theta_X(P) \mathscr{B} \theta_X(Q)$. Moreover, $\theta_X(P) \xrightarrow{a} P'$. Hence there is a Q' such that $\theta_X(Q) \xrightarrow{a} Q'$ and $P' \mathscr{B} Q'$. It must be that $Q \xrightarrow{a} Q'$. Moreover, $(P',Q') \in \mathscr{R}$.
- Suppose $(P,X,Q) \in \mathscr{R}$ and $P \xrightarrow{\tau} P'$. Then $\theta_X(P) \mathscr{B} \theta_X(Q)$. Since $P \xrightarrow{\tau} P'$, one has $\theta_X(P) \xrightarrow{\tau} \theta_X(P')$. Hence there is an R such that $\theta_X(Q) \xrightarrow{\tau} R$ and $\theta_X(P') \mathscr{B} R$. The process R must have the form $\theta_X(Q')$ for some Q' with $Q \xrightarrow{\tau} Q'$. It follows that $(P',X,Q') \in \mathscr{R}$.
- Suppose $(P,Y,Q) \in \mathscr{R}$, $X \subseteq A$, $\mathcal{I}(P) \cap (X \cup Y \cup \{\tau\}) = \emptyset$ and $P \xrightarrow{\mathrm{t}} P'$. Then $\theta_Y(P) \mathscr{B} \theta_Y(Q)$ and $\theta_Y(P) \xrightarrow{\mathrm{t}} P'$. Moreover, $\mathcal{I}(\theta_Y(P)) = \mathcal{I}(P)$, so by the second clause of Definition 8 there exists a Q' such that $\theta_Y(Q) \xrightarrow{\mathrm{t}} Q'$ and $\theta_X(P') \mathscr{B} \theta_X(Q')$. So $Q \xrightarrow{\mathrm{t}} Q'$ and $(P',X,Q') \in \mathscr{R}$. ◀

Stratification. Even though negative premises occur in Table 1, the meaning of this transition system specification is well-defined, for instance by the method of *stratification* explained in [21, 13]. Assign inductively to each expression $E \in \mathbb{E}$ an ordinal λ_E that counts the nesting depth of recursive specifications: if $E = \langle x|\mathcal{S}\rangle$ then λ_E is 1 more than the supremum

of the λ_{S_y} for $y \in V_{\mathcal{S}}$; otherwise λ_E is the supremum of $\lambda\langle x|\mathcal{S}\rangle$ for all subterms $\langle x|\mathcal{S}\rangle$ of E. Moreover $\kappa_E \in \mathbb{N}$ is the nesting depth of θ_L^U and ψ_X operators in E that remain after replacing any subterm F of E with $\lambda_F < \lambda_E$ by 0. Now the ordered pair (λ_P, κ_P) constitutes a valid stratification for closed literals $P \xrightarrow{\alpha} P'$. Namely, whenever a transition $P \xrightarrow{\alpha} P'$ depends on a transition $Q \xrightarrow{\beta} Q'$, in the sense that that there is a closed substitution instance $\mathfrak{r}$ of a rule from Table 1 with conclusion $P \xrightarrow{\alpha} P'$, and $Q \xrightarrow{\beta} Q'$ occurring in its premises, then either $\lambda_Q < \lambda_P$, or $\lambda_Q = \lambda_P$ and $\kappa_Q \leq \kappa_P$. Moreover, when $P \xrightarrow{\alpha} P'$ depends on a negative literal $Q \not\xrightarrow{\beta}$, then $\lambda_Q = \lambda_P$ and $\kappa_Q < \kappa_P$.

The above argument hinges on the exclusion of invalid $\mathrm{CCSP}_{\mathrm{t}}^{\theta}$ expressions. The invalid expression $P := \langle x \mid \{x = \theta_{\{a\}}^{\{a\}}(b.0 + \mathcal{R}(x))\}\rangle$ for instance, with $\mathcal{R} = \{(b, a)\}$, does not have a well-defined meaning, since the transition $P \xrightarrow{b} 0$ is derivable iff one has the premise $P \not\xrightarrow{b}$:

$$\dfrac{\dfrac{\dfrac{}{b.0 \xrightarrow{b} 0}}{b.0 + \mathcal{R}(P) \xrightarrow{b} 0} \qquad \dfrac{\dfrac{P \not\xrightarrow{b}}{\mathcal{R}(P) \not\xrightarrow{a}}}{b.0 + \mathcal{R}(P) \not\xrightarrow{a}} \qquad \dfrac{\dfrac{P \not\xrightarrow{\tau}}{\mathcal{R}(P) \not\xrightarrow{\tau}}}{b.0 + \mathcal{R}(P) \not\xrightarrow{\tau}} \text{ (OK)}}{\dfrac{\theta_{\{a\}}^{\{a\}}(b.0 + \mathcal{R}(P)) \xrightarrow{b} 0}{P \xrightarrow{b} 0}}$$

However, the meaning of the valid expression $\langle x \mid \{x = \theta_{\{a\}}^{\{a\}}(\langle y|\{y = b.y\}\rangle) \parallel_{\emptyset} \mathcal{R}(x)\}\rangle$, for instance, is entirely unproblematic.

▶ **Proposition 12.** Each $\mathrm{CCSP}_{\mathrm{t}}^{\theta}$ process is countably branching.

Proof. I show that for each $\mathrm{CCSP}_{\mathrm{t}}^{\theta}$ process Q there are only countably many transitions $Q \xrightarrow{\alpha} R$. Each such transition must be derivable from the rules of Table 1. So it suffices to show that for each Q there are only countably many derivations of transitions $Q \xrightarrow{\alpha} R$.

A derivation of a transition is a well-founded, upwardly branching tree, in which each node models an application of one of the rules of Table 1. Since each of these rules has finitely many positive premises, such a proof tree is finitely branching, and thus finite. Let $d(\pi)$, the *depth* of π, be the length of the longest branch in a derivation π. If π derives a transition $Q \xrightarrow{\alpha} R$, then I call Q the *source* of π.

It suffices to show that for each $n \in \mathbb{N}$ there are only finitely many derivations of depth n with a given source. This I do by induction on n.

In case $Q = f(Q_1, \ldots, Q_k)$, with f an k-ary $\mathrm{CCSP}_{\mathrm{t}}^{\theta}$ operator, a derivation π of depth n is completely determined by the concluding rule from Table 1, deriving a transition $Q \xrightarrow{\beta} R$, the subderivations of π with source Q_i for some of the $i \in \{1, \ldots, k\}$, and the transition label β. The choice of the concluding rule depends on f, and for each f there are at most three choices. The subderivations of π with source Q_i have depth $< n$, so by induction there are only finitely many. When f is not a renaming operator $\mathcal{R}$, there is no further choice for the transition label β, as it is completely determined by the premises of the rule, and thus by the subderivations of those premises. In case $f = \mathcal{R}$, there are finitely many choices for β when faced with a given transition label α contributed by the premise of the rule for renaming. Here I use the requirement of Section 5 that all sets $\{b \mid (a, b) \in \mathcal{R}\}$ are finite. This shows there are only finitely many choices for π.

In case $Q = \langle x|\mathcal{S}\rangle$, the last step in π must be application of the rule for recursion, so π is completely determined by a subderivation π' of a transition with source $\langle \mathcal{S}_x|\mathcal{S}\rangle$. By induction there are only finitely many choices for π', and hence also for π. ◀

▶ **Proposition 13.** Each $\mathrm{CCSP}_{\mathrm{t}}^{\theta}$ process with guarded recursion is finitely branching.

Proof. A trivial structural induction shows that if P is a $\mathrm{CCSP}_{\mathrm{t}}^{\theta}$ process with guarded recursion and Q is reachable from P, then also Q has with guarded recursion. Hence it suffices to show that for each $\mathrm{CCSP}_{\mathrm{t}}^{\theta}$ process Q with guarded recursion there are only finitely many derivations with source Q.

Let $\xrightarrow{u}$ be the smallest binary relation on $\mathbb{P}$ such that (i) $f(P_1,\ldots,P_k) \xrightarrow{u} P_i$ for each k-ary $\mathrm{CCSP}_{\mathrm{t}}^{\theta}$ operator f except action prefixing, and each $i \in \{1,\ldots,k\}$, and (ii) $\langle x|\mathcal{S}\rangle \xrightarrow{u} \langle \mathcal{S}_x|\mathcal{S}\rangle$. This relation is finitely branching. Moreover, on processes with guarded recursion, $\xrightarrow{u}$ has no forward infinite chains $P_0 \xrightarrow{u} P_1 \xrightarrow{u} \ldots$. In fact, this could have been used as an alternative definition of guarded recursion. Let, for any process Q with guarded recursion, $e(Q)$ be the length of the longest forward chain $Q \xrightarrow{u} P_1 \xrightarrow{u} \ldots \xrightarrow{u} P_{e(Q)}$. I show with induction on $e(Q)$ that there are only finitely many derivations with source Q. In fact, this proceeds exactly as in the previous proof. ◀

▶ **Proposition 14.** Each finitely branching processes in an LTS can be denoted by a $\mathrm{CCSP}_{\mathrm{t}}$ expression with guarded recursion. Here I only need the operations inaction, action prefixing, choice and recursion.

Proof. Let P be a finitely branching process in an LTS $(\mathbb{P}', Act, \rightarrow)$. Let

$$V_{\mathcal{S}} := \{x_Q \mid Q \in \mathbb{P}' \text{ is reachable from } P\} \subseteq Var.$$

For each Q reachable from P, let $next(Q)$ be the finite set of pairs $(\alpha, R) \in Act \times \mathbb{P}'$ such that there is a transition $Q \xrightarrow{\alpha} R$. Let $\mathcal{S} := \{x_Q = \sum_{(\alpha,R)\in next(Q)} \alpha.x_R \mid x_Q \in V_{\mathcal{S}}\}$. Here the finite choice operator $\sum_{i\in I} \alpha_i.P_i$ can easily be expressed in terms of inaction, action prefixing and choice. Now the $\mathrm{CCSP}_{\mathrm{t}}$ process $\langle x_P|\mathcal{S}\rangle$ denotes P. ◀

In fact, $\langle x_P|\mathcal{S}\rangle \leftrightarrows P$, where $\leftrightarrows$ denotes strong bisimilarity [29], formally defined in the next section.

▶ **Proposition 15.** Each countably branching processes in an LTS can be denoted by a $\mathrm{CCSP}_{\mathrm{t}}$ expression. Again I only need the $\mathrm{CCSP}_{\mathrm{t}}$ operations 0, $\alpha._-$, + and recursion.

Proof. The proof is the same as the previous one, except that $next(Q)$ now is a countable set, rather than a finite one, and consequently I need a countable choice operator $\sum_{i\in\mathbb{N}} \alpha_i.P_i$. The latter can be expressed in $\mathrm{CCSP}_{\mathrm{t}}$ with unguarded recursion by

$$\sum_{i\in\mathbb{N}} \alpha_i.P_i := \langle z_o|\{z_i = \alpha_i.P_i + z_{i+1} \mid i \in \mathbb{N}\}\rangle.$$ ◀

▶ **Lemma 32.** For each $\mathrm{CCSP}_{\mathrm{t}}^{\theta}$ process P there exists a $\mathrm{CCSP}_{\mathrm{t}}^{\theta}$ process Q only built using inaction, action prefixing, choice and recursion, such that $P \leftrightarrows Q$.

Proof. Immediately from Propositions 12 and 15. ◀

▶ **Theorem 20.** Strong bisimilarity is a full congruence for $\mathrm{CCSP}_{\mathrm{t}}^{\theta}$.

Proof. The structural operational rules for $\mathrm{CCSP}_{\mathrm{t}}$ fit the *tyft/tyxt format with recursion* of [14]. By [14, Theorem 3] this implies that $\leftrightarrows$ is a full congruence for $\mathrm{CCSP}_{\mathrm{t}}$. (In fact, when omitting the recursion construct, the operational rules for $\mathrm{CCSP}_{\mathrm{t}}$ fit the *tyft/tyxt format* of [22], and by the main theorem of [22], $\leftrightarrows$ is a congruence *for the operators of* $\mathrm{CCSP}_{\mathrm{t}}$, that is, it satisfies (1) in Definition 16. The work of [14] extends this result of [22] with recursion.)

The structural operational rules for all of $\mathrm{CCSP}^\theta_\mathrm{t}$ fit the *ntyft/ntyxt format with recursion* of [14]. By [14, Theorem 2] this implies that $\leftrightarroweq$ is a lean congruence for $\mathrm{CCSP}^\theta_\mathrm{t}$. (In fact, when omitting the recursion construct, the operational rules for $\mathrm{CCSP}^\theta_\mathrm{t}$ fit the *ntyft/ntyxt format* of [21], and by the main theorem of [21], $\leftrightarroweq$ is a congruence for the operators of $\mathrm{CCSP}^\theta_\mathrm{t}$. The work of [14] extends this result of [21] with recursion.)

To verify (2) for the whole language $\mathrm{CCSP}^\theta_\mathrm{t}$, let $\mathcal{S}$ and $\mathcal{S}'$ be recursive specifications with $x \in V_\mathcal{S} = V_{\mathcal{S}'}$, such that $\langle x|\mathcal{S}\rangle, \langle x|\mathcal{S}'\rangle \in \mathbb{P}$ and $\mathcal{S}_y \leftrightarroweq \mathcal{S}'_y$ for all $y \in V_\mathcal{S}$. Let $\{P_i \mid i \in I\}$ be the collection of processes of the form $\theta^U_L(Q)$ or $\psi_X(Q)$, for some L, U, X, that occur as a closed subexpression of $\mathcal{S}_y$ or $\mathcal{S}'_y$ for one of the $y \in V_\mathcal{S}$, not counting strict subexpressions of a closed subexpression R of $\mathcal{S}_y$ or $\mathcal{S}'_y$ that is itself of the form $\theta^U_L(Q)$ or $\psi_X(Q)$. Pick a fresh variable $z_i \notin V_\mathcal{S}$ for each $i \in I$, and let, for $y \in Y$, $\widehat{\mathcal{S}}_y$ be the result of replacing each occurrence of P_i in $\mathcal{S}_y$ by z_i. Then $\widehat{\mathcal{S}}_y$ does not contain the operators $\theta^U_L(Q)$ or $\psi_X(Q)$. In deriving this conclusion it is essential that $\langle x|\mathcal{S}\rangle$ is a valid expression, for this implies that the term $\mathcal{S}_y \in \mathbb{E}$, which may contain free occurrences of the variables $y \in V_\mathcal{S}$, does not have a subterm of the form $\theta^U_L(F)$ or $\psi_X(F)$ that contains free occurrences of these variables. Let $\widehat{\mathcal{S}} := \{y = \widehat{\mathcal{S}}_y \mid y \in V_\mathcal{S}\}$; it is a recursive specification in the language CCSP_t. The recursive specification $\widehat{\mathcal{S}}'$ is defined in the same way.

For each $i \in I$ there is, by Lemma 32, a process Q_i in the language CCSP_t such that $P_i \leftrightarroweq Q_i$. Now let $\rho, \eta : \{z_i \mid i \in I\} \to \mathbb{P}$ be the substitutions defined by $\rho(z_i) = P_i$ and $\eta(z_i) = Q_i$ for all $i \in I$. Then $\rho \leftrightarroweq \eta$. Since $\leftrightarroweq$ is a lean congruence for $\mathrm{CCSP}^\theta_\mathrm{t}$, one has $\langle x|\widehat{\mathcal{S}}\rangle[\rho] \leftrightarroweq \langle x|\widehat{\mathcal{S}}\rangle[\eta]$ and likewise $\langle x|\widehat{\mathcal{S}}'\rangle[\rho] \leftrightarroweq \langle x|\widehat{\mathcal{S}}'\rangle[\eta]$. For the same reason one has $\widehat{\mathcal{S}}_y[\eta] \leftrightarroweq \widehat{\mathcal{S}}_y[\rho] = \mathcal{S}_y \leftrightarroweq \mathcal{S}'_y \leftrightarroweq \widehat{\mathcal{S}}'_y[\rho] \leftrightarroweq \widehat{\mathcal{S}}'_y[\eta]$ for all $y \in V_\mathcal{S}$. Since $\widehat{\mathcal{S}}[\eta]$ and $\widehat{\mathcal{S}}'[\eta]$ are recursive specifications over CCSP_t, $\langle x|\widehat{\mathcal{S}}[\eta]\rangle \leftrightarroweq \langle x|\widehat{\mathcal{S}}'[\eta]\rangle$. Hence

$$\langle x|\mathcal{S}\rangle = \langle x|\widehat{\mathcal{S}}[\rho]\rangle = \langle x|\widehat{\mathcal{S}}\rangle[\rho] \leftrightarroweq \langle x|\widehat{\mathcal{S}}\rangle[\eta] = \langle x|\widehat{\mathcal{S}}[\eta]\rangle \leftrightarroweq \langle x|\widehat{\mathcal{S}}'[\eta]\rangle \leftrightarroweq \langle x|\widehat{\mathcal{S}}'[\rho]\rangle = \langle x|\mathcal{S}'\rangle. \qquad \blacktriangleleft$$

How Reversibility Can Solve Traditional Questions: The Example of Hereditary History-Preserving Bisimulation

Clément Aubert
School of Computer and Cyber Sciences, Augusta University, GA, USA
http://spots.augusta.edu/caubert/
caubert@augusta.edu

Ioana Cristescu
Tarides, Paris, France
ioana@tarides.com

Abstract

Reversible computation opens up the possibility of overcoming some of the hardware's current physical limitations. It also offers theoretical insights, as it enriches multiple paradigms and models of computation, and sometimes retrospectively enlightens them. Concurrent reversible computation, for instance, offered interesting extensions to the Calculus of Communicating Systems, but was still lacking a natural and pertinent bisimulation to study processes equivalences. Our paper formulates an equivalence exploiting the two aspects of reversibility: backward moves and memory mechanisms. This bisimulation captures classical equivalences relations for denotational models of concurrency (history- and hereditary history-preserving bisimulation, (H)HPB), that were up to now only partially characterized by process algebras. This result gives an insight on the expressiveness of reversibility, as both backward moves and a memory mechanism – providing "backward determinism" – are needed to capture HHPB.

2012 ACM Subject Classification Theory of computation → Program semantics; Computing methodologies → Concurrent programming languages

Keywords and phrases Formal semantics, Process algebras and calculi, Distributed and reversible computation, Configuration structures, Reversible CCS, Bisimulation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.7

Related Version A complete but preliminary research report containing all the proofs can be found at https://hal.archives-ouvertes.fr/hal-02568250.

Acknowledgements The authors would like to thank John Natale for correcting the definition of postfixing and the reviewers of an earlier version of this work and of this version for their precious comments that greatly improved the paper. We were unfortunately not able to accomodate all of their suggestions, but have tried to reflect their comments in the body of the paper.

1 Introduction

The Benefits of Reversible Computation. Future progresses in computing may heavily rely on reversibility [17]. The foreseeable limitations of conventional semiconductor technology, Landauer's principle [22] – promising low-energy consumption for reversible computers – and quantum computing [26] – intrinsically reversible and now within reach [1] – motivated a colossal push toward a better understanding of reversible computation. Those efforts have given birth to new paradigms [16], richer models of computation (e.g. for automata [20], Petri nets [14, 27], Turing machines [3]) and richer semantics, sometimes for preexisting calculi like the Calculus of Communicating Systems (CCS) [11, 28], the π-calculus [9, 10] and the higher-order π-calculus [25]. Those new perspectives sometimes additionally give in retrospect a better understanding of "traditional" (i.e., irreversible) computation, and our contribution illustrates this latter aspect.

Summary. In brief terms, we offer a solution to an open problem on classes of equivalence of (non-reversible) concurrent processes thanks to reversibility. *Hereditary history-preserving bisimulation* (HHPB) is considered "the gold standard" for establishing equivalence classes on "true" models of concurrency [21, 36]. However, no relation expressed in syntactical terms (e.g. on CCS) was known to capture it, despite intensive efforts: previous results [2, 29] characterized HHPB on limited classes of processes, that excluded CCS terms as simple as $a.a$, $a + a$, $a \mid a$ or containing similar patterns. We prove in this paper a somewhat expected result, namely that adding mechanisms to reverse the computation and keep track of the past enables syntactical characterizations of HHPB. This result uses natural reformulations of canonical CCS bisimulations, provides a "meaningful" bisimulation for reversible calculi, and furthermore validates the mechanism we use to reverse the computation in a concurrent set-up. It also connects with previous semantics of reversible processes [2, 18] and supports categorical treatment.

Reversing Concurrent Computation. Reversible systems can backtrack and return to a previous state. Implementing reversibility often requires a mechanism to record the history of the execution. Ideally, this history should be *complete*, so that every forward step can be backtracked, and *minimal*, so that only the relevant information is saved. Concurrent programming languages have additional requirements: the history should be *distributed*, to avoid centralization, and should prevent steps that required a synchronization with other parts of the program to backtrack without undoing this synchronization. To fulfill those requirements, Reversible CCS (RCCS) [11, 12] uses *memories* attached to the threads of a process[1], and CCS with Communication Keys (CCSK) [30] "marks" each occurrence of an action. The two calculi are equi-expressive [24] and are conservative extensions over CCS, and in this paper we will use RCCS, to benefit from its previous denotational semantics [2]. Reversible calculi are also *backward deterministic*, a term introduced [6, p. 15] to denote the fact that only one piece of information – in RCCS, the identifier in a memory – is enough to undo a particular action. It should be noted that using only the labels is not enough, as it can be the case that multiple events with the same label have taken place and could be undone.

Equivalences for Denotational Models of Concurrency. A theory of concurrent computation, be it reversible or irreversible, relies not only on a syntax and a model, but also on "meaningful" behavioral equivalences. This paper defines new bisimulations on RCCS, and proves their relevance by connecting them to bisimulations on configuration structures [40], a classical denotational model for concurrency. In configuration structures, an *event* represents an execution step, and a *configuration* – a set of events that occurred – represents a state. A forward transition is then represented as moving from a configuration to one of its supersets. Backward transitions have a "built-in" representation: it suffices to move from a configuration to one of its subsets. Among the behavioral equivalences on configuration structures, some of them, like *history-* and *hereditary history-preserving bisimulation* (HPB and HHPB) [6, 32, 34, 35], use that "built-in" notion of reversibility. HPB and HHPB are usually regarded as the "canonical" [29, p. 94] or "strongest desirable true" [31, p. 2] concurrency equivalences because they preserve causality, branching, their interplay, and are

[1] In this system, the distribution of the memory is given precedence over the minimality: while there is some redundancy in the memories, they are maximally distributed, and hence every thread carries its own complete history.

the coarser (or finer, for HHPB) reasonable equivalences with these properties [37, p. 309]. HHPB also naturally equals path bisimulations [21], an elegant notion of equivalence relying on category theory, and can be captured concisely using event identifier logic [31]. However, no relation on operational semantics, e.g. on the labeled transition systems (LTS) of CCS, RCCS, or CCSK, was known to capture it on all processes.

Encoding Reversible Processes in Configuration Structures. There have been multiple attempts [2, 30] to transfer equivalences defined on denotational models, by construction suited for reversibility, into reversible process algebras. Showing that an equivalence on configuration structures corresponds to one on processes starts by defining an encoding of the latter in the former. Encodings, for RCCS [2] and CCSK [18] alike, generally consider only *reachable* reversible processes, and map them to one particular configuration in the encoding of its *origin*, the CCS "memory-less" process to which it can backtrack. We come back to the relation between this encoding and our current work in the conclusion.

Contribution. This paper improves on previous results [2, 29] by defining relations on syntactical terms that correspond to HHPB on *all* processes, and hence are proven to be "the right" bisimulation to study reversible computation[2]. This result relies on encoding the processes' memories into *identified configuration structures*, an extension to configuration structures that constitutes our second contribution.

RCCS is exploited as a syntactic tool to decide HHPB for CCS processes, by endowing them with

1. backtracking capabilities and
2. a memory mechanism.

This gives a precious insight on the expressiveness of reversibility, as we show that having only one of those tools is not enough to define relations that capture HHPB. The memories attached to a process are no longer only a syntactic layer to implement reversibility, but become essential to define and capture equivalences, thanks to the backward determinism they provide.

Recursion is not treated in this paper: its treatment amounts to unfolding processes and structures up to a certain level, and it strongly suggests that there are not much insight to gain from its development, that we reserve as a technical appendix for future work.

Related Work. The correspondences between HHPB and back-and-forth bisimulations for restricted classes of processes [2, 29] inspired some of the work presented here. This correspondence has been studied in the denotational world, on structures without auto-concurrency [6].

The insight that backward determinism is essential to capture HHPB is also used in the event identifier logic [31] and in one if its sub-systems [5], that both characterize HHPB without restricting the structures considered. As in our case, the authors exploit identifiers to eliminate constraints due to label "duplication". Our work is similar, taking place in the operational world instead of the logical one. Note that we do not introduce extra syntax constructions on the LTS, but simply use the ones provided by RCCS, that we use "as is".

[2] Of course, this claim is open for discussion [23, Conclusion], nevertheless HHPB is to the best of our knowledge the strongest form of bisimulation one can obtain on truly concurrent systems, which makes it, at the very least, a point of comparison when studying other relations.

In the operational semantics, the previous attempts to characterize HHPB wrongly focused on the backward capabilities of processes and considered the memory mechanisms only as a tool to achieve it. As a result, those bisimulations could not decide that the encoding of $(a.a) \mid b$ and $a \mid a \mid b$ were not HHPB. Instead, their characterizations of HHPB were applicable only on "non-repeating" [29] or "singly-labeled" [2] processes. By integrating the memory mechanism, the equivalence relation we introduce and consider can correctly determine that these two processes are not HHPB, as we detail in Example 27.

Finally, our approach shares similarity with causal trees – in the sense that only *part of the execution*, its "past", is encoded in a denotational representation – which were used to characterize HPB for CCS terms [13]. Capturing (H)HPB with novel techniques can also impact model-checking and decidability issues [5], but we leave this as future work.

2 Denotational and Operational Models for the Reversible CCS

We write $\subseteq$ for the set inclusion, $\mathcal{P}$ for the power set, $\setminus$ for the set difference, $f : A \to B$ (resp. $f : A \rightharpoonup B$) for the (resp. partial) function from A to B, $f\restriction_C$ for the restriction of f to $C \subseteq A$, and $f \cup \{a \mapsto b\}$ for the function defined as f on A that additionally maps $a \notin A$ to b.

2.1 Identified Configuration Structures

Labeled configuration structures [38, 39] are a classical non-interleaving model of concurrent computation – also known as "stable configuration structures" [37, Definition 5.5] or "completed stable families" [41, Section 3.2] – , that we enrich here with identifiers. We then show that they support categorical understanding and the usual operations just as well as their "un-identified" variations.

▶ **Definition 1** ((Identified) Configuration structure). *A* configuration structure $\mathcal{C}$ *is a tuple* (E, C, L, ℓ) *where* E *is a set of events,* L *is a set of* labels, $\ell : E \to L$ *is a labeling function and* $C \subseteq \mathcal{P}(E)$ *is a set of subsets satisfying:*

$$\forall x \in C, \forall e \in x, \exists z \in C \text{ finite}, e \in z, z \subseteq x \qquad \text{(Finiteness)}$$

$$\forall x \in C, \forall d, e \in x, d \neq e \Rightarrow \exists z \in C, z \subseteq x, d \in z \iff e \notin z \qquad \text{(Coincidence Freeness)}$$

$$\forall X \subseteq C, \forall x, y \in X, \exists z \in C \text{ finite}, x \cup y \subseteq z \Rightarrow \bigcup X \in C \qquad \text{(Finite Completeness)}$$

$$\forall x, y \in C, x \cup y \in C \Rightarrow x \cap y \in C \qquad \text{(Stability)}$$

If $\mathcal{C}$ *also has a set of* identifiers I *and an* identifying function $m : E \to I$ *satisfying:*

$$\forall x \in C, \forall e_1, e_2 \in x, m(e_1) \neq m(e_2) \qquad \text{(Collision Freeness)}$$

then we write $\mathcal{C} \oplus m$*, and say that* $\mathcal{I} = (E, C, L, \ell, I, m)$ *is an* identified configuration structure*, or* $\mathcal{I}$-structure*, and call* $\mathcal{C}$ the underlying configuration structure *of* $\mathcal{I}$*.*

We denote with $\mathbf{0}$ *both the* $\mathcal{I}$*-structure and its underlying configuration structure with* $C = \{\emptyset\}$*, and, for* $x, y \in C$*, we write* $x \xrightarrow{e} y$ *and* $y \overset{e}{\rightsquigarrow} x$ *if* $x = y \cup \{e\}$ *and* $e \notin x$*.*

We omit the identifiers when representing $\mathcal{I}$-structures and write the label for the event (with a subscript if multiple events shares a label).

▶ **Definition 2** (Causality, concurrency, and maximality). *For all* $\mathcal{I}$*,* $x \in C$ *and* $d, e \in x$*, the* causality relation on x *is given by* $d <_x e$ *iff* $d \leqslant_x e$ *and* $d \neq e$*, where* $d \leqslant_x e$ *iff for all* $y \in C$ *with* $y \subseteq x$*, we have* $e \in y \Rightarrow d \in y$*. The* concurrency relation on x *is given by* $d \operatorname{co}_x e$ *iff* $\neg(d <_x e \vee e <_x d)$*. Finally,* x *is* a maximal configuration in $\mathcal{I}$ *if* $\forall y \in C$*,* $x = y$ *or* $x \not\subseteq y$*.*

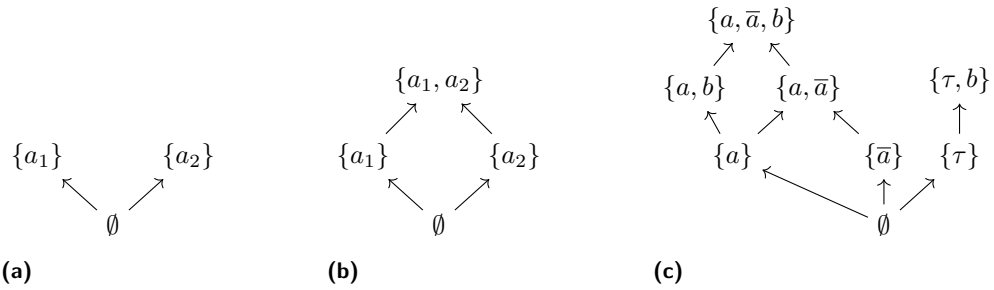


Figure 1 Examples of $\mathcal{I}$-structures.

▶ **Example 3.** Consider Fig. 1, where we let the events have distinct arbitrary identifiers: if two events with complement names as labels can happen at the same time (Fig. 1c), then they are "merged" into a single event labeled with τ, as is usual in CCS (Sect. 2.2). In Fig. 1c, $a <_{\{a,b\}} b$, $a <_{\{a,\overline{a},b\}} b$ and $\tau <_{\{\tau,b\}} b$; and in Fig. 1b, $a_1 \operatorname{co}_{\{a_1,a_2\}} a_2$. An $\mathcal{I}$-structure can have one (Fig. 1b) or multiple (Fig. 1a and 1c) maximal configurations.

Categorical point of view. We remind in Appendix A that configuration structures and "structure-preserving" functions form a category. We also prove that a similar category can be defined with $\mathcal{I}$-structures as objects and define a forgetful functor that returns the underlying configuration structure. This development supports the interest and validity of studying $\mathcal{I}$-structures, but can be omitted, except for the notion of morphisms:

▶ **Definition 4** (Morphism of $\mathcal{I}$-structure). *A morphism $f = (f_E, f_L, f_C, f_m)$ between $\mathcal{I}_1$ and $\mathcal{I}_2$ is given by $f_E : E_1 \to E_2$ such that $\ell_2(f_E(e)) = f_L(\ell_1(e))$, for $f_L : L_1 \to L_2$; $f_C : C_1 \to C_2$ defined as $f_C(x) = \{f_E(e) \mid e \in x\}$, and $f_m : I_1 \to I_2$ such that $f_m(m_1(e)) = m_2(f_E(e))$. We often write f for all the components, and write $\mathcal{I}_1 \cong \mathcal{I}_2$ if f is an isomorphism.*

Operations on $\mathcal{I}$-configurations. Operations on $\mathcal{I}$-structures are conservative extensions over their counterparts on configuration structures – forgetting about event identifiers gives back the "un-identified" definition [39] – , except for the parallel composition. The intuition here is that configuration structures encode CCS processes, and $\mathcal{I}$-structures encode memories of RCCS processes, where parallel composition has a different meaning. Examples of those operations will be given in Sect. 2.4, after introducing the calculi in Sect. 2.2 and the encodings in Sect. 2.3. Sect. C.1 gives intuitions on the correctness of those operations.

Given two sets A, B, and a symbol $\star \notin A \cup B$ denoting *undefined*, we write $C^\star = C \cup \{\star\}$ if $\star \notin C$ and define the partial product [39, Appendix A] of A and B to be

$$A \times_\star B = \{(a, \star) \mid a \in A\} \cup \{(\star, b) \mid b \in B\} \cup \{(a, b) \mid a \in A, b \in B\}$$

and the two projections to be $\pi_1 : A \times_\star B \to A^\star$ and $\pi_2 : A \times_\star B \to B^\star$.

▶ **Definition 5** (Operations on $\mathcal{I}$-structures). *Given $\mathcal{I}_i = (E_i, C_i, L_i, \ell_i, I_i, m_i)$, for $i = 1, 2$,*

The relabeling *of $\mathcal{I}_1$ along $\ell' : E_1 \to L$ is $\mathcal{I}_1[\ell'/\ell_1] = (E_1, C_1, L, \ell', I_1, m_1)$.*

The reidentifiying *of $\mathcal{I}_1$ along $m : E_1 \to I$ is $\mathcal{I}_1[m/m_1] = (E_1, C_1, L_1, \ell_1, I, m)$, provided $\mathcal{I}_1[m/m_1]$ respects the Collision Freeness condition.*

The restriction of a set of events *$A \subseteq E_1$ in $\mathcal{I}_1$ is $\mathcal{I}_1 \restriction_A = (E, C, L, \ell_1 \restriction_E, I, m_1 \restriction_E)$, where $E = E_1 \backslash A$, $C = \{x \mid x \in C_1$ and $x \cap A = \emptyset\}$, $L = \{a \mid \exists e \in E_1 \backslash A, \ell_1(e) = a\}$ and $I = \{i \mid \exists e \in E_1 \backslash A, m_1(e) = i\}$.*

The restriction of a set of labels $L \subseteq L_1$ *in* $\mathcal{I}_1$ *is* $\mathcal{I}_1{\restriction_L} = \mathcal{I}_1{\restriction_{E_1^L}}$ *where* $E_1^L = \{e \in E_1 \mid \ell_1(e) \in L\}$. *We write* $\mathcal{I}_1{\restriction_a}$, *when the restricting set of labels* L *is the singleton* $\{a\}$.

The prefixing *of* $\mathcal{I}_1$ *by the label* a *is* $a.\mathcal{I}_1 = (E, C, L, \ell, I, m)$ *where*

- $E = E_1 \cup \{e\}$, *for* $e \notin E_1$,
- $C = \{x \mid x = \emptyset$ *or* $\exists x' \in C_1, x = x' \cup e\}$,
- $L = L_1 \cup \{a\}$,
- $\ell = \ell_1 \cup \{e \mapsto a\}$,
- $I = I_1 \cup \{i\}$, *for* $i \notin I_1$
- $m = m_1 \cup \{e \mapsto i\}$.

The postfixing *of* (a, i) *to* $\mathcal{I}_1$ *is defined if* $i \notin I_1$ *as* $\mathcal{I}_1 ::(a, i) = (E, C, L, \ell, I, m)$ *where everything is as in* $a.\mathcal{I}_1$, *except that* $C = C_1 \cup \{x \cup \{e\} \mid x \in C_1$ *is maximal and finite*$\}$.

The nondeterministic choice *of* $\mathcal{I}_1$ *and* $\mathcal{I}_2$ *is* $\mathcal{I}_1 + \mathcal{I}_2 = (E, C, L, \ell, I, m)$, *where*

- $E = \{\{1\} \times E_1\} \cup \{\{2\} \times E_2\}$ *with* $\pi_1 : E \to \{1, 2\}$ *and* $\pi_2 : E \to E_1 \cup E_2$,
- $C = \{\{i\} \times x \mid x \in C_i\}$,
- $L = L_1 \cup L_2$,
- $\ell(e) = \ell_i(\pi_2(e))$ *for* $\pi_1(e) = i$,
- $I = I_1 \cup I_2$,
- $m(e) = m_i(\pi_2(e))$ *for* $\pi_1(e) = i$.

The product *of* $\mathcal{I}_1$ *and* $\mathcal{I}_2$ *is* $\mathcal{I}_1 \times \mathcal{I}_2 = (E, C, L, \ell, I, m)$, *where:*

- $E = E_1 \times_\star E_2$, *with* $\pi_i : E \to E_i^\star$ *the projections of the partial product,*
- *For* $i \in \{1, 2\}$, *define the projections* $\gamma_i : \mathcal{I}_1 \times \mathcal{I}_2 \to \mathcal{I}_i$ *and the configurations* $x \in C$*:*

$$\begin{array}{l}
\forall e \in E, \gamma_i(e) = \pi_i(e), \gamma_i(\ell_i(e)) = \ell_i(\pi_i(e)), \gamma_i(m_i(e)) = m_i(\pi_i(e)) \\
\gamma_i(x) \in C_i, \text{ with } \gamma_i(x) = \{e_i \mid \pi_i(e) = e_i \neq \star \text{ and } e \in x\} \\
\forall e, e' \in x, \pi_1(e) = \pi_1(e') \neq \star \text{ or } \pi_2(e) = \pi_2(e') \neq \star \Rightarrow e = e' \\
\forall e \in x, \exists z \subseteq x \text{ finite}, \gamma_i(z) \in C_i, e \in z \\
\forall e, e' \in x, e \neq e' \Rightarrow \exists z \subseteq x, \gamma_i(z) \in C_i, e \in z \iff e' \notin z
\end{array}$$

- $\ell : E \to L = L_1 \times_\star L_2$ *is* $\ell(e) = \begin{cases} (\ell_1(e_1), \star) & \text{if } \pi_2(e) = \star \\ (\star, \ell_2(e_2)) & \text{if } \pi_1(e) = \star \\ (\ell_1(e_1), \ell_2(e_2)) & \text{otherwise} \end{cases}$
- $m : E \to I = I_1 \times_\star I_2$ *is* $m(e) = \begin{cases} (m_1(\pi_1(e)), \star) & \text{if } \pi_2(e) = \star \\ (\star, m_2(\pi_2(e))) & \text{if } \pi_1(e) = \star \\ (m_1(\pi_1(e)), m_2(\pi_2(e))) & \text{otherwise} \end{cases}$

We now recall the definition of parallel composition for configuration structures, and detail the definition for $\mathcal{I}$-structures. Parallel composition consists of a combination of product, relabelling, reidentifiying for the $\mathcal{I}$-structures, and restriction. For the relabelling operation, we use a *synchronization algebra* [42] $(S, \star, \bot)$ consisting of a commutative and associative operation $\bullet$ on a set of labels $S \cup \{\star, \bot\}$, where $\{\star, \bot\} \notin S$ and such that $a \bullet \star = a$ (i.e. $\star$ is the identity element) and $a \bullet \bot = \bot$ (i.e. $\bot$ is the zero element), for all $a \in S$. To avoid repetition, below we assume given $(S, \star, \bot)$, such that $S \subseteq L_1 \cup L_2$. We give examples of synchronization algebras in Sect. 2.3.

▶ **Definition 6** (Parallel composition of configuration structures)**.** The parallel composition *of* $\mathcal{C}_1$ *and* $\mathcal{C}_2$ *is* $\mathcal{C}_1 \mid_S \mathcal{C}_2 = \big((\mathcal{C}_1 \times \mathcal{C}_2)[\ell'/\ell]\big){\restriction_\bot}$ *where* $\ell : E_1 \times_\star E_2 \to L_1 \cup L_2$ *is the labeling function from the product, and* $\ell' : E_1 \times_\star E_2 \to L_1 \cup L_2 \cup \{\bot\}$ *is defined as* $\ell'(e) = \ell_1(e_1) \bullet \ell_2(e_2)$.

▶ **Definition 7** (Parallel composition of $\mathcal{I}$-structures). The parallel composition *of* $\mathcal{I}_1$ *and* $\mathcal{I}_2$, *is* $\mathcal{I}_1 \mid_S \mathcal{I}_2 = (\mathcal{I}_3[m'/m_3][\ell'/\ell_3])\restriction_\perp$ *where* $\mathcal{I}_3 = (E_3, C_3, L_3, \ell_3, I_3, m_3)$ *is* $\mathcal{I}_1 \times \mathcal{I}_2$, *and*

- $m' : E_3 \to I_1 \cup I_2 \cup \{\perp_k \mid k \in I_1 \times_\star I_2\}$ *is defined as follows, for* $i \neq \star$*:*

$$m'(e) = \begin{cases} i & \text{if } m_3(e) = (i,i) & \text{(Sync. or Fork)} \\ i & \text{if } m_3(e) = (i,\star) \wedge \forall e_2 \in E_2, m_2(e_2) \neq i & \text{(Extra } \star\text{. 1)} \\ i & \text{if } m_3(e) = (\star,i) \wedge \forall e_1 \in E_1, m_1(e_1) \neq i & \text{(Extra } \star\text{. 2)} \\ \perp_k & \text{otherwise, with } m_3(e) = k & \text{(Error)} \end{cases}$$

- $\ell' : E_3 \to L_1 \cup L_2 \cup \{\perp\}$ *maps* e *to* $\perp$ *if* $m'(e) = \perp_k$, *and to* $\ell_1(e_1) \bullet \ell_2(e_2)$ *otherwise.*

Parallel composition removes from the product the pairs of events that represent "non-realizable" interactions: for configuration structures, pairs of events that do not represent *possible and future synchronizations*; for $\mathcal{I}$-structures, pairs of events that do not represent *past synchronizations or forks*. Definitions 11 and 12 will detail how those operations are used to encode a process or a memory, and both types of parallel compositions will be illustrated in Examples 14 and 15.

2.2 Concurrent Communicating Calculi

Let $\mathsf{I} = \mathbb{N}$ be a set of *identifiers* and i, j range over it. Let $\mathsf{N} = \{a, b, c, \dots\}$ be a set of *names* and $\overline{\mathsf{N}} = \{\overline{a}, \overline{b}, \overline{c}, \dots\}$ its set of *co-names*. We let $\mathsf{N} \cup \overline{\mathsf{N}} \cup \{\tau\}$ be the set of *labels*, and use α (resp. λ, μ, ν) to range over them (resp. over $\mathsf{N} \cup \overline{\mathsf{N}}$). The *complement* of a name is given by a bijection $\overline{\cdot} : \mathsf{N} \to \overline{\mathsf{N}}$, whose inverse is also written $\overline{\cdot}$, and that we extend to τ, i.e. $\overline{\tau} = \tau$.

▶ **Definition 8** (RCCS Processes). *The set of* reversible processes R *is built on top of the set of CCS processes by adding memory stacks to the threads:*

$$\begin{aligned} P, Q &:= P \mid Q \parallel \textstyle\sum_{i \geqslant 0} \lambda_i.P_i \parallel P\backslash a && \text{(CCS Processes)} \\ e &:= \langle i, \lambda, P \rangle && \text{(Memory Events)} \\ m &:= \emptyset \parallel \curlyvee.m \parallel e.m && \text{(Memory Stacks)} \\ T &:= m \rhd P && \text{(Reversible Threads)} \\ R, S &:= T \parallel R \mid S \parallel R\backslash a && \text{(RCCS Processes)} \end{aligned}$$

We denote $\mathsf{I}(e)$ *(resp* $\mathsf{I}(m)$, $\mathsf{I}(R)$*) the set of identifiers occurring in* e *(resp.* m, R*), and let* $\mathrm{nm}(m) = \{\alpha \mid \alpha \in \mathsf{N} \text{ or } \overline{\alpha} \in \overline{\mathsf{N}} \text{ occurs in } m\}$ *be the set of (co-)names occurring in* m.

Note that the nullary case of the sum[3] gives the inactive process, denoted 0, and that the unary case gives the "usual" prefixing of a process by a label, and we write e.g. $a.P$ for $\sum_1 \lambda_i.P_i$ with $\lambda_1 = a$ and $P_1 = P$. We assume sum to be associative and often consider only its binary case, that we denote with the $+$ sign. We often forget about the trailing $\emptyset$ in the memory stack, or the inactive process 0 and write e.g. $e \rhd a \mid (b + \overline{c})$ for $e.\emptyset \rhd (a.0 \mid (b.0 + \overline{c}.0))$. We work up to the structural congruence $\equiv$ of CCS – that we suppose familiar to the reader – in CCS processes under the memory, and write e.g. $\emptyset \rhd (P_1 \mid P_2 \mid P_3)$ without parenthesis since $(P_1 \mid P_2) \mid P_3 \equiv P_1 \mid (P_2 \mid P_3)$. Finally, the only binder is restriction, and alpha-equivalence, written $=_\alpha$ and supposed familiar, equates e.g. $((a + \overline{a}) \mid b)\backslash a$ with $((c + \overline{c}) \mid b)\backslash c$.

[3] This version of sum is used for simplifying the presentation of the LTS in Fig. 2, and always written with at least one prefix to ease remembering its particular form.

$$i \notin \mathsf{I}(m) \quad \frac{}{(m \rhd \lambda.P + Q) \xrightarrow{i:\lambda} \langle i, \lambda, Q\rangle.m \rhd P}\ \text{act.} \qquad \frac{R \xrightarrow{i:\lambda} R' \quad S \xrightarrow{i:\overline{\lambda}} S'}{R \mid S \xrightarrow{i:\tau} R' \mid S'}\ \text{syn.}$$

$$i \notin \mathsf{I}(m) \quad \frac{}{\langle i, \lambda, Q\rangle.m \rhd P \overset{i:\lambda}{\rightsquigarrow} m \rhd (\lambda.P + Q)}\ \text{act.}_* \qquad \frac{R \overset{i:\lambda}{\rightsquigarrow} R' \quad S \overset{i:\overline{\lambda}}{\rightsquigarrow} S'}{R \mid S \overset{i:\tau}{\rightsquigarrow} R' \mid S'}\ \text{syn.}_*$$

$$i \notin \mathsf{I}(S) \quad \frac{R \overset{i:\alpha}{\Longrightarrow} R'}{R \mid S \overset{i:\alpha}{\Longrightarrow} R' \mid S}\ \text{par.}_\text{L} \qquad i \notin \mathsf{I}(S) \quad \frac{S \overset{i:\alpha}{\Longrightarrow} S'}{R \mid S \overset{i:\alpha}{\Longrightarrow} R \mid S'}\ \text{par.}_\text{R}$$

$$\frac{R \overset{i:\alpha}{\Longrightarrow} R' \quad a \notin \alpha}{R\backslash a \overset{i:\alpha}{\Longrightarrow} R'\backslash a}\ \text{res.} \qquad \frac{R_1 \equiv R \quad R \overset{i:\alpha}{\Longrightarrow} R' \quad R' \equiv R'_1}{R_1 \overset{i:\alpha}{\Longrightarrow} R'_1}\ \equiv$$

Figure 2 Rules of the labeled transition system (LTS).

In a memory event $\langle i, \lambda, P\rangle$, the P component represents the process that was not chosen in a non-deterministic transition, but that can be restored if the process wants to go back. The "fork" symbol $\curlyvee$ tracks when a memory stack is split between two threads.

▶ **Definition 9** (Structural equivalence [2, 11]). *Structural equivalence on* R *is the smallest equivalence relation generated by the following rules:*

$$\frac{P =_\alpha Q}{m \rhd P \equiv m \rhd Q} \qquad (\alpha\text{-Conversion})$$

$$m \rhd (P \mid Q) \equiv (\curlyvee.m \rhd P \mid \curlyvee.m \rhd Q) \qquad \text{(Distribution of Memory)}$$

$$m \rhd P\backslash a \equiv (m \rhd P)\backslash a \text{ with } a \notin \mathrm{nm}(m) \qquad \text{(Scope of Restriction)}$$

The labeled transition system for RCCS is given by the rules of Fig. 2. We use $\overset{i:\alpha}{\Longrightarrow}$ for the union of $\xrightarrow{i:\alpha}$ (forward) and $\overset{i:\alpha}{\rightsquigarrow}$ (backward transition), and if there are indices $i_1, \ldots, i_n$ and labels $\alpha_1, \ldots, \alpha_n$ such that $R_1 \overset{i_1:\alpha_1}{\Longrightarrow} \cdots \overset{i_n:\alpha_n}{\Longrightarrow} R_n$, then we write $R_1 \Longrightarrow R_n$. Sect. 2.4 will provide examples of executions, but it should be noted that a process $m \rhd a.P$ is allowed to make a transition with label a and identifier $i \notin \mathsf{I}(m)$ using act. and add the event $\langle i, a, 0\rangle$ to the memory stack m if $i \notin \mathsf{I}(m)$. Conversely, a process $\langle i, a, 0\rangle.m \rhd P$ can do a backward transition using act.$_*$ with label a and identifier i and become $m \rhd a.P$. This system is a conservative extension over the LTS of CCS with prefixed sum, simply adding indices and backward transitions: in this sense, CCS can be seen as a sublanguage of RCCS, namely by identifying $\emptyset \rhd P$ with P and using only forward transitions.

▶ **Definition 10** (Reachable [2, Lemma 1]). *For all R, if there is a CCS process P such that $\emptyset \rhd P \Longrightarrow R$, we say that R is* reachable*, that P is the* unique origin *of R and write it O_R.*

An important result [11, Lemma 10] furthermore states that a forward-only[4] trace $\emptyset \rhd P \Longrightarrow R$ exists. Also, note that multiple RCCS processes can have the same origin, but that reachable RCCS processes have one unique origin (up to structural equivalence). We consider only reachable terms: unreachable terms are "dysfunctional" and their memory is considered *not coherent* [12], as they can not "rewind" back to an origin process.

[4] Traces and trace equivalences for RCCS are reminded in Appendix B: they are needed for some proofs and they are similar to their CCS's counterpart [8], but are not required to understand our results.

In RCCS, identifiers are needed to distinguish between events that have the same label. In CCS, events are considered "the same" up to labels: two events can synchrionise if they have the same label. Therefore, in the forward computation, events are free to choose a synchronization partner as long as they have the same label. Undoing a synchronization, however requires a precise pairs of events to backtrack: an event cannot change its synchronization partner when going backwards. Reversibility therefore needs to distinguish events are considered "the same" in the non-reversible setting. This constraint of using identifiers to distinguish events with the same label makes RCCS backward deterministic.

2.3 Processes and Memories as (Identified) Configuration Structures

In the definitions below, we write S for a synchronization algebra $(S, \star, \bot)$ with $S = \mathsf{N} \cup \overline{\mathsf{N}} \cup \{\tau\}$.

▶ **Definition 11** (Encoding CCS processes [42]). *Given a CCS process P, its encoding $[\![P]\!]$ as a configuration structure is built inductively:*

$$[\![\lambda.P + Q]\!] = [\![\lambda.P]\!] + [\![Q]\!] \qquad [\![P \mid Q]\!] = [\![P]\!] \mid_S [\![Q]\!] \qquad [\![P\backslash a]\!] = [\![P]\!]\upharpoonright_{\{a,\overline{a}\}}$$
$$[\![\lambda.P]\!] = \lambda.[\![P]\!] \qquad [\![0]\!] = \mathbf{0}$$

where S includes $\alpha \bullet \overline{\alpha} = \tau$ and $\alpha \bullet \beta = \bot$, if $\beta \neq \overline{\alpha}$.

▶ **Definition 12** (Encoding RCCS memories). *Given a RCCS process R, the encoding $\lceil R \rceil$ of its memory as an $\mathcal{I}$-structure is built by induction on the process and on the memory:*

$$\lceil m \rhd P \rceil = \lfloor m \rfloor \qquad \lceil R_1 \mid R_2 \rceil = \lceil R_1 \rceil \mid_S \lceil R_2 \rceil \qquad \lceil R\backslash a \rceil = \lceil R \rceil$$
$$\lfloor \langle i, \lambda, P \rangle.m \rfloor = \lfloor m \rfloor :: (\lambda, i) \qquad \lfloor \emptyset \rfloor = \mathbf{0} \qquad \lfloor \curlyvee.m \rfloor = \lfloor m \rfloor$$

where S includes $\alpha \bullet \overline{\alpha} = \tau$; $\alpha \bullet \alpha = \alpha$ and $\alpha \bullet \beta = \bot$ if $\beta \notin \{\overline{\alpha}, \alpha\}$.

Intuitively, a memory is a linear sequence of events executed by a process, which has resolved choices (i.e. in the sum or in synchronizations). The encoding of a memory is a chain of configurations. Appending a memory event corresponds to adding an event on top of the chain, and parallel composition combines two chains by fusing "partial" events resulting from a fork or a synchronization. We show examples in the following section and make this argument more formal in Lemma 17.

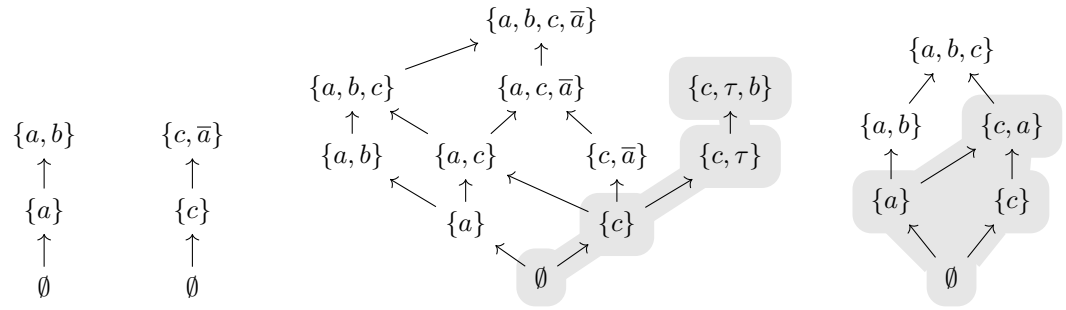
2.4 Examples

We now illustrate the execution of RCCS processes, the encoding of CCS processes and of RCCS memories, and how they relate.

▶ **Example 13** (Executing a RCCS process). An example of forward-only trace is:

$$\begin{aligned}
\emptyset \rhd (a.b \mid c.\overline{a}) \equiv\ & (\curlyvee.\emptyset \rhd a.b) \mid (\curlyvee.\emptyset \rhd c.\overline{a}) && \text{(Distribution of Memory)}\\
\xrightarrow{1:c}\ & (\curlyvee.\emptyset \rhd a.b) \mid (\langle 1, c, 0\rangle.\ \curlyvee.\emptyset \rhd \overline{a}) && \text{(act.)}\\
\xrightarrow{2:\tau}\ & (\langle 2, a, 0\rangle.\ \curlyvee.\emptyset \rhd b) \mid (\langle 2, \overline{a}, 0\rangle.\langle 1, c, 0\rangle.\ \curlyvee.\emptyset \rhd 0) && \text{(syn.)}\\
\xrightarrow{3:b}\ & (\langle 3, b, 0\rangle.\langle 2, a, 0\rangle.\ \curlyvee.\emptyset \rhd 0) \mid (\langle 2, \overline{a}, 0\rangle.\langle 1, c, 0\rangle.\ \curlyvee.\emptyset \rhd 0) && \text{(act.)}
\end{aligned}$$

Reading it from end to beginning and replacing $\xrightarrow{\vdots}$ with $\xleftarrow{\vdots}$ gives a *backward-only* trace, that would rewind the process back to its origin. Of course, a trace can mix forward and backward transitions, as we illustrate in Example 20. The memory of this process is encoded in Example 15.



(a) $[\![a.b]\!]$ **(b)** $[\![c.\overline{a}]\!]$ **(c)** $[\![(a.b) \mid (c.\overline{a})]\!]$ **(d)** $[\![a.b \mid c]\!]$

Figure 3 $\mathcal{I}$-structures for Examples 14, 15, 16 and 20, with the CCS term encoded by their underlying configuration in caption.

▶ **Example 14** (Encoding CCS processes)**.** We can see the $\mathcal{I}$-structures from Fig. 1 as configuration structures obtained by encoding the CCS processes $a + a$, $a \mid a$, and $(a.b) \mid \overline{a}$. Similarly, we can consider the $\mathcal{I}$-structures from Fig. 3 – ignoring the grayed out parts for now – as configuration structures. The interested reader can check that the encoding of $(a.b) \mid (c.\overline{a})$ in Fig. 3c is indeed the result of applying the parallel composition of configurations structures (Definition 6) to the encoding of $a.b$ in Fig. 3a and of $c.\overline{a}$ in Fig. 3b. Lastly, Fig. 3d shows the encoding of $(a.b) \mid c$.

The parallel composition of $\mathcal{I}$-structures (Definition 7) differs slightly and is new, and hence deserves a detailed example.

▶ **Example 15** (Encoding RCCS memories)**.** The process obtained at the end of Example 13 has its memory encoded as follows:

$$\begin{aligned}
&\lceil(\langle 3,b,0\rangle.\langle 2,a,0\rangle.\ \curlyvee\ .\emptyset \rhd 0) \mid (\langle 2,\overline{a},0\rangle.\langle 1,c,0\rangle.\ \curlyvee\ .\emptyset \rhd 0)\rceil \\
=&\lceil\langle 3,b,0\rangle.\langle 2,a,0\rangle.\ \curlyvee\ .\emptyset \rhd 0\rceil \mid \lceil\langle 2,\overline{a},0\rangle.\langle 1,c,0\rangle.\ \curlyvee\ .\emptyset \rhd 0\rceil \\
=&\lfloor\langle 3,b,0\rangle.\langle 2,a,0\rangle.\ \curlyvee\ .\emptyset\rfloor \mid \lfloor\langle 2,\overline{a},0\rangle.\langle 1,c,0\rangle.\ \curlyvee\ .\emptyset\rfloor
\end{aligned}$$

Letting $E = L = \{a, \overline{a}, b, c\}$, $\ell = \mathrm{id}$, $I = \{1, 2, 3\}$, using $\lfloor \curlyvee.\emptyset\rfloor = \lfloor\emptyset\rfloor = \mathbf{0}$ and the postfixing:

$$\begin{aligned}
\lfloor\langle 3,b,0\rangle.\langle 2,a,0\rangle.\ \curlyvee\ .\emptyset\rfloor &= (\{a,b\}, \{\emptyset, \{a\}, \{a,b\}\}, L, \ell, I, \{a \mapsto 2, b \mapsto 3\} \\
\lfloor\langle 2,\overline{a},0\rangle.\langle 1,c,0\rangle.\ \curlyvee\ .\emptyset\rfloor &= (\{c,\overline{a}\}, \{\emptyset, \{c\}, \{c,\overline{a}\}\}, L, \ell, I, \{c \mapsto 1, \overline{a} \mapsto 2\})
\end{aligned}$$

Those are displayed in Fig. 3a and 3b, and their product (which is the first step to compute their parallel composition) gives the following sets of events and identifiers:

Event	$(a,\star)$	$(b,\star)$	$(\star,c)$	$(\star,\overline{a})$	(a,c)	$(a,\overline{a})$	(b,c)	$(b,\overline{a})$
Identifier	$(2,\star)$	$(3,\star)$	$(\star,1)$	$(\star,2)$	$(2,1)$	$(2,2)$	$(3,1)$	$(3,2)$

Re-identifying and re-labeling according to the definition of parallel composition gives:

Event	$(a,\star)$	$(b,\star)$	$(\star,c)$	$(\star,\overline{a})$	(a,c)	$(a,\overline{a})$	(b,c)	$(b,\overline{a})$
Re-identified	$\bot_{(2,\star)}$	3	1	$\bot_{(\star,2)}$	$\bot_{(2,1)}$	2	$\bot_{(3,1)}$	$\bot_{(3,2)}$
Re-labeled	$\bot$	b	c	$\bot$	$\bot$	τ	$\bot$	$\bot$

Indeed, if two events occur at the same time with the same identifier (Sync. or Fork), then their identifier is simply picked. Hence, $m'(a,\overline{a}) = 2$. If only one event is present in the pair, and no event on the other component have the same identifier (Extra $\star$. 1, Extra $\star$. 2),

then this event's identifier is picked. Hence, $m'(b,\star) = 3$ and $m'(\star,c) = 1$. The remaining cases get re-identified with $\bot_k$ (Error). Finally, $(b,\star)$, $(\star,c)$ and $(a,\overline{a})$ gets relabeled with b, c and τ respectively, and after restricting to the label $\bot$ we obtain the grayed out part of Fig. 3c.

Observe that in this last example, the structure underlying the encoding of the memory is just a particular "path" in the encoding of the origin. We can observe this intuition again with the following example:

▶ **Example 16** (Memory and origin)**.** The encoding of the memory resulting from the execution $\emptyset \rhd ((a.b) \mid c) \xrightarrow{1:c} \xrightarrow{2:a} (\langle 2,a,0\rangle.\, \curlyvee .\emptyset \rhd b) \mid (\langle 1,c,0\rangle.\, \curlyvee .\emptyset \rhd 0)$ is the grayed out part in Fig. 3d, with $m(c) = 1$ and $m(a) = 2$. We name this process R_1 and come back to it in Example 20.

2.5 Operational Correspondence

Before studying bisimulations on configuration structures and processes, we prove the operational correspondence[5] between RCCS processes and the encodings of their memories in $\mathcal{I}$-structures (Lemma 19, cf. also Sect. C.2). Events in $\mathcal{I}$-structures resulting from the encoding of a process have different identifiers, and they are either causally linked or concurrent.

▶ **Lemma 17** (Memories give posets)**.** *For all R, letting x be the maximal configuration in $\lceil R \rceil$ (Definition 2), $(\lceil R \rceil, \subseteq)$ is a partially ordered set (poset) with maximal element x.*

This is proved by induction on R and illustrated by Examples 15 and 16. However, having at most one maximal configuration does not imply that one particular event has to be "the last one" introduced. We use the following definition to make it formal.

▶ **Definition 18** (Maximal event)**.** *An event e is* maximal *in $\mathcal{I}$ if there is no event e' such that $e <_x e'$, for x a maximal configuration of $\mathcal{I}$.*

For instance, the encoding of the memory of Example 16, pictured in Fig. 3d, has two maximal events, labeled a and c. We can now state the main result of this section:

▶ **Lemma 19** (Operational Correspondence)**.** *For all R and S, writing $(E_R, C_R, \ell_R, I_R, m_R)$ for $\lceil R \rceil$ and similarly for S, if $R \xrightarrow{i:\alpha} S$ or $S \overset{i:\alpha}{\rightsquigarrow} R$, then there exists $e \in E_S$ maximal in $\lceil S \rceil$ with $m_S(e) = i$ s.t. $\lceil R \rceil \cong \lceil S \rceil \restriction_{\{e\}}$. For all R and e a maximal event in $\lceil R \rceil$, there is a transition $R \overset{m_R(e):\ell_R(e)}{\rightsquigarrow} S$ with $\lceil S \rceil \cong \lceil R \rceil \restriction_{\{e\}}$.*

For the first part, it suffices to show that the forward transition triggers the creation of a maximal event with the same identifier, and nothing else, and that this event can be "traced" in $\lceil S \rceil$. It uses intermediate lemma (Lemmas 43–45) showing how maximal events are preserved by certain operations on $\mathcal{I}$-structures. The result follows easily for backward transitions, but the last part is more involved: it requires to show that e can be mapped to a particular transition in the trace from O_R to R, and, using a notion of trace equivalence, that this particular transition can be "postponed" and done last, so that R can backtrack on it.

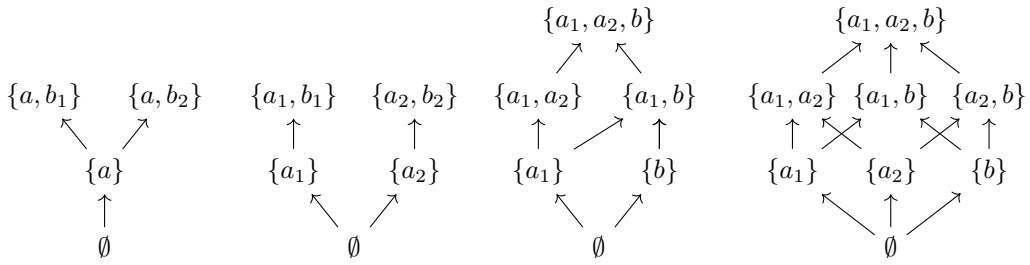
▶ **Example 20** (Forward and backward transitions)**.** Looking back at the process of Example 16, we could further have $R_1 \overset{1:c}{\rightsquigarrow} R_2 \xrightarrow{3:b} R_3$, i.e.

$$(\langle 2,a,0\rangle.\, \curlyvee .\emptyset \rhd b) \mid (\langle 1,c,0\rangle.\, \curlyvee .\emptyset \rhd 0) \overset{1:c}{\rightsquigarrow} (\langle 2,a,0\rangle.\, \curlyvee .\emptyset \rhd b) \mid \curlyvee .\emptyset \rhd c)) \quad (\text{act.}_*)$$

$$\xrightarrow{3:b} (\langle 3,b,0\rangle.\langle 2,a,0\rangle.\, \curlyvee .\emptyset \rhd 0) \mid \curlyvee .\emptyset \rhd c)) \quad (\text{act.})$$

We can see using Fig. 3d that $\lceil R_1 \rceil \restriction_c = \lceil R_2 \rceil$ and that $\lceil R_2 \rceil = \lceil R_3 \rceil \restriction_b$.

[5] Similar to the operational correspondance between configuration structures and CCS processes [7, Section 3] i.e., if $P \xrightarrow{\alpha} Q$, then $[\![P]\!] = [\![Q]\!] \restriction_{\{e\}}$, where e is an event in $[\![P]\!]$ such that $\ell(e) = \alpha$.



(a) $[\![a.(b+b)]\!]$ **(b)** $[\![(a.b)+(a.b)]\!]$ **(c)** $[\![(a.a) \mid b]\!]$ **(d)** $[\![a \mid a \mid b]\!]$

Figure 4 Configuration structures for Examples 23 and 27.

3 Reversible and Truly Concurrent Bisimulations Are the Same

3.1 History-Preserving Bisimulations in Configuration Structures

History-preserving bisimulation (HPB) [4, 31, 32] and hereditary history-preserving bisimulation (HHPB) [4, 6] are equivalences on configuration structures that use label- and order-preserving bijections on events. Below, assume given $\mathcal{C}_i = (E_i, C_i, L_i, \ell_i)$ for $i = 1, 2$.

▶ **Definition 21** (Label- and order-preserving functions (l&o-p)). *A function $f : x_1 \to x_2$, for $x_i \in C_i$, $i \in \{1, 2\}$ is* label-preserving *if $\ell_1(e) = \ell_2(f(e))$ for all $e \in x_1$. It is* order-preserving *if $e_1 \leqslant_{x_1} e_2 \Rightarrow f(e_1) \leqslant_{x_2} f(e_2)$, for all $e_1, e_2 \in x_1$. We write that f is* l&o-p *if it is both.*

▶ **Definition 22** (HPB and HHPB). *A relation $\mathcal{R} \subseteq C_1 \times C_2 \times (E_1 \rightharpoonup E_2)$ such that $(\emptyset, \emptyset, \emptyset) \in \mathcal{R}$, and if $(x_1, x_2, f) \in \mathcal{R}$, then f is a l&o-p bijection between x_1 and x_2 and (1) and (2) (resp. (1–4)) hold is called a* history- *(resp.* hereditary history-*)* preserving bisimulation (HPB, resp. HHPB) between $\mathcal{C}_1$ and $\mathcal{C}_2$.

$$\forall y_1, x_1 \xrightarrow{e_1} y_1 \Rightarrow \exists y_2, g, x_2 \xrightarrow{e_2} y_2, g\restriction_{x_1} = f, (y_1, y_2, g) \in \mathcal{R} \tag{1}$$
$$\forall y_2, x_2 \xrightarrow{e_2} y_2 \Rightarrow \exists y_1, g, x_1 \xrightarrow{e_1} y_1, g\restriction_{x_1} = f, (y_1, y_2, g) \in \mathcal{R} \tag{2}$$
$$\forall y_1, x_1 \overset{e_1}{\rightsquigarrow} y_1 \Rightarrow \exists y_2, g, x_2 \overset{e_2}{\rightsquigarrow} y_2, g = f\restriction_{y_1}, (y_1, y_2, g) \in \mathcal{R} \tag{3}$$
$$\forall y_2, x_2 \overset{e_2}{\rightsquigarrow} y_2 \Rightarrow \exists y_1, g, x_1 \overset{e_1}{\rightsquigarrow} y_1, g = f\restriction_{y_1}, (y_1, y_2, g) \in \mathcal{R} \tag{4}$$

We write that $\mathcal{C}_1$ and $\mathcal{C}_2$ are HHPB (resp. HPB) *if there exists a* HHPB *(resp.* HPB*) relation between them.*

Note that HPB and HHPB are two different relations, as e.g. $(a \mid (b+c))+(a \mid b)+((a+c) \mid b)$ and $(a \mid (b+c)) + ((a+c) \mid b)$ have HPB but not HHPB encodings [37] .

▶ **Example 23.** The encoding of the two processes $a.(b+b)$ and $(a.b)+(a.b)$, in Fig. 4a and 4b, are HHPB: the relation can start by mapping a to a_1 or a_2, and then maps b_1 or b_2 (depending on the superset reached) to b_1 or b_2, according to the first choice made. This relation can "follow" the forward and backward movements in both structures, giving a l&o-p bijection. This example also proves that HHPB is not CCS's structural congruence.

3.2 Back-And-Forth Bisimulations in Reversible CCS

This section presents the relations we will be using, explain the restrictions on previous attempts to capture HHPB as a relation on process algebra terms [2, 29], and shows why both backward transitions *and* identifiers are needed to capture HHPB.

Below, assume given two reachable processes R_1 and R_2, and if $f : A \to B$ is such that $f(a) = b$, we write $f \setminus \{a \mapsto b\}$ for $f\restriction_{A\setminus\{a\}}$.

▶ **Definition 24** (B&F and SB&F bisimulations). *A relation $\mathcal{R} \subseteq \mathsf{R} \times \mathsf{R} \times (\mathsf{I} \rightharpoonup \mathsf{I})$ such that $(\emptyset \rhd O_{R_1}, \emptyset \rhd O_{R_2}, \emptyset) \in \mathcal{R}$ and if $(R_1, R_2, f) \in \mathcal{R}$, then f is a bijection between $\mathsf{I}(R_1)$ and $\mathsf{I}(R_2)$ and (5–8) hold is called* a back-and-forth bisimulation (B&F) between R_1 and R_2.

$$\forall S_1, R_1 \xrightarrow{i:\alpha} S_1 \Rightarrow \exists S_2, g, R_2 \xrightarrow{j:\alpha} S_2, g = f \cup \{i \mapsto j\}, (S_1, S_2, g) \in \mathcal{R} \tag{5}$$

$$\forall S_2, R_2 \xrightarrow{i:\alpha} S_2 \Rightarrow \exists S_1, g, R_1 \xrightarrow{j:\alpha} S_1, g = f \cup \{i \mapsto j\}, (S_1, S_2, g) \in \mathcal{R} \tag{6}$$

$$\forall S_1, R_1 \overset{i:\alpha}{\rightsquigarrow} S_1 \Rightarrow \exists S_2, f, R_2 \overset{j:\alpha}{\rightsquigarrow} S_2, g = f\setminus\{i \mapsto j\}, (S_1, S_2, g) \in \mathcal{R} \tag{7}$$

$$\forall S_2, R_2 \overset{i:\alpha}{\rightsquigarrow} S_2 \Rightarrow \exists S_1, g, R_1 \overset{j:\alpha}{\rightsquigarrow} S_1, g = f\setminus\{i \mapsto j\}, (S_1, S_2, g) \in \mathcal{R} \tag{8}$$

If we remove the requirements on f and g in the second part of (5–8), we call $\mathcal{R}$ a simple back-and-forth bisimulation (SB&F). *We write that R_1 and R_2 are* B&F *(resp.* SB&F*) if there exists a* B&F *(resp.* SB&F*) relation between them.*

Other back-and-forth bisimulations have been studied for a variety of systems [15, 25], and are similar to SB&F bisimulation in the sense that they focus on backward and forward moves, but did not took the identifiers into account.

Restrictions and Previous Results.

▶ **Definition 25** (Constraints). *Given $\mathcal{C}$, if $\forall x \in C$, $\forall e_1, e_2 \in x$, $\ell(e_1) = \ell(e_2)$ implies $e_1 = e_2$, then $\mathcal{C}$ is* non-repeating *[29, Definition 3.5]. If, $\forall x \in C$, $\forall e_1, e_2 \in x$, $e_1 \operatorname{co}_x e_2$ and $\ell(e_1) = \ell(e_2)$ implies $e_1 = e_2$, then $\mathcal{C}$ is* without auto-concurrency *[37, Definition 9.5]. A process R is* non-repeating *(resp.* without auto-concurrency*) if $[\![O_R]\!]$ is.*

Those are the constraints used in showing equivalences between process algebra and configuration structures. We omitted the definition of *singly labeled* [2, Definition 26], as it does not contribute to the understanding of our results. Every non-repeating process is without auto-concurrency, but being non-repeating events and singly labeled are incomparable features. Simple processes can be repeating (e.g. $a.a$), with auto-concurency (e.g. $(a.b) \mid a$), not singly labeled (e.g. $a + a$), and the same can be said of more complex processes using similar patterns.

The first syntactical characterization of HHPB was obtained on non-repeating processes, using the "forward-reverse bisimulation" (FR) [30, Definition 6.5], which is essentially defined as B&F, with the additional requirement that $f = \mathrm{id}$. The theorem states that non-repeating CCSK processes are FR iff their encoding are HHPB [29, Theorem 5.4]. We argue that FR gives too much importance to the technical apparatus implementing reversibility. To the best of our knowledge, freshness is the only constrain imposed on identifiers in reversible works. Hence, to obtain meaningful FR equivalences, one would have to force a particular strategy for choosing the identifiers[6]. We prefer instead to impose only the freshness constraint on the identifiers, and use bijections (instead of equality on identifier) in our equivalences.

[6] For example, a possible strategy is to always pick the smallest available identifier, which then guarantees that both events labelled a in $(a.b) + (a.b)$ of Example 23 picks the same identifier, in their respective execution trace. Using this strategy we have then that $(a.b) + (a.b)$ and $a.(b + b)$ are FR equivalent. However, this strategy makes selecting an identifier a bottleneck of the system, as all the processes have to check which is the smallest available from the same "pool".

A second attempt [2] to capture HHPB used a *back-and-forth barbed congruence* on RCCS processes which was proven to correspond to HHPB on their encodings for a restricted class of processes as well, the class of singly-labeled processes.

Pinpointing the Right Reversible Bisimulation. We lift both restrictions in Corollary 31, by proving that B&F captures HHPB on *all* processes. Before doing so, let us note that even though B&F is the right notion to capture HHPB, when restricted to non-repeating processes, which are also without auto-concurrency, it does not use in a meaningful way the identifiers.

▶ **Theorem 26** (Collapsing B&F and SB&F)**.** *If R_1 and R_2 are without auto-concurrency, then they are* B&F *iff they are* SB&F*.*

The intuition is that since two concurrent transitions sharing the same label can not be fired at the same time, the identifiers do not add any information. The proof is easy for the forward transitions, and uses an order on the transitions enforced by causality for the backward traces. In the presence of auto-concurrency, the relations differ, e.g. the process with auto-concurrency $a \mid a$ and $a.a$ are SB&F but not B&F.

▶ **Example 27** (Reversibility is *not* "just back and forth")**.** Observe that the bisimulation relation obtained by only considering (5–6) and ignoring the identifiers in Definition 24 is the "standard" CCS bisimulation. Hence, it could seem natural to assume that "simply adding the backwards transitions", i.e. taking (5–8) without the identifiers, giving SB&F, would be "the right" bisimulation for RCCS. Processes like $(a.a) \mid b$ and $a \mid a \mid b$ are SB&F, but their encodings, presented in Fig. 4c and 4d, are not HPB and hence not HHPB: SB&F does not account for reversibility in a satisfactory manner.

Both the bijection on identifiers *and* backward transitions are necessary to capture HHPB. Indeed, as suggested by Example 27, "simply" considering forward and backward transitions is not enough. Let us now consider the role of the bijection on identifiers a bit further. A first remark is that Theorem 26 shows that it is easy to overlook the role of identifiers when restricting the class of processes considered. Secondly, we can prove, as an immediate corollary of Theorems 29 and 30, that considering only (5–6) (*with* the identifiers) in Definition 24 gives a characterization of HPB (Corollary 49): if anything, having a bijection between identifiers – thanks to the order on events that can be deducted from it – helps getting closer to "truly concurrent" bisimulation than adding backward transitions does. However, as HPB and HHPB do not coincide, the identifiers are not enough either.

Of course, similar mechanisms could achieve similar results, but it is our hope that reversibility is fully understood as not "just" being about adding backward transitions or memories, but to use both to obtain backward determinism.

3.3 History-Preserving Bisimulations in (R)CCS

Proving our main result (Corollary 31) will use intermediate relations on processes – called HPB and HHPB as well – that use the encoding of the memories into $\mathcal{I}$-structures. Those relations are proven to correspond to (H)HPB on the encoding of the processes on one hand (Theorem 29), and the one that characterizes HHPB is proven to coincide with B&F (Theorem 30) on the other hand. The connections between formalisms and additional discussion are gathered in Sect. C.3.

▶ **Definition 28** (HPB and HHPB on RCCS). *A relation* $\mathcal{R} \subseteq \mathsf{R} \times \mathsf{R} \times (E_1 \rightharpoonup E_2)$ *such that* $(\emptyset \rhd O_{R_1}, \emptyset \rhd O_{R_2}, \emptyset) \in \mathcal{R}$ *and if* $(R_1, R_2, f) \in \mathcal{R}$ *then* f *is an isomorphism between* $\lceil R_1 \rceil$ *and* $\lceil R_2 \rceil$ *and (9) and (10) (resp. (9–12)) hold is called a* history-*(resp.* hereditary history-*)* preserving bisimulation between R_1 and R_2.

$$\forall S_1, R_1 \xrightarrow{i:\alpha} S_1 \Rightarrow \exists S_2, g, R_2 \xrightarrow{j:\alpha} S_2, g\restriction_{\lceil R_1 \rceil} = f, (S_1, S_2, g) \in \mathcal{R} \tag{9}$$

$$\forall S_2, R_2 \xrightarrow{i:\alpha} S_2 \Rightarrow \exists S_1, g, R_1 \xrightarrow{j:\alpha} S_1, g\restriction_{\lceil R_1 \rceil} = f, (S_1, S_2, g) \in \mathcal{R} \tag{10}$$

$$\forall S_1, R_1 \overset{i:\alpha}{\rightsquigarrow} S_1 \Rightarrow \exists S_2, f, R_2 \overset{j:\alpha}{\rightsquigarrow} S_2, g = f\restriction_{\lceil S_1 \rceil}, (S_1, S_2, g) \in \mathcal{R} \tag{11}$$

$$\forall S_2, R_2 \overset{i:\alpha}{\rightsquigarrow} S_2 \Rightarrow \exists S_1, g, R_1 \overset{j:\alpha}{\rightsquigarrow} S_1, g = f\restriction_{\lceil S_1 \rceil}, (S_1, S_2, g) \in \mathcal{R} \tag{12}$$

where we write $g\restriction_{\lceil R \rceil}$ *for the restriction of each component of* g *to* $\lceil R \rceil$.

We write that R_1 and R_2 are (H)HPB *if there exists a* (H)HPB *relation between them.*

Note that the definitions above reflect the definition of (H)HPB (Definition 22): the condition $(\emptyset \rhd O_{R_1}, \emptyset \rhd O_{R_2}, \emptyset) \in \mathcal{R}$ is intuitively the counterpart to the condition that $(\emptyset, \emptyset, \emptyset)$ has to be included in the relation on configuration structures. Also, f shares similarity with the l&o-p bijection, in the sense that it exists, with the identity as the component on the labels, iff there exists a l&o-p bijection between the unique maximal configurations in $\lceil R_1 \rceil$ and $\lceil R_2 \rceil$ (Lemma 46).

Relations defined on RCCS (B&F, SB&F, HPB and HHPB) straightforwardly extend to CCS, by simply stating that P_1 and P_2 are in it if $\emptyset \rhd P_1$ and $\emptyset \rhd P_2$ are too. Therefore we can state below our results in terms of CCS processes.

▶ **Theorem 29** (Equivalences). *P_1 and P_2 are* HHPB *(resp.* HPB*) iff $[\![P_1]\!]$ and $[\![P_2]\!]$ are.*

This result can easily be extended to *weak*-HPB and *weak*-HHPB [6, 31], which are defined by removing from (H)HPB on configuration structures (Definition 22) and on RCCS (Definition 28) the condition that f must be preserved from one step to the next one.

▶ **Theorem 30** (Equivalence (contd)). *P_1 and P_2 are* B&F *iff they are* HHPB.

Theorem 29 (resp. Theorem 30) uses our operational correspondence between RCCS processes (resp. RCCS memories) and their encodings as configuration structures [2, Lemma 6] (resp. as $\mathcal{I}$-structure (Lemma 19)) to transition between the semantic and syntactic worlds.

Our main result will come as an immediate corollary of Theorems 29 and 30.

▶ **Corollary 31** (Main result). *P_1 and P_2 are* B&F *iff $[\![P_1]\!]$ and $[\![P_2]\!]$ are* HHPB.

4 Concluding Remarks

This work offers a "definitive" answer to the question of finding a meaningful bisimulation for reversible LTS by providing relations that correspond to (H)HPB on their encodings on *all* processes. We believe this contribution is of value because:

1. This result solves a problem that was open since HHPB was defined [6], nearly 30 years ago, for which despite the use of multiple techniques, only partial results were obtained,
2. This idea in appearance simple still requires a lot of technical work, as sketched in the Appendix and detailed in `https://hal.archives-ouvertes.fr/hal-02568250`,
3. The use of reversibility (both the backtracking capability *and* the memory mechanism) is critical to characterize HHPB on syntactical terms.

This result also enforces the importance of identifiers in general and not just as part of a backtracking mechanism. Indeed, they are generally already present when concurrency is implemented, e.g. when two `Unix` threads terminate with the same signal, the parent process have the capacity of determining which process sent which signal.

As a byproduct of our result, we also proposed an encoding of RCCS memories into an "enriched" configuration structure, called identified configuration structure. This observation echoes our previous formalism [2] and similar encoding [18] in an interesting way: as mentioned in the Introduction, a reversible process R was encoded as a pair $(\llbracket O_R \rrbracket, x_R)$ made of the configuration structure encoding the origin of R, and a configuration x_R in it, called the *address* of R. The intuition was that we could "match" a partially executed process with a configuration. We can now go further by observing that $\lceil R \rceil$ is isomorphic to the $\mathcal{I}$-structure generated by x_R, which is everything "below" it. This result (Lemma 48) is used in our proof, and exemplified by Example 16: the encoding of the memory of $(\langle 2, a, 0\rangle.\ \curlyvee\ .\emptyset \rhd b) \mid (\langle 1, c, 0\rangle.\ \curlyvee\ .\emptyset \rhd 0)$ corresponds to the "past" of the process, whose underlying structure is grayed out in Fig. 3d, and what is left to execute – $b \mid 0$ – corresponds to the "future" of that process, and is represented by the configuration $\{c, a, b\}$ in Fig. 3d.

References

1 Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell, Brian Burkett, Yu Chen, Zijun Chen, Ben Chiaro, Roberto Collins, William Courtney, Andrew Dunsworth, Edward Farhi, Brooks Foxen, Austin Fowler, Craig Gidney, Marissa Giustina, Rob Graff, Keith Guerin, Steve Habegger, Matthew P. Harrigan, Michael J. Hartmann, Alan Ho, Markus Hoffmann, Trent Huang, Travis S. Humble, Sergei V. Isakov, Evan Jeffrey, Zhang Jiang, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Paul V. Klimov, Sergey Knysh, Alexander Korotkov, Fedor Kostritsa, David Landhuis, Mike Lindmark, Erik Lucero, Dmitry Lyakh, Salvatore Mandrà, Jarrod R. McClean, Matthew McEwen, Anthony Megrant, Xiao Mi, Kristel Michielsen, Masoud Mohseni, Josh Mutus, Ofer Naaman, Matthew Neeley, Charles Neill, Murphy Yuezhen Niu, Eric Ostby, Andre Petukhov, John C. Platt, Chris Quintana, Eleanor G. Rieffel, Pedram Roushan, Nicholas C. Rubin, Daniel Sank, Kevin J. Satzinger, Vadim Smelyanskiy, Kevin J. Sung, Matthew D. Trevithick, Amit Vainsencher, Benjamin Villalonga, Theodore White, Z. Jamie Yao, Ping Yeh, Adam Zalcman, Hartmut Neven, and John M. Martinis. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, October 2019. doi:10.1038/s41586-019-1666-5.

2 Clément Aubert and Ioana Cristescu. Contextual equivalences in configuration structures and reversibility. *Journal of Logical and Algebraic Methods in Programming*, 86(1):77–106, 2017. doi:10.1016/j.jlamp.2016.08.004.

3 Holger Bock Axelsen and Robert Glück. On reversible turing machines and their function universality. *Acta Informatica*, 53(5):509–543, 2016. doi:10.1007/s00236-015-0253-y.

4 Paolo Baldan and Silvia Crafa. Hereditary history-preserving bisimilarity: Logics and automata. In Jacques Garrigue, editor, *APLAS*, volume 8858 of *Lecture Notes in Computer Science*, pages 469–488. Springer, 2014. doi:10.1007/978-3-319-12736-1_25.

5 Paolo Baldan and Silvia Crafa. A logic for true concurrency. *Journal of the ACM*, 61(4):24, 2014. doi:10.1145/2629638.

6 Marek A. Bednarczyk. Hereditary history preserving bisimulations or what is the power of the future perfect in program logics. Technical report, Instytut Podstaw Informatyki PAN filia w Gdańsku, 1991. URL: http://www.ipipan.gda.pl/~marek/papers/historie.ps.gz.

7 Gérard Boudol and Ilaria Castellani. On the semantics of concurrency: Partial orders and transition systems. In Hartmut Ehrig, Robert A. Kowalski, Giorgio Levi, and Ugo Montanari, editors, *TAPSOFT'87*, volume 249 of *Lecture Notes in Computer Science*, pages 123–137. Springer, 1987. doi:10.1007/3-540-17660-8_52.

8 Gérard Boudol and Ilaria Castellani. Permutation of transitions: An event structure semantics for CCS and SCCS. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop, Noordwijkerhout, The Netherlands, May 30 - June 3, 1988, Proceedings*, volume 354 of *Lecture Notes in Computer Science*, pages 411–427. Springer, 1988. doi:10.1007/BFb0013028.

9 Ioana Cristescu, Jean Krivine, and Daniele Varacca. A compositional semantics for the reversible p-calculus. In *LICS*, pages 388–397. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.45.

10 Ioana Cristescu, Jean Krivine, and Daniele Varacca. Rigid families for CCS and the π-calculus. In Martin Leucker, Camilo Rueda, and Frank D. Valencia, editors, *Theoretical Aspects of Computing - ICTAC 2015 - 12th International Colloquium Cali, Colombia, October 29-31, 2015, Proceedings*, volume 9399 of *Lecture Notes in Computer Science*, pages 223–240. Springer, 2015. doi:10.1007/978-3-319-25150-9_14.

11 Vincent Danos and Jean Krivine. Reversible communicating systems. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 292–307. Springer, 2004. doi:10.1007/978-3-540-28644-8_19.

12 Vincent Danos and Jean Krivine. Transactions in RCCS. In Martín Abadi and Luca de Alfaro, editors, *CONCUR*, volume 3653 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 2005. doi:10.1007/11539452_31.

13 Philippe Darondeau and Pierpaolo Degano. Causal trees: Interleaving + causality. In Irène Guessarian, editor, *Semantics of Systems of Concurrent Processes, LITP Spring School on Theoretical Computer Science, La Roche Posay, France, April 23-27, 1990, Proceedings*, volume 469 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 1990. doi:10.1007/3-540-53479-2_10.

14 David de Frutos-Escrig, Maciej Koutny, and Lukasz Mikulski. Reversing steps in petri nets. In Susanna Donatelli and Stefan Haar, editors, *Application and Theory of Petri Nets and Concurrency - 40th International Conference, PETRI NETS 2019, Aachen, Germany, June 23-28, 2019, Proceedings*, volume 11522 of *Lecture Notes in Computer Science*, pages 171–191. Springer, 2019. doi:10.1007/978-3-030-21571-2_11.

15 Rocco De Nicola, Ugo Montanari, and Frits W. Vaandrager. Back and forth bisimulations. In Jos C. M. Baeten and Jan Willem Klop, editors, *CONCUR '90*, volume 458 of *Lecture Notes in Computer Science*, pages 152–165. Springer, 1990. doi:10.1007/BFb0039058.

16 Michael P. Frank. Foundations of generalized reversible computing. In Iain Phillips and Hafizur Rahaman, editors, *Reversible Computation - 9th International Conference, RC 2017, Kolkata, India, July 6-7, 2017, Proceedings*, volume 10301 of *Lecture Notes in Computer Science*, pages 19–34. Springer, 2017. doi:10.1007/978-3-319-59936-6_2.

17 Michael P. Frank. Throwing computing into reverse. *IEEE Spectrum*, 54(9):32–37, September 2017. doi:10.1109/MSPEC.2017.8012237.

18 Eva Graversen, Iain Phillips, and Nobuko Yoshida. Event structure semantics of (controlled) reversible CCS. In Jarkko Kari and Irek Ulidowski, editors, *Reversible Computation - 10th International Conference, RC 2018, Leicester, UK, September 12-14, 2018, Proceedings*, volume 11106 of *Lecture Notes in Computer Science*, pages 102–122. Springer, 2018. doi:10.1007/978-3-319-99498-7_7.

19 Thomas Troels Hildebrandt. *Categorical Models for Concurrency: Independence, Fairness and Dataflow*. PhD thesis, BRICS, University of Aarhus, February 2000. URL: http://www.brics.dk/DS/00/1/.

20 Markus Holzer and Martin Kutrib. Reversible nondeterministic finite automata. In Iain Phillips and Hafizur Rahaman, editors, *Reversible Computation*, pages 35–51. Springer International Publishing, 2017. doi:10.1007/978-3-319-59936-6_3.

21 André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996. doi:10.1006/inco.1996.0057.

22 Yonggun Jun, Momčilo Gavrilov, and John Bechhoefer. High-precision test of landauer's principle in a feedback trap. *Physical Review Letters*, 113:190601, November 2014. doi:10.1103/PhysRevLett.113.190601.

23 Vasileios Koutavas and Matthew Spaccasassi, Carloand Hennessy. Bisimulations for communicating transactions - (extended abstract). In Anca Muscholl, editor, *FoSSaCS*, volume 8412 of *Lecture Notes in Computer Science*, pages 320–334. Springer, 2014. doi:10.1007/978-3-642-54830-7_21.

24 Ivan Lanese, Doriana Medić, and Claudio Antares Mezzina. Static versus dynamic reversibility in CCS. *Acta Informatica*, November 2019. doi:10.1007/s00236-019-00346-6.

25 Ivan Lanese, Claudio Antares Mezzina, and Jean-Bernard Stefani. Reversing higher-order pi. In Paul Gastin and François Laroussinie, editors, *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 478–493. Springer, 2010. doi:10.1007/978-3-642-15375-4_33.

26 Michael Aaron Nielsen and Isaac L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, 2010. doi:10.1017/CBO9780511976667.

27 Anna Philippou and Kyriaki Psara. Reversible computation in petri nets. In Jarkko Kari and Irek Ulidowski, editors, *Reversible Computation - 10th International Conference, RC 2018, Leicester, UK, September 12-14, 2018, Proceedings*, volume 11106 of *Lecture Notes in Computer Science*, pages 84–101. Springer, 2018. doi:10.1007/978-3-319-99498-7_6.

28 Iain Phillips and Irek Ulidowski. Reversing algebraic process calculi. In Luca Aceto and Anna Ingólfsdóttir, editors, *FoSSaCS*, volume 3921 of *Lecture Notes in Computer Science*, pages 246–260. Springer, 2006. doi:10.1007/11690634_17.

29 Iain Phillips and Irek Ulidowski. Reversibility and models for concurrency. *Electronic Notes in Theoretical Computer Science*, 192(1):93–108, 2007. doi:10.1016/j.entcs.2007.08.018.

30 Iain Phillips and Irek Ulidowski. Reversing algebraic process calculi. *The Journal of Logic and Algebraic Programming*, 73(1-2):70–96, 2007. doi:10.1016/j.jlap.2006.11.002.

31 Iain Phillips and Irek Ulidowski. Event identifier logic. *Mathematical Structures in Computer Science*, 24(2), 2014. doi:10.1017/S0960129513000510.

32 Alexander Rabinovich and Boris Avraamovich Trakhtenbrot. Behavior structures and nets. *Fundamenta Informaticae*, 11(4):357–404, 1988.

33 Vladimiro Sassone, Mogens Nielsen, and Glynn Winskel. Models for concurrency: Towards a classification. *Theoretical Computer Science*, 170(1-2):297–348, 1996. doi:10.1016/S0304-3975(96)80710-9.

34 Rob J. van Glabbeek and Ursula Goltz. Refinement of actions in causality based models. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *Proceedings REX Workshop on Stepwise Refinement of Distributed Systems*, volume 430 of *Lecture Notes in Computer Science*, pages 267–300. Springer, 1989. doi:10.1007/3-540-52559-9_68.

35 Robert J. van Glabbeek. History preserving process graphs. Technical report, Stanford University, 1996. URL: http://kilby.stanford.edu/~rvg/pub/history.draft.dvi.

36 Robert J. van Glabbeek and Ursula Goltz. Equivalence notions for concurrent systems and refinement of actions (extended abstract). In Antoni Kreczmar and Grazyna Mirkowska, editors, *MFCS*, volume 379 of *Lecture Notes in Computer Science*, pages 237–248. Springer, 1989. doi:10.1007/3-540-51486-4_71.

37 Robert J. van Glabbeek and Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37(4/5):229–327, 2001. doi:10.1007/s002360000041.

38 Robert J. van Glabbeek and Gordon D. Plotkin. Configuration structures, event structures and petri nets. *Theoretical Computer Science*, 410(41):4111–4159, 2009. doi:10.1016/j.tcs.2009.06.014.

39 Glynn Winskel. Event structure semantics for CCS and related languages. In Mogens Nielsen and Erik Meineche Schmidt, editors, *ICALP*, volume 140 of *Lecture Notes in Computer Science*, pages 561–576. Springer, 1982. doi:10.1007/BFb0012800.

40 Glynn Winskel. Event structures. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, 8.-19. September 1986*, volume 255 of *Lecture Notes in Computer Science*, pages 325–392. Springer, 1986. doi:10.1007/3-540-17906-2_31.

41 Glynn Winskel. Event structures, stable families and concurrent games. Lecture notes, University of Cambridge, 2017. URL: https://www.cl.cam.ac.uk/~gw104/ecsym-notes.pdf.
42 Glynn Winskel and Mogens Nielsen. Models for concurrency. In Samson Abramsky, Dov M. Gabbay, and Thomas Stephen Edward Maibaum, editors, *Semantic Modelling*, volume 4 of *Handbook of Logic in Computer Science*, pages 1–148. Oxford University Press, 1995.

A Event Structures as Categories

Configuration structures often use the insights provided by the categorical framework [19, 33, 39, 42]. This appendix regroups the categorical treatment of (identified) configuration structures (Definition 1).

▶ **Definition 32** (Category of configuration structures). *We define $\mathbb{C}$ the category of configuration structures, where an object is a configuration structure, and a morphism $f : \mathcal{C}_1 \to \mathcal{C}_2$ is a triple (f_E, f_L, f_C) such that*
- *$f_L : L_1 \to L_2$;*
- *$f_E : E_1 \to E_2$ preserves labels: $\ell_2(f_E(e)) = f_L(\ell_1(e))$;*
- *$f_C : C_1 \to C_2$ is defined as $f_C(x) = \{f_E(e) \mid e \in x\}$.*

We write $\mathcal{C}_1 \cong \mathcal{C}_2$ if there exists an isomorphism between $\mathcal{C}_1$ and $\mathcal{C}_2$.

For simplicity, we often assume that $L_1 = L_2$, i.e., that all the configuration structures use the same set of labels, take f_L to be the identity and remove it from the notation.

▶ **Definition 33** (Category of $\mathcal{I}$-structures). *We define $\mathbb{D}$ the category of identified configuration structures, where objects are $\mathcal{I}$-structures, and a morphism $f : \mathcal{I}_1 \to \mathcal{I}_2$ is a tuple $q = (f, f_m)$ such that*
- *$f = (f_E, f_C)$ is a morphism in $\mathbb{C}$ between the underlying structures of $\mathcal{I}_1$ and $\mathcal{I}_2$,*
- *$f_m : I_1 \to I_2$ preserves identifiers: $f_m(m_1(e)) = m_2(f_E(e))$.*

We write $\mathcal{I}_1 \cong \mathcal{I}_2$ if there exists an isomorphism between $\mathcal{I}_1$ and $\mathcal{I}_2$.

Observe that $\mathbb{C}$ is a subcategory of $\mathbb{D}$. In both $\mathbb{C}$ and $\mathbb{D}$, composition is written $\circ$ and defined componentwise.

▶ **Lemma 34.** *Identified configuration structures and their morphisms form a category.*

Unsurprisingly, a forgetful functor and an enrichment functor can be defined between those two categories. The only assumption is that we need to suppose that every configuration structure can be endowed with a total ordering $\preceq$ on its events.

▶ **Lemma 35.** *The forgetful functor $\mathcal{F} : \mathbb{D} \to \mathbb{C}$, defined by*
- *$\mathcal{F}(E, C, \ell, I, m) = (E, C, \ell)$*
- *$\mathcal{F}(f_E, f_C, f_m) = (f_E, f_C)$*

and the enrichment functor $\mathcal{S} : \mathbb{C} \to \mathbb{D}$, defined by
- *$\mathcal{S}(E, C, \ell) = (E, C, \ell, I, m)$, where $I = \{1, \dots, |E|\}$ for $|E|$ the cardinality of E, and*

$$m(e) = \begin{cases} 1 & \text{if } \forall e', e \preceq e' \\ i+1 & \text{if } \exists e', e' \preceq e,\ m(e') = i \text{ and there is no } e'' \text{ s.t. } e' \preceq e'' \preceq e \end{cases}$$

- *For $(f_E, f_C) : (E_1, C_1, \ell_1) \to (E_2, C_2, \ell_2)$, $\mathcal{S}(f_E, f_C) = (f_E, f_C, f_m)$, where we let $f_m(m_1(e)) = m_2(f_E(e_2))$.*

are functors.

▶ Remark 36. In $\mathbb{D}$, every morphism $f = (f_E, f_L, f_C, f_m)$ from $\mathcal{I}_1$ to $\mathcal{I}_2$ is actually fully determined by f_E whenever $f_L = \mathrm{id}$. Indeed, given $f_E : E_1 \to E_2$, then we can define for all $x \in C_1$, $f_C(x) = \{f_E(e) \mid e \in x\}$ and f_m as $f_m(m_1(e)) = m_2(f_E(e))$. We will often make the abuse of notation of writing f_E for f and reciprocally.

B Concurrency in a Trace and Trace Equivalence

We give here a quick reminder on concurrency and causality in CCS [8] and RCCS [11] traces. Aside from the convenient notation $m_{R/S}$ that represents the memory stack(s) modified by a forward transition from R to S, and of the notation $\overrightarrow{a}$ for a list of names $a_1, \cdots, a_n$, nothing new is introduced in this Section. However, the results reminded below are used in the proofs of Lemma 19 and Theorem 26.

Concurrency on events corresponds to a notion of concurrency on transitions in RCCS traces [11, Definition 7 and Lemma 8]. For this reminder we consider only concurrency and causality for forward transitions, so that CCS intuitions work equally well. We make a remark at the end about extending the concurrency to backward transitions, but it should be noted that forward and backward transitions are not mixed.

Two transitions $t_1 = R \xrightarrow{i_1:\alpha_1} R_1$ and $t_2 = R' \xrightarrow{i_2:\alpha_2} R_2$ are *composable* if $R_1 = R'$, and in this case, doing t_1 then t_2 is written as the composition $t_1; t_2$. Given n composable transitions $t_i : R_i \xrightarrow{i:\alpha_i} R_{i+1}$ and their composition $t_1; \ldots; t_n$, we say that t_i *is a direct cause of* t_k for $1 \leqslant i < k \leqslant n$ and write $t_i < t_k$ (or, for short, $i < k$) if there is a memory stack m in R_{i+1} and a memory stack m' in R_{k+1} such that $m < m'$, where the order on memory stacks is given by prefix ordering. Note that, if they exist, m and m' are unique, as memory events in reachable processes all have a different pairs (identifier, label).

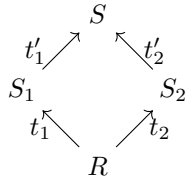
Let $R \xrightarrow{i:\alpha} S$ be a transition. If $\alpha \neq \tau$, we write $m_{R/S} = \{m\}$ where

$$R = (\cdots((R_3 \mid ((R_1 \mid (m \rhd P)) \mid R_2)\backslash\overrightarrow{b^1}) \mid R_4)\backslash\overrightarrow{b^2} \cdots \mid R_n)\backslash\overrightarrow{b^m}$$
$$S = (\cdots((R_3 \mid ((R_1 \mid \langle i, a, Q\rangle.m \rhd P) \mid R_2)\backslash\overrightarrow{b^1}) \mid R_4)\backslash\overrightarrow{b^2} \cdots \mid R_n)\backslash\overrightarrow{b^m}$$

for some R_i any of which could be missing and for some $\overrightarrow{b^j}$, possibly missing as well. If $\alpha = \tau$, then $m_{R/S}$ will contain the pair of memory stacks that has been changed by the transition. Intuitively, the notation $m_{R/S}$ is useful to extract the memory stack(s) modified by a forward transition from R to S.

Two transitions are *coinitial* if they have the same source process and *cofinal* if they have the same target process. We say that two coinitial transitions $t_1 = R \xrightarrow{i_1:\alpha_1} S_1$ and $t_2 = R \xrightarrow{i_2:\alpha_2} S_2$ are *concurrent* if $m_{R/S_1} \cap m_{R/S_2} = \emptyset$, that is, if the transitions modify disjoint memories in R.

The square lemma [11, Lemma 8] says that moreover, given two such concurrent transitions, there exists two cofinal and concurrent transitions $t'_1 = S_1 \xrightarrow{i_2:\alpha_2} S$ and $t'_2 = S_2 \xrightarrow{i_1:\alpha_1} S$. The name of the lemma comes from this picture:



Moreover, the traces $\theta_1 = t_1; t'_1$ and $\theta_2 = t_2; t'_2$ are *equivalent* [11, Definition 9]. This allows one to define *equivalence classes on transitions*: t_1 in θ_1 is equivalent to t'_2 in θ_2 if θ_1 is equivalent to θ_2 and t_1 and t'_2 have the same index. Then in the trace $t_1; t'_1$ we are now allowed to say that t_1 is concurrent to t'_1.

In a trace $t_1; t_2$ we have that t_1 is concurrent to t_2 iff t_1 is not a cause of t_2. This follows from a case analysis using the definitions of concurrency and causality. Thanks to trace equivalence, we also have that in a trace $t_1; \ldots; t_n$ either t_1 is a cause of t_n or the two transitions are concurrent. Those intuitions are enough for us to carry on our development, but a complete treatment of concurrency and causality in the trace of a CCS process [8] can give better insight to the curious reader.

The definitions of concurrency for forward coinitial traces and of causality for forward traces can easily be "flipped" into definitions of concurrency for backward cofinal traces, and of causality for backward traces.

C Auxiliary Materials

In this section we introduce some intermediate definitions and lemmas that are necessary for the proofs, the details of which can be found at `https://hal.archives-ouvertes.fr/hal-02568250`.

C.1 Operations on Identified Configurations Structures (Sect. 2.1)

Our main goal here is to state that the operations of Definition 5 preserve $\mathcal{I}$-structures (Lemma 40), and to give some intuitions about them. The product and coproduct (used to define the nondeterministic choice below) have particular roles, since they have a direct representation in the categorical world (Lemma 38).

The structures we considered are *full* w.r.t. the sets of labels and identifiers, i.e. the labeling and identifying functions are surjective. This only impacts the relabelling and reidentifying operations, where we have to additionally require that ℓ' and m' are surjective.

We redefine the nondeterministic choice of Definition 5 by first defining the coproduct on $\mathcal{I}$-structures and then using relabeling and reidentifying to get rid of the extra indices in the label and the identifier of events. It is easy to check that the two definitions of nondeterministic choice are equivalent, but working with the one from Definition 5 is easier and simpler.

▶ **Definition 37.** *We redefine nondeterministic choice using coproduct as follows:*

The coproduct *of $\mathcal{I}_1$ and $\mathcal{I}_2$ is $\mathcal{I}_1 \pm \mathcal{I}_2 = (E, C, L, \ell, I, m)$, where*

- $E = \{\{1\} \times E_1\} \cup \{\{2\} \times E_2\}$ *with* $\pi_1 : E \to \{1, 2\}$ *and* $\pi_2 : E \to E_1 \cup E_2$,
- $C = \{\{i\} \times x \mid x \in C_i\}$,
- $L = \{\{1\} \times L_1\} \cup \{\{2\} \times L_2\}$,
- $\ell(e) = (i, \ell_i(\pi_2(e)))$ *for* $\pi_1(e) = i$,
- $I = \{\{1\} \times I_1\} \cup \{\{2\} \times I_2\}$,
- $m(e) = (i, m_i(\pi_2(e)))$ *for* $\pi_1(e) = i$.

and with the expected injections $\iota_i : \mathcal{I}_i \to \mathcal{I}_1 + \mathcal{I}_2$.

The nondeterministic choice *of $\mathcal{I}_1$ and $\mathcal{I}_2$ is $\mathcal{I}_1 + \mathcal{I}_2 = (\mathcal{I}_1 + \mathcal{I}_2)[\ell'/\ell][m'/m]$ where*

- $\ell'(e) = a$ *if* $\ell(e) = (j, a), j \in \{1, 2\}$,
- $m'(e) = i$ *if* $\ell(e) = (j, i), j \in \{1, 2\}$.

▶ **Lemma 38.** *The product and coproduct of $\mathcal{I}$-structures is the product and coproduct in $\mathbb{D}$.*

In the categorical setting, the product and coproduct on labeled configuration structures can be obtained by a straightforward enrichment of un-labeled configuration structures [42, Propositions 11.2.2 and 11.2.3]. In a similar vein, we obtain the extension of those operations on identified (labeled) configuration structures directly.

The restriction of the operations of Definition 5 to configuration structures are standard [39, 40], except for postfixing. We state below that the restriction of this operation to configuration structures is correct.

▶ **Lemma 39.** *The postfixing of a label a to an event structure $\mathcal{C}_1 = (E_1, C_1, L_1, \ell_1)$, defined as $\mathcal{C}_1 :: (a) = (E, C, L, \ell)$ where*

- $E = E_1 \cup \{e\}$, *for* $e \notin E_1$,
- $L = L_1 \cup \{a\}$,
- $C = C_1 \cup \{x \cup \{e\} \mid x \in C_1$ *is maximal and finite*$\}$,
- $\ell = \ell_1 \cup \{e \mapsto a\}$,

is a configuration structure.

▶ **Lemma 40.** *The operations of Definition 5 (relabeling, reidentifiying, restriction, prefixing, postfixing, non-deterministic choice and product), coproduct, as well as the parallel composition (Definition 7) preserve $\mathcal{I}$-structures.*

C.2 Properties of Memory Encodings and Operational Correspondence

Our goal here is to give some intuition as to how to prove that that there is an operational correspondence between R and $\lceil R \rceil$ (Lemma 19) by stating intermediate lemmas (Lemmas 43–45) about the encoding of memories and their relation to maximal events. Those lemmas, in turn, requires some useful properties of memory encoding (Sect. C.2.1).

C.2.1 Properties of Memory Encodings

We assume given reachable processes R and S and we write O_R for the origin of R, and $\lceil R \rceil$ as $(E_R, C_R, \ell_R, I_R, m_R)$ and similarly for S. To prove interesting properties about the encoding of memory, we first need this small technical lemma.

▶ **Lemma 41.** *For every reversible thread $m \rhd P$ of a reachable process R, and for all $i \in \mathsf{I}(m)$, i occurs once in m.*

Note that the property above holds for reversible *threads*, and not for RCCS *processes* in general: we actually *want* memory events to sometimes share the same identifiers. Indeed, two memory events need to have the same identifiers if they result from a synchronization (i.e., the application of the syn. rule of Fig. 2) or a fork (i.e., the application of the Distribution of Memory rule of structural equivalence, Definition 9).

▶ **Lemma 42** (Uniqueness of identifiers)**.** *For all $e_1, e_2 \in E_R$, $m_R(e_1) = m_R(e_2)$ implies $e_1 = e_2$.*

▶ **Lemma 17** (Memories give posets)**.** *For all R, letting x be the maximal configuration in $\lceil R \rceil$ (Definition 2), $(\lceil R \rceil, \subseteq)$ is a partially ordered set (poset) with maximal element x.*

C.2.2 Operational Correspondence

▶ **Lemma 43.** *If $R \equiv S$, then the exists an isomorphism f between $\lceil R \rceil$ and $\lceil S \rceil$, with $f_L = \mathrm{id}$ and $f_m = \mathrm{id}$.*

▶ **Lemma 44.** *The event introduced in the postfixing of a memory event to an identified structure is maximal in the resulting identified structure.*

Furthermore, the maximality of an event can be "preserved" by parallel composition:

▶ **Lemma 45.** *For all identified structure $\mathcal{I}_1 = (E_1, C_1, L_1, \ell_1, I_1, m_1)$ with $e_1 \in E_1$ a maximal event in it, and for all identified configuration $\mathcal{I}_2 = (E_2, C_2, L_2, \ell_2, I_2, m_2)$ such that $m_1(e_1) \notin I_2$, $(e_1, \star)$ is maximal in $\mathcal{I}_1 \mid \mathcal{I}_2$.*

And then Lemmas 43–45 give all the required tools to prove the operational correspondence of Lemma 19.

C.3 Connecting Formalisms to Obtain Our Main Results

Our goal here is to give the tools and intuitions needed to prove Theorems 29 and 30, which give as an immediate corollary our main result (Corollary 31) and an interesting remark (Corollary 49). Our main task will be to identify isomorphisms of $\mathcal{I}$-structures with l&o-p functions (Lemma 46) and then to connect our previous formalism with the encoding of memories (Lemma 48). This connection coupled to the operational correspondence detailed in Sect. 2.5 makes it possible to prove our two main theorems.

We first establish a connection between isomorphisms (in category theory) and l&o-p bijections (that are used to define (H)HPB). Then, we construct a bridge between $\lfloor R \rfloor$ and our previous formalism [2].

▶ **Lemma 46.** *Letting x_1^m and x_2^m be the unique maximal configurations in $\lceil R_1 \rceil$ and $\lceil R_2 \rceil$, there is an isomorphism f between $\lceil R_1 \rceil$ and $\lceil R_2 \rceil$ with $f_L = \mathrm{id}$ iff there exists a l&o-p bijection between x_1^m and x_2^m.*

For the reader familiar with event structures, a configuration x defines an event structure $(x, \leqslant_x, \ell)$. The construction below mirrors the transformation from an event structure to a configuration structure [42].

▶ **Definition 47** (Generation of a $\mathcal{I}$-structure from a configuration)**.** *Given $\mathcal{I} = (E, C, L, \ell, I, m)$, for $x \in C$, the* $\mathcal{I}$-structure generated by *x is $x{\downarrow} = (x, \{y \mid y \in C, y \subseteq x\}, \{a \mid \exists e \in x_R, \ell(e) = a\}, \ell{\restriction_x}, \{i \mid \exists e \in x_R, m(e) = i\}, m{\restriction_x})$.*

▶ **Lemma 48.** *The $\mathcal{I}$-structure $\lceil R \rceil$ is isomorphic to $x_R{\downarrow}$, where $(\llbracket O_R \rrbracket, x_R)$ is the encoding previously defined [2].*

▶ **Corollary 49.** *The relation obtained by considering only (5–6) in the definition of* B&F *(Definition 24) is equal to* HPB *on CCS terms (Definition 28).*

A Near-Linear-Time Algorithm for Weak Bisimilarity on Markov Chains

David N. Jansen
State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, Beijing, China
dnjansen@ios.ac.cn

Jan Friso Groote
Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands
J.F.Groote@tue.nl

Ferry Timmers
Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands
F.Timmers@tue.nl

Pengfei Yang
State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, Beijing, China
University of Chinese Academy of Sciences, Beijing, China
yangpf@ios.ac.cn

Abstract

This article improves the time bound for calculating the weak/branching bisimulation minimisation quotient on state-labelled discrete-time Markov chains from $O(mn)$ to an expected-time $O(m \log^4 n)$, where n is the number of states and m the number of transitions. For these results we assume that the set of state labels AP is small ($|AP| \in O(m/n \log^4 n)$). It follows the ideas of Groote et al. (ACM ToCL 2017) in combination with an efficient algorithm to handle decremental strongly connected components (Bernstein et al., STOC 2019).

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains; Theory of computation → Formal languages and automata theory; Theory of computation → Probabilistic computation; Software and its engineering → Formal software verification

Keywords and phrases Behavioural Equivalence, weak Bisimulation, Markov Chain

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.8

Funding *David N. Jansen*: This research is partly done during a visit to Eindhoven University of Technology, The Netherlands. This author is supported by the National Natural Science Foundation of China, Grants No. 61761136011 and 61532019.
Jan Friso Groote: This research is partly done during a visit to the State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, P.R. China.

1 Introduction

Bisimilarity formalises when two behaviours are equal, where behaviours are given by automata or variants of directed graphs, such as labelled transition systems, Kripke structures or Markov chains. In these fields bisimilarity is also known as the zig-zag relation or lumping. Bisimilarity is an equivalence relation that preserves all core properties of behaviour. Moreover, calculating the bisimilarity quotient of a behaviour can lead to substantially smaller graphs. This is particularly useful when analysing the behaviour either by visual inspection or using other tools.

Further reduction of the behaviour is possible by considering some or all of the edges as silent steps, steps that cannot be observed directly. Milner referred to such steps as τ-actions [18]. Bisimulation equivalences that take such internal steps into account are weak bisimilarity [18], branching bisimilarity [9], stuttering equivalence [5], and weak/branching bisimulation on fully probabilistic systems [1].

In this article we are interested in weak behavioural equivalences for discrete-time Markov chains, similar to those introduced by [1]. There it was shown that branching and weak bisimilarity are equal notions on fully probabilistic systems, and an $O(mn)$ algorithm was given to calculate the weak/branching bisimilarity quotient, where m is the number of transitions and n is the number of states. In this paper we substantially improve upon this by providing an expected-time $O(m \log^4 n)$ algorithm, which is nearly linear in the number of transitions m, to calculate the weak/branching bisimilarity quotient on Markov chains.

The algorithm is an intricate combination of a number of rather different ideas stemming from various algorithms.

- The first idea is to use the principle "Process the smaller half" of Hopcroft [13] in the setting of probabilistic processes as in [7, 12, 20]. This means that whenever a state is revisited in an algorithm, its context is at least reduced by half compared to the previous visit. For n states this then means that each state is processed at most $O(\log n)$ times. This leads to $O(m \log n)$ algorithms to calculate (strong) bisimilarity on graphs. In our algorithm this is reflected in the use of two partitions of states, one containing constellations and a finer (or equal) partition containing blocks. The blocks are the context of a state that are reduced by a factor 2 for each visit.
- As weak and branching bisimilarity coincide on Markov chains, we can use the ideas from the algorithms for branching bisimulation minimisation. The first idea stems from [11]: detecting whether blocks need to be split can be done as efficiently as in strong bisimulation by only looking at *bottom states,* states without outgoing silent steps. The second idea comes from [10, 14]: the actual splitting of blocks can be done in time proportional to the smaller resulting subblock, guaranteeing that each state is visited at most $O(\log n)$ times in a split, according to the principle "Process the smaller half". This is achieved by simultaneously extending the markings and non-markings of bottom states to all other states in a block, and stop when the first of these two processes finishes.

  We introduce the notion of *$\mathcal{C}$-silent states,* which are states of which all outgoing transitions lead to the constellation containing the state itself. The role of the bottom states can now be played by non-$\mathcal{C}$-silent states. A block is only splittable if it has non-$\mathcal{C}$-silent states marked with different probabilities to go to a splitter block. These probabilities are then extended to the $\mathcal{C}$-silent states in the block in time proportional to the sizes of the smallest resulting subblocks.
- For bottom states and the simultaneous extension of the (non-)markings, it is essential that strongly connected components (SCCs) of inert steps can be contracted to a single state in the behavioural graph. The algorithms in [11, 10, 14] preprocess the graph accordingly. Unfortunately, it is not possible to contract inert SCCs in Markov chains, as this does not preserve weak bisimilarity. This is caused by the fact that often, the probability to leave an SCC through a specific edge is different for every state in the SCC, which means that states within SCCs are not necessarily weakly bisimilar.

  To extend the markings and non-markings, we need to know what the $\mathcal{C}$-silent SCCs are, i.e., the SCCs restricted to the $\mathcal{C}$-silent states in each block. This would not be a problem if the SCCs were static throughout each run of the algorithm, as they could be determined in time $O(m)$ as a preprocessing step. But whenever a constellation is split, the SCCs

change, as more states become non-$\mathcal{C}$-silent. It is not an option to recalculate the SCCs each time a state becomes non-$\mathcal{C}$-silent and is moved, as done in [21] for orthogonal bisimulation, as this would lead to a runtime of $O(mn)$.

Fortunately, we can use a recent result showing that SCCs can be maintained under the removal of edges within an expected $O(m \log^4 n)$ time [4]. Whenever splitting a constellation, states become non-$\mathcal{C}$-silent, and in particular their transitions are removed from the SCCs. The algorithm of [4] maintains the SCCs using the above complexity, allowing at any moment during the run of our algorithm to determine in constant time which states are in the same SCC, and that is exactly what we require.

The result is the first algorithm with an expected near-linear time complexity and a linear space complexity for the reduction of weak/branching bisimilarity of Markov chains.

Note that contrary to the previous less efficient algorithms we can only guarantee an *expected* runtime. The reason for this is deeply embedded in the algorithm of [4] to maintain decremental SCCs. The SCCs are constructed using a generalization of so-called ES-trees [8]. When SCCs fall apart, the ES-trees have to be recalculated, except for those SCCs which contain the roots of the old ES-trees. If such roots are chosen uniformly at random, there is a higher probability that the roots are in the larger ES-trees, and the work to recalculate them falls within an expected "Process the smaller half" regime. This is the only place where the algorithm is randomized. For the remainder it is completely deterministic.

The structure of this article is as follows. In Section 2 the required preliminaries are explained. In Section 3 the algorithm is outlined. The details, correctness and complexity are presented in Section 4, followed by a conclusion.

2 Preliminaries

We consider finite discrete-time Markov chains in the line of [1, 3]. In order to distinguish states, we allow for a state labelling with atomic propositions from a set AP.

▶ **Definition 1.** *A* discrete-time Markov chain *(DTMC) is a quadruple* $\mathcal{M} = (S, AP, \mathbf{P}, L)$ *where:*

- S *is a finite set of states. We write* n *for the number of states.*
- AP *is a finite set of atomic propositions.*
- $\mathbf{P} : S \times S \to [0, 1]$ *is a probability matrix satisfying* $\sum_{t \in S} \mathbf{P}(s, t) = 1$ *for all* $s \in S$*. We write* m *for the number of non-zero entries in* $\mathbf{P}$*.*
- $L : S \to 2^{AP}$ *is a labelling function, which assigns to each state* $s \in S$ *the set* $L(s)$ *of atomic propositions that are valid in* s*.*

In a DTMC $\mathcal{M} = (S, AP, \mathbf{P}, L)$, the transition probability function $\mathbf{P}(s, t)$ intuitively gives the probability of a state s going to t in a single step. For $s \in S$ and $A \subseteq S$, we define $\mathbf{P}(s, A) := \sum_{t \in A} \mathbf{P}(s, t)$ to be the probability of a state s entering A in a single step. If $\mathbf{P}(s, t) > 0$, we sometimes write $s \to t$ if the numerical value of the probability is irrelevant. We define the sets of incoming transitions $in(A) = \{s \to t \mid s \notin A \wedge t \in A\}$ and outgoing transitions $out(A) = \{s \to t \mid s \in A \wedge t \notin A\}$ for $A \subseteq S$. We assume the set AP to be small. For the complexity results we concretely require that $|AP| \in O(m/n \log^4 n)$.

As a side note we observe that it is easy to accommodate subprobabilistic DTMCs (i.e., to allow $\mathbf{P}$ to be a subprobability matrix satisfying $0 \leq \mathbf{P}(s, S) \leq 1$). In that case, one adds a pseudostate $\perp \notin S$ and defines $\mathbf{P}(s, \perp) = 1 - \mathbf{P}(s, S)$ and $\mathbf{P}(\perp, \perp) = 1$. Also, $\perp$ is separated from normal states by an atomic proposition: $L(\perp) = \{\mathsf{pseudo}\}$ for $\mathsf{pseudo} \notin AP$. Then, $(S \cup \{\perp\}, AP \cup \{\mathsf{pseudo}\}, \mathbf{P}, L)$ is a fully probabilistic DTMC.

A *path* in a DTMC is an infinite sequence of states $\pi = (s_0, s_1, s_2, \ldots)$ with $s_{i-1} \to s_i$ for $i > 0$. A *cylinder set* $Cyl(s_0, s_1, \ldots, s_n)$ is the set of paths that start with the sequence $(s_0, s_1, \ldots, s_n)$. Given an initial probability distribution μ on states, each cylinder set is assigned a probability: $Pr_\mu(Cyl(s_0, s_1, \ldots, s_n)) = \mu(s_0) \prod_{i=1}^{n} \mathbf{P}(s_{i-1}, s_i)$. The probability space of a DTMC can be defined as the unique extension of this content Pr_μ to the σ-algebra generated by the cylinder sets; details can be found in [16, Chapters 2, 4]. For a state s we write the Dirac distribution on s as $\delta(s)$, i.e., $\delta(s)(s) = 1$ and $\delta(s)(t) = 0$ for $t \neq s$. We denote the probability to take a path to a state in $C \subseteq S$ through states in $B \subseteq S$:

$$Pr(s, B, C) := \sum_{s_0,\ldots,s_{n-1} \in B \setminus C,\, s_n \in C} Pr_{\delta(s)}\left(Cyl(s_0, s_1, \ldots, s_n)\right).$$

Note that if $s \in C$, we have $Pr(s, B, C) = 1$ for any B; but if $s \notin B \cup C$, then $Pr(s, B, C) = 0$.

The presented algorithm is based on refining partitions of finite sets of states. For any set S, a *partition* of S is a set $\mathcal{B} = \{B_i \subseteq S \mid i \in I\}$ satisfying $\emptyset \notin \mathcal{B}$, $B_i \cap B_j = \emptyset$ whenever $i \neq j$, and $\bigcup \mathcal{B} = S$. We call each B_i a *block.*

For two partitions $\mathcal{B}_1 \neq \mathcal{B}_2$ of S, we say $\mathcal{B}_1$ is *finer* than $\mathcal{B}_2$, or that $\mathcal{B}_2$ is *coarser* than $\mathcal{B}_1$, iff for every block $B_1 \in \mathcal{B}_1$, there is a block $B_2 \in \mathcal{B}_2$, such that $B_1 \subseteq B_2$.

Given a partition $\mathcal{B}$, we denote the block containing state s by $[s]_\mathcal{B}$. If a set of states $B' \subseteq B$ for some block B in $\mathcal{B}$, we also write $[B']_\mathcal{B}$ for B being the block in which B' is contained. Every partition $\mathcal{B}$ induces an equivalence relation, also denoted $\mathcal{B}$, defined by $s \,\mathcal{B}\, t$ iff $s \in [t]_\mathcal{B}$. Conversely, every equivalence relation R on S has a unique corresponding partition $\{\{t \mid t\ R\ s\} \mid s \in S\}$. This partition is denoted as S/R. We denote by $[s]_R$ the R-equivalence class of $s \in S$, i.e., $[s]_R = \{t \mid t\ R\ s\}$.

Given a DTMC $\mathcal{M} = (S, AP, \mathbf{P}, L)$, we define AP-equivalence to be the relation that distinguishes states based on their labels: $s \equiv_{AP} t$ iff $L(s) = L(t)$. Its equivalence class for s is denoted $[s]_{AP}$.

▶ **Definition 2.** *Let $\mathcal{M} = (S, AP, \mathbf{P}, L)$ be a DTMC and $R \subseteq S \times S$ an equivalence relation. A state s is* R-silent *iff $\mathbf{P}(s, [s]_R) = 1$. A transition $s \to t$ is* R-inert *iff $s\ R\ t$.*

If the equivalence relation R is given by a partition $\mathcal{B}$ of states, we also speak about a $\mathcal{B}$-silent state. For non-R-silent states, we define the conditional probability to enter some set of states in a single step under the condition to leave the R-equivalence class:

$$\mathbf{P}(s, B \mid \text{non-}R\text{-inert}) := \frac{\mathbf{P}(s, B)}{1 - \mathbf{P}(s, [s]_R)}.$$

We are now ready to introduce the notions of weak and branching bisimilarity.

▶ **Definition 3.** *Let $\mathcal{M} = (S, AP, \mathbf{P}, L)$ be a DTMC and $R \subseteq \equiv_{AP}$ an equivalence relation on S (which respects the atomic propositions of states). We say that R is*

a weak bisimulation *iff $s\ R\ t$ implies, for all R-equivalence classes $C \in S/R$ with $s, t \notin C$, that $Pr(s, [s]_{AP}, C) = Pr(t, [t]_{AP}, C)$.*

a branching bisimulation *iff $s\ R\ t$ implies, for all R-equivalence classes $C \in S/R$ with $s, t \notin C$, that $Pr(s, [s]_R, C) = Pr(t, [t]_R, C)$.*

a conditional-probability bisimulation *iff $s\ R\ t$ implies*

1. *If s and t are both non-R-silent, for all R-equivalence classes $C \in S/R$ with $s, t \notin C$, we have $\mathbf{P}(s, C \mid \text{non-}R\text{-inert}) = \mathbf{P}(t, C \mid \text{non-}R\text{-inert})$; and*
2. *s has a path to a state outside $[s]_R$ iff t has a path to a state outside $[t]_R$.*

The states s and t are weakly bisimilar, *denoted $s \approx_w t$, iff a weak bisimulation R exists such that $s\ R\ t$. Similarly, s and t are* branching bisimilar, *denoted $s \approx_b t$, iff a branching bisimulation R exists such that $s\ R\ t$. Finally, s and t are* conditional-probability-bisimilar, *denoted $s \approx_c t$, iff a conditional-probability bisimulation R exists such that $s\ R\ t$.*

All three relations $\approx_w$, $\approx_b$ and $\approx_c$ are equivalence relations. The essential difference between branching and weak bisimulation is that in branching bisimulation a step from state s must be mimicked by a number of steps through the equivalence class of s, whereas in weak bisimulation a step can be mimicked by steps through states labelled with the same propositions. For nondeterministic transition systems branching bisimilarity implies weak bisimilarity but not vice versa. Furthermore, deciding branching bisimilarity is more efficient. Remarkably, in the context of Markov chains, the notions are equal:

▶ **Proposition 4.** *Let $\mathcal{M} = (S, AP, \mathbf{P}, L)$ be a DTMC. States in S are weakly bisimilar iff they are branching bisimilar iff they are conditional-probability-bisimilar.*

Proof. [1, 2] prove this result for fully probabilistic systems (with action labels instead of atomic propositions). See the Appendix for an adaptation of the proof to DTMCs. ◀

Due to this proposition, we only write "weak bisimulation" in the remainder of this paper. Because conditional-probability bisimulation can be tested by looking at single-step probabilities only, we use its conditions to calculate the weak bisimilarity quotient without having to calculate a transitive closure.

If we allowed DTMCs with an infinite state space, conditional-probability bisimilarity would not imply weak/branching bisimilarity. See [15] for an example and a possible strengthening of Condition 2 in the definition of conditional-probability bisimulation.

3 Main ideas of the algorithm

Now we state our problem formally:

> Given a DTMC $\mathcal{M} = (S, AP, \mathbf{P}, L)$, we need to compute the weak bisimilarity relation, or equivalently, to give a partition $\mathcal{B}$ of the state space S consisting of the weak bisimilarity equivalence classes.

This section explains the main ideas to solve this problem efficiently.

Partition refinement. Typically, bisimilarity is computed by partition refinement. In our case this starts from an initial partition where states are in the same block iff they have the same atomic propositions and satisfy Condition 2 of conditional-probability bisimulation – note that refinements of a partition preserve these conditions. Then the algorithm checks Condition 1 of conditional-probability bisimulation for every block. If it finds a pair of states in one block with different transition probabilities to another block, it splits the former block, bringing the validity of Condition 1 closer. The latter block is called a *splitter*.

Avoid superfluous refinements. After a splitter $Sp \in \mathcal{B}$ has been used to split all blocks with transitions to Sp, every further refined partition is stable w.r.t. Sp. However, the algorithm of [1] does not register this information and may check whether Sp is a splitter repeatedly. We optimize by registering former splitters. In addition to the current partition $\mathcal{B}$, we store a coarser (or equal) partition $\mathcal{C}$ to record the splitters already used in previous iterations. We call blocks in $\mathcal{C}$ *constellations* and print them in boldface $\mathbf{C} \in \mathcal{C}$. The relation between blocks $\in \mathcal{B}$ and constellations $\in \mathcal{C}$ is described by the main invariant:

▶ **Main Invariant 5.** *If s and t are non-$\mathcal{C}$-silent states in the same block, i.e., $[s]_{\mathcal{B}} = [t]_{\mathcal{B}}$, and $\mathbf{C} \in \mathcal{C}$ is a constellation that does not contain s and t, then*

$$\mathbf{P}(s, \mathbf{C} \mid \text{non-}\mathcal{C}\text{-inert}) = \mathbf{P}(t, \mathbf{C} \mid \text{non-}\mathcal{C}\text{-inert}).$$

This invariant expresses that states in the same block have the same conditional probability to enter every other constellation in a single step. This means that blocks cannot be split by constellations. They can however be split by other blocks.

▶ **Example 6.** The formulation of the main invariant is inspired by conditional-probability bisimulation. The DTMC fragment in Figure 1 shows that a formulation inspired by weak or branching bisimulation does not appear to work.

States s and t have the same conditional probability to enter $\mathbf{C}$ in a single step: $\mathbf{P}(s, \mathbf{C} \mid \text{non-}\mathcal{C}\text{-inert}) = \frac{0.5}{1-0.2-0.3} = 1 = \frac{1}{1-0} = \mathbf{P}(t, \mathbf{C} \mid \text{non-}\mathcal{C}\text{-inert})$. However, because s also has a transition to state u, which is in a different block of the same constellation and has no $\mathcal{C}$-inert path to $\mathbf{C}$, we have $Pr(s, [s]_{\mathcal{C}}, \mathbf{C}) = 0.5 + 0.2 \cdot 1 = 0.7 \neq 1 = Pr(t, [t]_{\mathcal{C}}, \mathbf{C})$, and also $\mathbf{P}(s, \mathbf{C} \mid \text{non-}\mathcal{B}\text{-inert}) = \frac{0.5}{1-0.2} = 0.625 \neq 1 = \frac{1}{1-0} = \mathbf{P}(t, \mathbf{C} \mid \text{non-}\mathcal{B}\text{-inert})$. If $\mathcal{B} = \mathcal{C}$, then no states like u exist, and Main Invariant 5 (together with Condition 2 of conditional-probability bisimulation, which was already ensured by the initial partition) implies weak bisimulation.

Refining constellations in $\mathcal{C}$. As mentioned in the example, we should try to reach the situation that $\mathcal{B} = \mathcal{C}$. Therefore, as long as these partitions are different, we choose a small splitter block $Sp \in \mathcal{B} \setminus \mathcal{C}$. We move Sp to its own constellation, and reestablish Main Invariant 5 by splitting blocks with transitions to Sp. By choosing a small splitter, we ensure that each state takes part in constellation processing only logarithmically often – according to the principle "Process the smaller half".

To register the fact that we have used some Sp as a splitter, we refine its $\mathcal{C}$-equivalence class $[Sp]_{\mathcal{C}}$ into Sp and the rest $[Sp]_{\mathcal{C}} \setminus Sp$. Note that establishing Main Invariant 5 w.r.t. Sp also automatically establishes the invariant w.r.t. $[Sp]_{\mathcal{C}} \setminus Sp$ for most states: As we know $\mathbf{P}(s, [Sp]_{\mathcal{C}} \mid \text{non-}\mathcal{C}\text{-inert}) = \mathbf{P}(t, [Sp]_{\mathcal{C}} \mid \text{non-}\mathcal{C}\text{-inert})$ for states s, t in the same block $\not\subseteq [Sp]_{\mathcal{C}}$, and the algorithm ensures that $\mathbf{P}(s, Sp \mid \text{non-}\mathcal{C}\text{-inert}) = \mathbf{P}(t, Sp \mid \text{non-}\mathcal{C}\text{-inert})$, we get for free that $\mathbf{P}(s, [Sp]_{\mathcal{C}} \setminus Sp \mid \text{non-}\mathcal{C}\text{-inert}) = \mathbf{P}(t, [Sp]_{\mathcal{C}} \setminus Sp \mid \text{non-}\mathcal{C}\text{-inert})$. Only for states in Sp itself we have to additionally check whether they are stable under $[Sp]_{\mathcal{C}} \setminus Sp$.

Refining blocks in $\mathcal{B}$. For every block B with transitions to the selected splitter Sp, we first split its non-$\mathcal{C}$-silent states into subblocks whose conditional probabilities to enter the splitter in a single step are equal. Similarly, in algorithms for branching bisimilarity [11, 10], one splits the bottom states into two subblocks first. But in our case the non-$\mathcal{C}$-silent states may fall apart into more than two subblocks, depending on the (conditional) probability to enter the splitter in a single step.

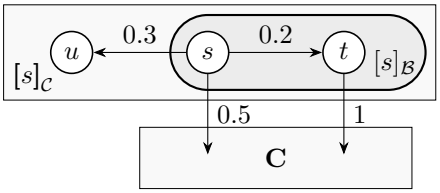


Figure 1 Even if $\mathbf{P}(s, \mathbf{C} \mid \text{non-}\mathcal{C}\text{-inert}) = \mathbf{P}(t, \mathbf{C} \mid \text{non-}\mathcal{C}\text{-inert})$, we still may have $Pr(s, [s]_{\mathcal{C}}, \mathbf{C}) \neq Pr(t, [t]_{\mathcal{C}}, \mathbf{C})$.

The next step is to extend this splitting to the $\mathcal{C}$-silent states in B. Assume for the moment that there are no strongly connected components of $\mathcal{C}$-silent states. This means that the $\mathcal{B}$-inert transitions in B form a dag. The $\mathcal{C}$-silent states that only have paths to a single subblock of non-$\mathcal{C}$-silent states join this subblock. This condition is checked by traversing the transition relation backwards, to find states whose outgoing $\mathcal{B}$-inert transitions all lead to the same subblock. Only after we have investigated all outgoing $\mathcal{B}$-inert transitions of a state, we can extend the splitting to its predecessors. Those $\mathcal{C}$-silent states that have $\mathcal{B}$-inert paths to multiple subblocks move to a special subblock, which we call the *residue*. These paths to other subblocks may have different probabilities for different states, but as the residue will be split further later in the algorithm, this is not a problem.

However, the subblocks have to be extended in a way compatible with the principle "Process the smaller half". This forbids to spend time on a subblock with more than half the states of B. Because we do not know which subblock will end up to be the largest, all subblocks are extended simultaneously, and when a subblock (which can be the residue) becomes too large, we abort extending this subblock. Note that at most one block can be so large. This process of simultaneous backward extension of the subblocks ends if all but one block have been completed. All non-visited states necessarily belong to this non-completed block. The time used for the backward extension can now be attributed completely to the smaller subblocks whose sizes are guaranteed to be at most half that of B.

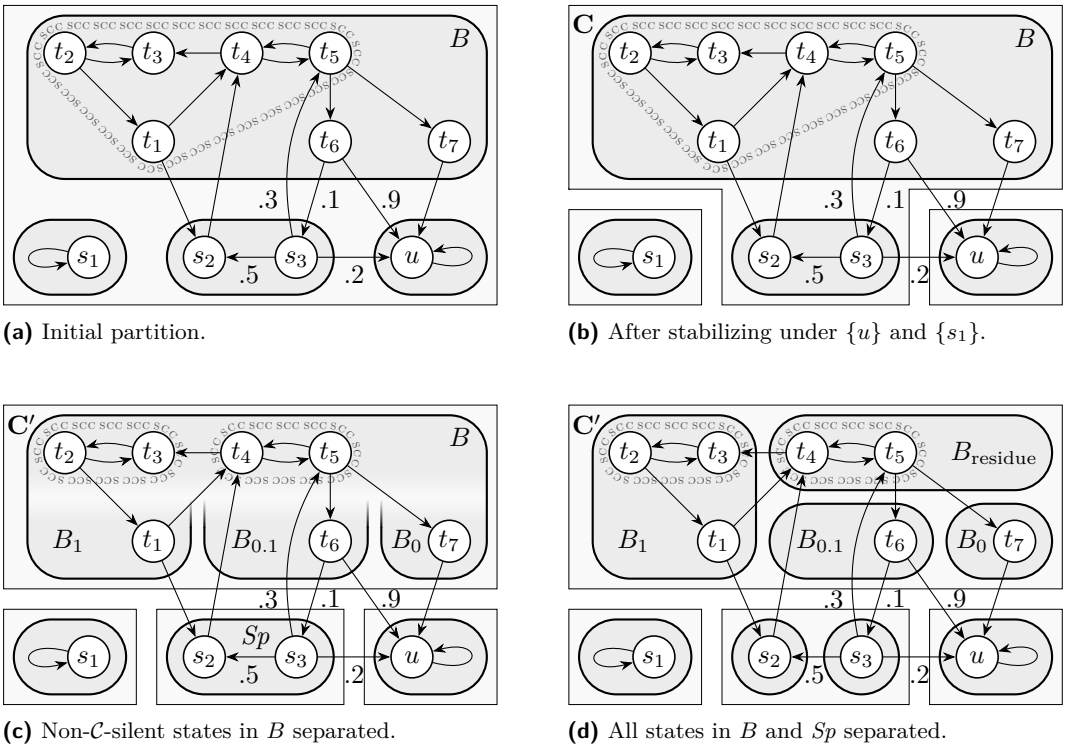
Strongly connected components. Unfortunately, if there are non-trivial SCCs in the $\mathcal{B}$-inert transitions, the backward extension does not work. The reason is that in order to decide whether a state belongs to a subblock or to the residue, all its outgoing $\mathcal{B}$-inert transitions must have been investigated first. In a dag this is guaranteed, but with non-trivial SCCs, this is impossible. Still, all states in such a non-trivial SCC have paths to the same subblocks. It is therefore possible to view this SCC as a single state, and apply the dag-based backward extension of subblocks as described above to the complete SCC.

For this to work we have to keep track of SCCs in the $\mathcal{B}$-inert transitions within the $\mathcal{C}$-silent states of every block. When a constellation is split, some states may become non-$\mathcal{C}$-silent, and we have to dynamically recompute the sub-SCCs within the states that remain $\mathcal{C}$-silent. It is not possible to use a classical linear algorithm for this purpose, as the strongly connected components have to be recomputed for all new blocks, including a potential block that is larger than half of the size of B. Therefore, the "Process the smaller half" strategy cannot be applied, leaving us with an $O(mn)$ algorithm.

Fortunately, recently an efficient algorithm has been presented that can maintain the strongly connected components within a lower time bound [4]. This algorithm initializes a data structure for SCCs in worst-case time complexity $O(m \log^4 n)$ and recomputes all sub-SCCs in total *expected* time complexity $O(m \log^4 n)$.

▶ **Example 7.** Figure 2 illustrates the above ideas for an example DTMC. The state labellings with atomic propositions are not explicitly indicated, but different letters in the state name indicate different labellings. For some transitions the exact probability value is not shown, but it is nonzero, and all outgoing probabilities sum up to 1.

Subfigure 2a shows the initial partition: states with different labelling are in different blocks (shown as rounded, darker grey shapes). Additionally, to satisfy Condition 2 of conditional-probability bisimulation, we have separated s_1 from states s_2 and s_3 as only the latter two have paths to a state outside $\{s_1, s_2, s_3\}$. All states are in one constellation (shown as light grey rectangle).



(a) Initial partition.

(b) After stabilizing under $\{u\}$ and $\{s_1\}$.

(c) Non-$\mathcal{C}$-silent states in B separated.

(d) All states in B and Sp separated.

Figure 2 The first few steps of partition refinement.

Subfigure 2b shows the situation after blocks $\{u\}$ and $\{s_1\}$ have been used as splitters. This is recorded by putting them into their own constellations. However, these two splitters did not lead to actual refinements. State s_2 only has a transition to another block in the same constellation, and when this block becomes a splitter, it will separate s_2 from s_3.

Then, the only remaining small splitter is block $Sp = \{s_2, s_3\}$ in constellation $\mathbf{C}$. Block B, the block with transitions to Sp, needs to be refined. Subfigure 2c shows the situation after the non-$\mathcal{C}$-silent states t_1, t_6 and t_7 have been separated: every state is in a block corresponding to its conditional probability to enter Sp in a single step. Also, state t_1 was $\mathcal{C}$-silent but is so no longer, so it had to be removed from its SCC, and the algorithm quickly finds two sub-SCCs in the remaining $\mathcal{C}$-silent states. Note that the transition $t_1 \to t_4$ is ignored when calculating the conditional probability $\mathbf{P}(t_1, Sp \mid \text{non-}\mathcal{C}\text{-inert})$.

The refinement has to be extended to the $\mathcal{C}$-silent SCCs in the block. The situation after this has finished is shown in subfigure 2d: SCC $\{t_2, t_3\}$ is completely added to B_1 because its only outgoing transition goes to a state in B_1. Here, one can see that it is necessary to move the SCC to B_1 as a whole. If we would have tried to move state t_2 individually to B_1, we would have to wait until t_3 is in B_1 first, which in turn depends on t_2 already being in B_1. The SCC $\{t_4, t_5\}$ is moved to the residue because t_5 can enter both $B_{0.1}$ and B_0 in a single step. Note that also Sp has been split into two because it was unstable under $\mathbf{C}' = [Sp]_{\mathcal{C}} \setminus Sp$. State s_2 has (conditional) probability 1 to enter $\mathbf{C}'$ in a single step, while s_3 has conditional probability $.3/(.3 + .2) = 0.6$.

In further refinements, where the new blocks are used as splitters, the algorithm will find that the states in SCC $\{t_2, t_3\}$ are equivalent, but SCC $\{t_4, t_5\}$ will be split up because the states have different probabilities to reach other blocks like B_0.

Algorithm 1 Efficient algorithm for weak bisimulation of Markov chains.

1.1: $\mathcal{B}$:= coarsest partition of S that is at least as fine as $S/\equiv_{AP}$ and satisfies Condition 2 of conditional-probability bisimulation
1.2: Add all $B \in \mathcal{B}$ (except one maximal-size one) to the set of potential splitters
1.3: Label all transitions as $\mathcal{C}$-inert, $\mathcal{C} := \{S\}$
1.4: Construct all SCCs based on the $\mathcal{B}$-inert transitions
1.5: **for all** potential splitters $Sp \in \mathcal{B}$ **do**
1.6: **for all** incoming transitions $t \to s \in in(Sp)$ **do**
1.7: **if** $t \to s$ is labelled as $\mathcal{C}$-inert **then** {i.e. $t \in [Sp]_{\mathcal{C}} \setminus Sp$}
1.8: **if** t is $\mathcal{C}$-silent **then** remove t from $SCC(t)$, $P_{\text{non-}\mathcal{C}\text{-inert}}(t) := 0$, $P_{\to\text{splitter}}(t) := 0$
1.9: $P_{\text{non-}\mathcal{C}\text{-inert}}(t) := P_{\text{non-}\mathcal{C}\text{-inert}}(t) + \mathbf{P}(t, s)$
1.10: Label $t \to s$ as non-$\mathcal{C}$-inert
1.11: **end if**
1.12: **if** $P_{\to\text{splitter}}(t) = 0$ **then** mark state t (as a predecessor)
1.13: $P_{\to\text{splitter}}(t) := P_{\to\text{splitter}}(t) + \mathbf{P}(t, s)$
1.14: **end for**
1.15: **for all** outgoing transitions $s \to t \in out(Sp)$ labelled as $\mathcal{C}$-inert **do** {i.e. $t \in [Sp]_{\mathcal{C}} \setminus Sp$}
1.16: **if** s is $\mathcal{C}$-silent **then** remove s from $SCC(s)$, $P_{\text{non-}\mathcal{C}\text{-inert}}(s) := 0$, $P_{\to\text{splitter}}(s) := 0$
1.17: $P_{\text{non-}\mathcal{C}\text{-inert}}(s) := P_{\text{non-}\mathcal{C}\text{-inert}}(s) + \mathbf{P}(s, t)$
1.18: Label $s \to t$ as non-$\mathcal{C}$-inert
1.19: **if** $P_{\to\text{splitter}}(s) = 0$ **then** mark state s (as a predecessor)
1.20: $P_{\to\text{splitter}}(s) := P_{\to\text{splitter}}(s) + \mathbf{P}(s, t)$
1.21: **end for**
1.22: Remove Sp from the set of potential splitters, $\mathcal{C}' := (\mathcal{C} \setminus \{[Sp]_{\mathcal{C}}\}) \cup \{[Sp]_{\mathcal{C}} \setminus Sp, Sp\}$
1.23: **for all** all blocks B containing marked states **do**
1.24: Refine B according to $\frac{P_{\to\text{splitter}}(\,\cdot\,)}{P_{\text{non-}\mathcal{C}\text{-inert}}(\,\cdot\,)}$ (Algorithm 2)
1.25: **end for**
1.26: $\mathcal{C} := \mathcal{C}'$
1.27: **end for**
1.28: **return** $\mathcal{B}$

4 Detailed algorithm for weak bisimilarity on Markov chains

In this section, we walk through the pseudocode to explain how the ideas of the previous section can be fleshed out.

We store all states such that we can easily traverse all incoming transitions. There is a partition $\mathcal{B}$ of states. We store a set of potential splitter blocks, and we label every transition to indicate whether it is $\mathcal{C}$-inert. As this is all we need to know of the constellation, we do not have to maintain $\mathcal{C}$ explicitly. We treat $\mathcal{C}$ as a ghost variable, as it is only used to prove correctness. Code referring to the ghost variable is printed in grey.

Non-$\mathcal{C}$-silent states are handled individually, while $\mathcal{C}$-silent states are handled as part of an SCC. When some state becomes non-$\mathcal{C}$-silent, the algorithm moves it to the individually-handled states and recalculates the maximal sub-SCCs in the remaining $\mathcal{C}$-silent states.

To make the code slightly more efficient, every non-$\mathcal{C}$-silent state also has an associated probability $P_{\text{non-}\mathcal{C}\text{-inert}}(s)$, which is the probability to take a non-$\mathcal{C}$-inert transition. This is the denominator in the conditional probability $\mathbf{P}(s, \cdot \mid \text{non-}\mathcal{C}\text{-inert})$.

Algorithm 2 Refine block B (Line 1.24).

2.1: Move the non-$\mathcal{C}$-silent states of B to new subblocks $B_p = \{s \in B \mid \frac{P_{\to\text{splitter}}(s)}{P_{\text{non-}\mathcal{C}\text{-inert}}(s)} = p\}$
2.2: Unmark all states and set $P_{\to\text{splitter}}(\,\cdot\,) := 0$ in the subblocks B_p
2.3: **if** for some B_p, we have $|B_p| > \frac{1}{2}$(original size of B) **then** Mark this B_p as aborted
2.4: Create an empty block B_{residue} for the residue
2.5: **while** there are at least two non-completed subblocks (including B_{residue}) **do**
2.6: **for all** non-completed and non-aborted normal subblocks B_p **do**
2.7: Take one step for B_p in Algorithm 3
2.8: **end for**
2.9: **if** B_{residue} is not aborted **then** take one step for B_{residue} in Algorithm 4
2.10: **end while**
2.11: Let B_{aborted} be the only non-completed subblock (or residue).
2.12: **if** $B_{\text{residue}} = \emptyset$ **then**
2.13: **if** $B_{\text{residue}} = B_{\text{aborted}}$ **then** let B_{aborted} := a maximal-size normal B_p
2.14: Delete B_{residue}
2.15: **end if**
2.16: Move all states in B_{aborted} back to B and delete B_{aborted}

Algorithm 1: Main program. We initialize all data structures in Lines 1.1–1.4. To construct the initial partition (Line 1.1), one starts with the partition $S/\equiv_{AP}$ and marks all states s that have a path to a state outside their block $[s]_{AP}$. Then, every block is separated into the marked and unmarked states if necessary. The resulting partition satisfies Condition 2 of conditional-probability bisimulation. – Line 1.4 does not need to check whether states are $\mathcal{C}$-silent, as there are no non-$\mathcal{C}$-inert transitions at this moment.

Each iteration of the main loop refines one constellation of $\mathcal{C}$. This is done by selecting a small splitter Sp and the following two steps.

First, it calculates the probability to enter the splitter Sp in a single step for each state in the loop at Lines 1.6–1.14. It also calculates, for the states in Sp, the probability to enter $[Sp]_{\mathcal{C}} \setminus Sp$ in a single step at Lines 1.15–1.21 because Sp is the one block that is not automatically stabilized under $[Sp]_{\mathcal{C}} \setminus Sp$ by stabilizing under Sp. During these calculations, the algorithm may find that some states in $[Sp]_{\mathcal{C}}$ are no longer $\mathcal{C}$-silent at Lines 1.8 and 1.16. In that case, it (efficiently) recalculates the SCCs that formerly contained these states.

Second, from Line 1.23 it splits all blocks that have some transition to their splitter by calling Algorithm 2.

Algorithm 2: Refine block B. At Line 2.1, we split the non-$\mathcal{C}$-silent states. This can be done by sorting them according to their conditional probability and group those with the same probability in one subblock.

To extend the separation to $\mathcal{C}$-silent states, we start a coroutine (see Algorithm 3) for each initial subblock. Additionally, we also start one coroutine (see Algorithm 4) for the so-called residue, i.e. the part of the block that cannot be put into one subblock because it has paths to several non-$\mathcal{C}$-silent states with different probabilities $P_{\to\text{splitter}}(s)/P_{\text{non-}\mathcal{C}\text{-inert}}(s)$. In the main loop in Lines 2.5–2.10, we let each coroutine do one step (execute one loop iteration, handle one transition) in turn until all of them except one have completed their search. Then, the remaining states must belong to the incomplete subblock. If there is a subblock containing the majority of the states, this will be the incomplete subblock; otherwise, the incomplete subblock will often be large, but it does not matter much. This guarantees that the processing time is proportional to the cumulative size of those subblocks that are at most half the original block.

Algorithm 3 Extend a normal subblock B_p (Line 2.7).

```
3.1: for all unmarked states s ∈ B_p do
3.2:   for all incoming transitions t → s ∈ in(SCC(s)) with C-silent t do
3.3:     if SCC(t) ⊆ B then
3.4:       Move SCC(t) from B to the marked states of B_p
3.5:       untested(SCC(t)) := |{outgoing B-inert transitions of SCC(t)}|
3.6:     end if
3.7:     if SCC(t) ⊆ B_p then {SCC(t) must have been marked}
3.8:       untested(SCC(t)) := untested(SCC(t)) − 1
3.9:       if untested(SCC(t)) = 0 then
3.10:        Unmark SCC(t)
3.11:        if |{unmarked states ∈ B_p}| > ½(original size of B) then
3.12:          Mark B_p as aborted
3.13:          Abort this copy of Algorithm 3
3.14:        end if
3.15:      end if
3.16:    else if SCC(t) ⊆ B_q for some q ≠ p then {SCC(t) must have been marked}
3.17:      Unmark SCC(t) and move it from B_q to B_residue
3.18:      if |B_residue| > ½(original size of B) then Mark B_residue as aborted
3.19:    end if
3.20:  end for
3.21: end for
3.22: Unmark all marked SCCs in B_p and move them to B_residue
3.23: if |B_residue| > ½(original size of B) then Mark B_residue as aborted
3.24: Insert B_p into the set of potential splitters
3.25: Mark B_p as completed
```

After B has been divided completely, the identity of B is reused to become a large subblock (so that most pointers from states to their block need not be changed). Only if the residue is empty, it may have become aborted accidentally, so we correct for that.

Algorithm 3: Extend a normal subblock B_p. This algorithm extends the subblock B_p by all $\mathcal{C}$-silent states in B that only have transitions to B_p-states. The basic idea is as follows. As soon as we find a transition from such a state t to an unmarked B_p-state s, we provisionally add it to B_p as a marked state at Line 3.4 – here, marked states indicate that if t is not in the residue, then it is in B_p. Note that each such transition from t to s is only investigated once. We also initialize a counter $\mathit{untested}(t)$ to the number of outgoing $\mathcal{B}$-inert transitions of t that are not known to go to B_p. When we have determined that no more of such transitions exist, we unmark t at Line 3.10 to indicate that it is definitely in B_p. If, however, we find that state t has already been moved to another $B_q \neq B_p$, it has transitions to several subblocks, so it has to move to the residue at Line 3.17.

The above is slightly complicated by the fact that we may have nontrivial SCCs in B. Therefore, we do not handle $\mathcal{C}$-silent states individually, but always their SCC as a whole. Instead of moving a single state t, we move the whole $\mathit{SCC}(t)$ to B_p or the residue. And instead of considering the incoming transitions of a single $\mathcal{C}$-silent state s, we simultaneously consider all incoming transitions of $\mathit{SCC}(s)$ at Line 3.2. In case s is not $\mathcal{C}$-silent, we regard $\mathit{in}(\mathit{SCC}(s))$ to be $\mathit{in}(s)$ at this line.

Algorithm 4 Extend the residue B_{residue} (Line 2.9).

```
4.1: while there are at least two non-completed subblocks B_p, B_q do
4.2:   for all states s ∈ B_residue that are not yet completely handled do
4.3:     for all incoming transitions t → s ∈ in(SCC(s)) with C-silent t do
4.4:       if SCC(t) ⊆ B_p for some p or SCC(t) ⊆ B then
4.5:         Unmark SCC(t) and move it to B_residue
4.6:         if |B_residue| > ½(original size of B) then
4.7:           Mark the residue as aborted
4.8:           Abort Algorithm 4
4.9:         end if
4.10:      end if
4.11:    end for
4.12:  end for
4.13: end while
4.14: Insert B_residue into the set of potential splitters
4.15: Mark the residue as completed
```

When B_p gets too large at Line 3.11 its handling is aborted, simply meaning that it is not handled further in the algorithm. Note that at most one subblock can be too large, so every subblock except possibly one is completely investigated. At the end of Algorithm 2 the aborted block is cleaned up. When no more states can be added, the states that are still marked must have a transition to B_p and either to some other $B_q \neq B_p$ or to the residue. So, they also move to the residue at Line 3.22.

Algorithm 4: Extend the residue. The handling of the residue is, in a sense, simpler than B_p: every predecessor t of a state in the residue with t originally in B also belongs to the residue. Therefore, we greedily add all states in $SCC(t)$ to the residue.

However, when all states currently in the residue have been handled and still two or more normal subblocks B_p, B_q are incomplete, it may happen that new states will be added to the residue at Lines 3.17 or 3.22 in the coroutines for B_p or B_q. Therefore Algorithm 4 has to wait until (at most) one normal subblock is incomplete (Line 4.1). When only one subblock is incomplete, no other subblock has marked states left that could still move to the residue.

4.1 Correctness

The correctness can be expressed by the following lemmas.

▶ **Lemma 8** (Loop invariants in Algorithm 1)**.**

1. *Whenever Line 1.5 is reached, the following holds:*
   - *$\mathcal{B}$ satisfies Condition 2 of the definition of conditional-probability bisimulation, and is at least as coarse as weak bisimilarity and at least as fine as $S/\equiv_{AP}$ and $\mathcal{C}$.*
   - *Main Invariant 5.*
   - *For every constellation $\mathbf{C} \in \mathcal{C}$, exactly one block $B \subseteq \mathbf{C}$ is not in the set of potential splitters.*
   - *For non-$\mathcal{C}$-silent state $s \in S$, we have $P_{\text{non-}\mathcal{C}\text{-inert}}(s) = 1 - \mathbf{P}(s, [s]_{\mathcal{C}})$ and $P_{\to\text{splitter}}(s) = 0$.*
   - *Every transition is labelled as $\mathcal{C}$-inert iff it is (cf. Line 1.10).*

2. *After executing Line 1.22, it holds:*
   - *For every constellation $\mathbf{C} \in \mathcal{C}'$, exactly one block $B \subseteq \mathbf{C}$ is not in the set of potential splitters.*
   - *For every state $s \in S$,* (†)
     - $P_{\text{non-}\mathcal{C}\text{-inert}}(s) = 1 - \mathbf{P}(s, [s]_{\mathcal{C}'})$.
     - *If $s \notin Sp$, then* $P_{\to\text{splitter}}(s) = \mathbf{P}(s, Sp)$.
     - *If $s \in Sp$, then* $P_{\to\text{splitter}}(s) = \mathbf{P}(s, [Sp]_{\mathcal{C}} \setminus Sp)$.
   - *Every transition is labelled as $\mathcal{C}$-inert iff it is $\mathcal{C}'$-inert.*

In the proof of Lemma 8, we refer to Algorithm 2. Therefore, we first state its correctness:

▶ **Lemma 9** (Correctness of Algorithm 2). *If all states in B satisfy the conditions* (†) *of Lemma 8, then Algorithm 2 splits B into the coarsest subblocks that satisfy*

- *Main Invariant 5 w.r.t. $\mathcal{C}'$, i.e. if s and t are non-$\mathcal{C}'$-silent states in B before Algorithm 2, then they are in the same subblock B_p after Algorithm 2 iff for all constellations $\mathbf{C} \in \mathcal{C}'$ that do not contain s or t, $\mathbf{P}(s, \mathbf{C} \mid \text{non-}\mathcal{C}'\text{-inert}) = \mathbf{P}(t, \mathbf{C} \mid \text{non-}\mathcal{C}'\text{-inert})$.*
- *$\mathcal{C}'$-silent states in B_p have a B_p-inert path to some state outside B_p, but not to a state in another B_q.*
- *All states $\in B_{\text{residue}}$ are $\mathcal{C}'$-silent and have B_{residue}-inert paths to at least two subblocks of the form B_p.*
- *For every state $s \in B$ before Algorithm 2, we have $P_{\to\text{splitter}}(s) = 0$ after Algorithm 2.*

Proof of Lemma 9. Line 2.1 ensures the condition on non-$\mathcal{C}'$-silent states holds; the remainder of Algorithms 2–4 only moves $\mathcal{C}'$-silent states around.

During the execution of Algorithm 3, we have for every $\mathcal{C}'$-silent SCC $\subseteq B$: If all its outgoing $\mathcal{B}$-inert transitions have been tested and go to B_p, then the SCC also is in the unmarked part of B_p. If some (but not all) of its outgoing $\mathcal{B}$-inert transitions have been tested, then the SCC is either in the marked part of B_p or in B_{residue}. (It is in B_{residue} only if it satisfies the conditions in the next paragraph.) Also, every state in B_p has a B_p-inert path to some non-$\mathcal{C}'$-silent state in B_p. From this we can conclude that at Line 3.22, the unmarked $\mathcal{C}'$-silent states of B_p have B_p-inert paths to some non-$\mathcal{C}'$-silent state in B_p (and therefore some state outside B_p), but they have no B_p-inert path to different B_q. The marked $\mathcal{C}'$-silent states of B_p can go in a single step to some state in B_p and also some other state, that either is in some $B_q \neq B_p$ or in B_{residue}. In both cases, adding the marked states to B_{residue} maintains the condition on B_{residue} of the lemma.

During the execution of Algorithm 4, we have for every SCC $\subseteq B$: The SCC is in the residue if it has tested transitions to at least two subblocks of the form B_p (because it was a marked SCC of one copy of Algorithm 3, and another copy executed Line 3.17), or if it has a tested transition to a completed subblock B_p and a transition that leads elsewhere (because it was a marked SCC of B_p at Line 3.22), or if it has a transition to some other state in B_{residue} that was visited in Line 4.5. In all such cases, it consists of $\mathcal{C}'$-silent states which have B_{residue}-inert paths to at least two subblocks of the form B_p. When Algorithm 4 leaves the loop at Lines 4.1–4.13, at most one copy of Algorithm 3 is still running. This copy will not add anything to the residue because all states with transitions to completed copies of Algorithm 3 have already been visited and moved to the appropriate B_p or B_{residue}. ◀

Proof of Lemma 8. When Line 1.5 is reached for the first time, $\mathcal{B}$ and $\mathcal{C}$ have just been initialized to values that obviously satisfy Item 1 of Lemma 8.

Let's now look at the situation after Line 1.22. Lines 1.6–1.22 do not change $\mathcal{B}$ or $\mathcal{C}$, so Item 1 remains valid, except that Sp is not marked as a splitter any more. – The only difference between $\mathcal{C}$ and $\mathcal{C}'$ is that Sp is now in its own constellation. As Sp was in the set of potential splitters, exactly one other block $\subseteq [Sp]_{\mathcal{C}}$ was not in this set, and we now have

that this is the one block $\subseteq [Sp]_{\mathcal{C}} \setminus Sp$ not in the set of potential splitters. Therefore, the first part of Item 2 of Lemma 8 holds. Condition (†) is ensured for every state by the loops in Lines 1.6–1.21. Note that $\mathcal{C}$-inert transitions that enter or leave Sp must be transitions from and to $[Sp]_{\mathcal{C}} \setminus Sp$, so these transitions are not $\mathcal{C}'$-inert.

After Algorithm 2 has been called for all blocks with marked states (i.e. all blocks that have transitions to a splitter) at Lines 1.23–1.25, we have that Item 1 of Lemma 8 is satisfied for $\mathcal{C}'$. In particular, no split by Algorithm 2 separated weakly bisimilar states. Line 1.26 then ensures Item 1 for $\mathcal{C}$. ◀

▶ **Theorem 10.** *Algorithm 1 returns the partition of S into weak bisimilarity equivalence classes.*

Proof. This is an immediate consequence of Item 1 of Lemma 8, in particular Main Invariant 5 and the fact that $\mathcal{B} = \mathcal{C}$ if the set of potential splitters is empty. ◀

4.2 Complexity

The time complexity can be described by the following theorem. Observe that $n \leq m$.

▶ **Theorem 11.** *All operations in the main loop of Algorithm 1 fall under one of the following cases.*

1. *State s is handled as part of a small splitter at most $\lfloor \log_2 n \rfloor$ times. Whenever this happens, $O((|in(s)| + 1) \log n + |out(s)|)$ time is spent.*
2. *State s becomes part of a small subblock at most $\lfloor \log_2 n \rfloor$ times. Whenever this happens, $O(|in(s)| + 1)$ time is spent.*
3. *All operations on decremental SCC handling together run in expected time $O(m \log^4 n)$.*

Summing up, the overall time complexity is in expected time $O(m \log^4 n)$.

Data structures. To prove Theorem 11, we propose that the algorithm stores the following information.

Per block: a set of unmarked non-$\mathcal{C}$-silent states, a set of marked non-$\mathcal{C}$-silent states, a set of unmarked $\mathcal{C}$-silent SCCs, a set of marked $\mathcal{C}$-silent SCCs, and a way to iterate over its incoming and outgoing non-$\mathcal{B}$-inert transitions (possibly through its states and SCCs). The set of potential splitters can be stored as a set or list of pointers to blocks.
In Algorithm 2, some subblocks are marked as completed, and possibly one block is marked as aborted; one can store the non-completed and the completed subblocks in two (circular) list of pointers to blocks, and the aborted block as a (possibly NULL) pointer to a block.

Per state: a flag to indicate whether it is $\mathcal{C}$-silent.

Per non-$\mathcal{C}$-silent state: its block, $P_{\to\text{splitter}}$ and $P_{\text{non-}\mathcal{C}\text{-inert}}$, and a way to iterate over its incoming $\mathcal{B}$-inert transitions.

Per $\mathcal{C}$-silent state: the SCC it belongs to.

Per $\mathcal{C}$-silent SCC: its block, the *untested* counter, the number of states, the number of outgoing $\mathcal{B}$-inert transitions, and a way to iterate over its incoming $\mathcal{B}$-inert transitions. The last three items can be collected and stored while (re)computing the ES-trees in the SCC algorithm without increasing the time complexity.

Per transition: a label to indicate whether it is $\mathcal{C}$-inert.

Proof of Theorem 11. We walk through the algorithms and show under which clause each individual step can be subsumed. *Algorithm 1* mostly falls under Clause 1: the incoming and outgoing transitions of Sp are visited in Lines 1.6–1.22, and every transition is handled in

constant time, except for Lines 1.8 and 1.16. In these lines predecessor states are removed from SCCs. Since a state is initially in at most one SCC, and it can effectively only be removed once, these lines fall under Clause 3. In Lines 1.23–1.25, Algorithm 2 is called.

The first part of *Algorithm 2* runs in time proportional to $\log n$ times the marked states of B. Note that marked states either have a transition to Sp or are in Sp. Therefore, the calls to Algorithm 2 caused by one splitter together lead to at most $|in(Sp)| + |Sp|$ marked states. Hence, those parts of Algorithm 2 that use marked states fall under Clause 1.

At Line 2.1, we separate the non-$\mathcal{C}$-silent states. This boils down to sorting the marked states of B according to the conditional probability $P_{\to\text{splitter}}(s)/P_{\text{non-}\mathcal{C}\text{-inert}}(s)$, which can be done in time $O(|Marked(B)| \log n)$. Note that using techniques as in [7, 20, 12] the factor $\log n$ can be saved, but this will not reduce the overall complexity of our algorithm. The subblock B_p with $p = 0$ requires special care, as it receives the unmarked non-$\mathcal{C}$-silent states of $B \neq Sp$. If $p > 0$, states moved to B_p have transitions to the small splitter Sp, and we are guaranteed to move only a small number of states to B_p. This does not hold for B_0; there may be too many non-$\mathcal{C}'$-silent states that have no transition to the splitter. In our proposed data structure, B_0 can be initialized from the unmarked non-$\mathcal{C}$-silent states, which are stored in a separate subset of B.

Unmarking states and related operations at Lines 2.2–2.3 take $O(|Marked(B)|)$.

The loop at Lines 2.5–2.10 falls under Clause 2. We interpret "Take one step" as "execute one loop iteration". In *Algorithms 3 and 4*, this takes constant time. The work in these algorithms is proportional to the number of states and incoming transitions of the respective subblock. The only part that is not trivial is Line 3.22, which can be executed in time proportional to the incoming transitions of the (clearly small) subblock B_p, as every marked SCC in B_p has a transition to the unmarked states of B_p.

In most cases, executing one loop iteration means that one state or transition is handled, or alternatively, that Algorithm 4 confirms that nothing needs to be done right now. In this way, we ensure that at most one more step is handled in the largest subblock than in the *second* largest subblock. The overhead caused by this is at most proportional to the work done for the second largest subblock (including the residue) and can be attributed to this.

In Clauses 1 and 2 it is indicated that each state is handled at most $\lfloor \log_2 n \rfloor$ times. This is due to the fact that each state when it is processed again is in a block of at most half the size of the previous block it belonged to, according the the principle "Process the smaller half". When we sum the complexity up over all states in Clauses 1 and 2 we obtain the complexity $O((m + n) \log^2 n)$. Together with the complexity of Clause 3, we obtain the overall expected time complexity $O(m \log^4 n)$, as $n \leq m$. ◀

In Theorem 11 we did not include the time complexity of the initialisation of the algorithm. However, if we assume that AP is small enough to find the initial partition fast, namely $|AP| \in O(m/n \log^4 n)$, then we can find an initial partition within the desired overall time complexity. In Line 1.4 the SCCs are constructed, which also fits within the time assigned to Clause 3 of Theorem 11.

5 Conclusion and Outlook

The combination of the ideas from the algorithms from [10, 14] on the one hand and [4] on the other hand allow to come up with a near-linear expected-time algorithm for weak bisimilarity on Markov chains. There are many equivalences that have the same structure, namely direct steps between equivalence classes as in branching bisimilarity and irreducible strongly connected components of inert steps in these equivalence classes. Examples are orthogonal

bisimilarity [21], governed stuttering bisimilarity [6], various branching bisimilarities on non-deterministic probabilistic systems [3, 19] and equivalences using other forms of weighted transitions [17]. We expect that all these equivalences can be provided with a near-linear expected-time algorithm using the techniques provided in this paper.

References

1 Christel Baier and Holger Hermanns. Weak bisimulation for fully probabilistic processes. In Orna Grumberg, editor, *Computer aided verification: CAV*, volume 1254 of *LNCS*, pages 119–130. Springer, Berlin, 1997. doi:10.1007/3-540-63166-6_14.

2 Christel Baier and Holger Hermanns. Weak bisimulation for fully probabilistic processes. Technical Report TR-CTIT-12, Center for Telematics and Information Technology, Enschede, The Netherlands, 1999. URL: https://research.utwente.nl/files/26732523/00000015.pdf.

3 Christel Baier, Joost-Pieter Katoen, Holger Hermanns, and Verena Wolf. Comparative branching-time semantics for Markov chains. *Information and computation*, 200(2):149–214, 2005. doi:10.1016/j.ic.2005.03.001.

4 Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In Moses Charikar and Edith Cohen, editors, *STOC'19: Proceedings of the 51st annual ACM SIGACT symposium on theory of computing*, pages 365–376. ACM, New York, 2019. doi:10.1145/3313276.3316335.

5 M. C. Browne, E. M. Clarke, and O. Grümberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical computer science*, 59(1–2):115–131, 1988. doi:10.1016/0304-3975(88)90098-9.

6 Sjoerd Cranen, Jeroen J.A. Keiren, and Tim A.C. Willemse. A cure for stuttering parity games. In Abhik Roychoudhury and Meenakshi D'Souza, editors, *Theoretical aspects of computing – ICTAC 2012*, volume 7521 of *LNCS*, pages 198–212. Springer, Heidelberg, 2012. doi:10.1007/978-3-642-32943-2_16.

7 Salem Derisavi, Holger Hermanns, and William H. Sanders. Optimal state-space lumping in Markov chains. *Information processing letters*, 87(6):309–315, 2003. doi:10.1016/S0020-0190(03)00343-0.

8 Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *Journal of the ACM*, 28(1):1–4, 1981. doi:10.1145/322234.322235.

9 Rob J. van Glabbeek and Peter W. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the ACM*, 43(3):555–600, 1996. doi:10.1145/233551.233556.

10 Jan Friso Groote, David N. Jansen, Jeroen J. A. Keiren, and Anton Wijs. An $O(m \log n)$ algorithm for computing stuttering equivalence and branching bisimulation. *ACM transactions on computational logic*, 18(2):Article 13, 2017. doi:10.1145/3060140.

11 Jan Friso Groote and Frits Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In M. S. Paterson, editor, *Automata, languages and programming [ICALP]*, volume 443 of *LNCS*, pages 626–638. Springer, Berlin, 1990. doi:10.1007/BFb0032063.

12 Jan Friso Groote, Jao Rivera Verduzco, and Erik P. de Vink. An efficient algorithm to determine probabilistic bisimulation. *Algorithms*, 11(9):131, 2018. doi:10.3390/a11090131.

13 John Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Zvi Kohavi and Azaria Paz, editors, *Theory of machines and computations*, pages 189–196. Academic Press, New York, 1971. doi:10.1016/B978-0-12-417750-5.50022-1.

14 David N. Jansen, Jan Friso Groote, Jeroen J. A. Keiren, and Anton Wijs. An $O(m \log n)$ algorithm for branching bisimilarity on labelled transition systems. In Armin Biere and David Parker, editors, *Tools and algorithms for the construction and analysis of systems: TACAS*, volume 12079 of *LNCS*, pages 3–20. Springer, Cham, 2020. doi:10.1007/978-3-030-45237-7_1.

15 David N. Jansen, Lei Song, and Lijun Zhang. Revisiting weak simulation for substochastic Markov chains. In Kaustubh Joshi, Markus Siegle, Mariëlle Stoelinga, and Pedro D'Argenio, editors, *Quantitative evaluation of systems: QEST*, volume 8054 of *LNCS*, pages 209–224. Springer, Heidelberg, 2013. doi:10.1007/978-3-642-40196-1_18.

16 John G. Kemeny, J. Laurie Snell, and Anthony W. Knapp. *Denumerable Markov chains*. Van Nostrand, Princeton, NJ, 1966.

17 Marino Miculan and Marco Peressotti. Deciding weak weighted bisimulation. In Dario Della Monica, Aniello Murano, Sasha Rubin, and Luigi Sauro, editors, *ICTCS 2017 and CILC 2017: Italian conference on theoretical computer science and Italian conference on computational logic*, volume 1949 of *CEUR Workshop Proceedings*, pages 126–137. CEUR-WS.org, 2017. URL: `http://ceur-ws.org/Vol-1949/ICTCSpaper11.pdf`.

18 Robin Milner. *A calculus of communicating systems*, volume 92 of *LNCS*. Springer, Berlin, 1980. `doi:10.1007/3-540-10235-3`.

19 Andrea Turrini and Holger Hermanns. Polynomial time decision algorithms for probabilistic automata. *Information and computation*, 244:134–171, 2015. `doi:10.1016/j.ic.2015.07.004`.

20 Antti Valmari and Giuliana Franceschinis. Simple $O(m \log n)$ time Markov chain lumping. In Javier Esparza and Rupak Majumdar, editors, *Tools and algorithms for the construction and analysis of systems: TACAS*, volume 6015 of *LNCS*, pages 38–52. Springer, Berlin, 2010. `doi:10.1007/978-3-642-12002-2_4`.

21 Thuy Duong Vu. Deciding orthogonal bisimulation. *Formal aspects of computing*, 19(4):475–485, 2007. `doi:10.1007/s00165-007-0023-x`.

A Proof of Proposition 4

This appendix compares the three notions of bisimilarity. First let's recall the definition of the three bisimulation relations:

▶ **Definition 3.** *Let $\mathcal{M} = (S, AP, \mathbf{P}, L)$ be a DTMC and $R \subseteq \equiv_{AP}$ an equivalence relation on S (which respects the atomic propositions of states). We say that R is*

a weak bisimulation *iff $s\ R\ t$ implies, for all R-equivalence classes $C \in S/R$ with $s, t \notin C$, that $Pr(s, [s]_{AP}, C) = Pr(t, [t]_{AP}, C)$.*

a branching bisimulation *iff $s\ R\ t$ implies, for all R-equivalence classes $C \in S/R$ with $s, t \notin C$, that $Pr(s, [s]_R, C) = Pr(t, [t]_R, C)$.*

a conditional-probability bisimulation *iff $s\ R\ t$ implies*

1. *If s and t are both non-R-silent, for all R-equivalence classes $C \in S/R$ with $s, t \notin C$, we have $\mathbf{P}(s, C \mid \text{non-}R\text{-inert}) = \mathbf{P}(t, C \mid \text{non-}R\text{-inert})$; and*
2. *s has a path to a state outside $[s]_R$ iff t has a path to a state outside $[t]_R$.*

The states s and t are weakly bisimilar, *denoted $s \approx_{\mathrm{w}} t$, iff a weak bisimulation R exists such that $s\ R\ t$. Similarly, s and t are* branching bisimilar, *denoted $s \approx_{\mathrm{b}} t$, iff a branching bisimulation R exists such that $s\ R\ t$. Finally, s and t are* conditional-probability-bisimilar, *denoted $s \approx_{\mathrm{c}} t$, iff a conditional-probability bisimulation R exists such that $s\ R\ t$.*

▶ **Proposition 4.** *Let $\mathcal{M} = (S, AP, \mathbf{P}, L)$ be a DTMC. States in S are weakly bisimilar iff they are branching bisimilar iff they are conditional-probability-bisimilar.*

We prove this proposition by the three lemmas below. The proofs are adapted from [2], where Markov chains are defined with action labels instead of atomic propositions.

▶ **Lemma 12.** *Let R be a conditional-probability bisimulation. Then R is a branching bisimulation.*

Proof. Consider a state $s \in S$. If all states in $[s]_R$ are R-silent, they are all trivially branching bisimilar to s, as $Pr(s, [s]_R, C) = 0$ for all $C \in S/R \backslash \{[s]_R\}$. Otherwise, $Pr(s, [s]_R, S \backslash [s]_R) = 1$ because S is finite. Fix some non-R-silent $s' \in [s]_R$. Note that for any $C \in S/R \setminus \{[s]_R\}$, we have that $\mathbf{P}(s', C \mid \text{non-}R\text{-inert})$ does not depend on the concrete choice of s'. Then,

$$
\begin{aligned}
Pr(s,[s]_R,C) &= \sum_{s_1,\ldots,s_n\in[s]_R, s_{n+1}\in C} Pr_{\delta(s)}(Cyl(s,s_1,\ldots,s_n,s_{n+1})) \\
&= \sum_{s_1,\ldots,s_n\in[s]_R} Pr_{\delta(s)}(Cyl(s,s_1,\ldots,s_n))\mathbf{P}(s_n,C) \\
&= \sum_{s_1,\ldots,s_n\in[s]_R} Pr_{\delta(s)}(Cyl(s,s_1,\ldots,s_n))\mathbf{P}(s_n,C \mid \text{non-}R\text{-inert})(1-\mathbf{P}(s_n,[s]_R)) \\
&= \mathbf{P}(s',C \mid \text{non-}R\text{-inert}) \sum_{s_1,\ldots,s_n\in[s]_R} Pr_{\delta(s)}(Cyl(s,s_1,\ldots,s_n))\mathbf{P}(s_n,S\setminus[s]_R) \\
&= \mathbf{P}(s',C \mid \text{non-}R\text{-inert}) \sum_{s_1,\ldots,s_n\in[s]_R, s_{n+1}\notin[s]_R} Pr_{\delta(s)}(Cyl(s,s_1,\ldots,s_n,s_{n+1})) \\
&= \mathbf{P}(s',C \mid \text{non-}R\text{-inert})Pr(s,[s]_R,S\setminus[s]_R) \\
&= \mathbf{P}(s',C \mid \text{non-}R\text{-inert})
\end{aligned}
$$

By the same method $Pr(t,[s]_R,C)=\mathbf{P}(s',C \mid \text{non-}R\text{-inert})$ for any $t\in[s]_R$, so $Pr(s,[s]_R,C)=Pr(t,[t]_R,C)$ whenever $s\ R\ t$. ◀

▶ **Lemma 13.** *Let R be a branching bisimulation. Then R is a weak bisimulation.*

Proof. We first define the auxiliary notion of a block-cylinder set: $Cyl(B_1,B_2,\ldots,B_n)$ is the set of paths that start in B_1, after visiting a number of states in B_1 move on to B_2, and visit all further blocks in the order mentioned. After entering B_n the paths may continue without any restrictions. A block-cylinder set is a countable union of (basic) cylinder sets. Hence, it is in the σ-algebra generated by the basic cylinder sets allowing to write $Pr_{\delta(s)}(Cyl([s]_R,B_1,B_2,\ldots,B_n,C))$. We can therefore define:

$$Pr(s,[s]_R,B_1,B_2,\ldots,B_n,C) := Pr_{\delta(s)}(Cyl([s]_R,B_1,B_2,\ldots,B_n,C)).$$

Further, we have, for all states $s\in S\setminus C$,

$$Pr(s,[s]_{AP},C) = \sum_{\substack{B_1,B_2,\ldots,B_n\in S/R \\ B_1,B_2,\ldots,B_n\subseteq[s]_{AP}\setminus C \\ [s]_R\neq B_1\neq B_2\neq\cdots\neq B_n}} Pr(s,[s]_R,B_1,B_2,\ldots,B_n,C).$$

Now assume given two branching bisimilar states $s\ R\ t$. Note that $[s]_{AP}=[t]_{AP}$. It is easy to show that $Pr(s,[s]_R,B_1,B_2,\ldots,B_n,C)=Pr(t,[t]_R,B_1,B_2,\ldots,B_n,C)$ for blocks $B_i\in S/R$ for $i=1,\ldots,n$, and therefore $Pr(s,[s]_{AP},C)=Pr(t,[t]_{AP},C)$. ◀

▶ **Lemma 14.** *Weak bisimilarity, $\approx_{\mathrm{w}}$, is a conditional-probability bisimulation.*

Proof. We adapt the proof of [1] (for action-labelled fully probabilistic systems) to our state-labelled DTMCs. Condition 2 of conditional-probability bisimulation is easy to check, so we concentrate on Condition 1.

Let $B_1,B_2,\ldots,B_k$ be an enumeration of the $\approx_{\mathrm{w}}$-equivalence classes with at least one non-$\approx_{\mathrm{w}}$-silent state, and let $B_0,B_{-1},\ldots,B_{k_{\mathrm{sil}}}$ be an enumeration of the other $\approx_{\mathrm{w}}$-equivalence classes, in which all states are $\approx_{\mathrm{w}}$-silent. Let s be a non-$\approx_{\mathrm{w}}$-silent state. Assume w.l.o.g. that $[s]_{\approx_{\mathrm{w}}}=B_k$. Let $C\neq[s]_{\approx_{\mathrm{w}}}$ be an $\approx_{\mathrm{w}}$-equivalence class. Then

$$Pr(s,[s]_{AP},C) = \sum_{i=k_{\mathrm{sil}}}^{k} \mathbf{P}(s,B_i)Pr(B_i,[s]_{AP},C).$$

As $\approx_{\mathrm{w}}$ is a weak bisimulation, $Pr(B_i, [s]_{AP}, C)$ does not depend on the concrete initial state in B_i if $B_i \neq C$, and $Pr(C, [s]_{AP}, C) = 1$. Therefore the above sum is well-defined. Note that $Pr(B_i, [s]_{AP}, C) = 0$ if $C \neq B_i \not\subseteq [s]_{AP}$. Then

$$\begin{aligned}\frac{Pr(s, [s]_{AP}, C)}{1 - \mathbf{P}(s, [s]_{\approx_{\mathrm{w}}})} &= \sum_{i=k_{\mathrm{sil}}}^{k} \frac{\mathbf{P}(s, B_i)}{1 - \mathbf{P}(s, [s]_{\approx_{\mathrm{w}}})} Pr(B_i, [s]_{AP}, C) \\ &= \sum_{i=k_{\mathrm{sil}}}^{k-1} \frac{\mathbf{P}(s, B_i)}{1 - \mathbf{P}(s, [s]_{\approx_{\mathrm{w}}})} Pr(B_i, [s]_{AP}, C) \quad + \frac{\mathbf{P}(s, B_k)}{1 - \mathbf{P}(s, [s]_{\approx_{\mathrm{w}}})} Pr(B_k, [s]_{AP}, C) \\ &= \sum_{i=k_{\mathrm{sil}}}^{k-1} \frac{\mathbf{P}(s, B_i)}{1 - \mathbf{P}(s, [s]_{\approx_{\mathrm{w}}})} Pr(B_i, [s]_{AP}, C) \quad + \frac{\mathbf{P}(s, [s]_{\approx_{\mathrm{w}}})}{1 - \mathbf{P}(s, [s]_{\approx_{\mathrm{w}}})} Pr(s, [s]_{AP}, C).\end{aligned}$$

We now solve the above equation for $Pr(s, [s]_{AP}, C)$ and get:

$$Pr(s, [s]_{AP}, C) = \sum_{i=k_{\mathrm{sil}}}^{k-1} \frac{\mathbf{P}(s, B_i)}{1 - \mathbf{P}(s, [s]_{\approx_{\mathrm{w}}})} Pr(B_i, [s]_{AP}, C).$$

This shows that the conditional probabilities $x_i = \mathbf{P}(s, B_i \mid \text{non-}\approx_{\mathrm{w}}\text{-inert}) = \frac{\mathbf{P}(s,B_i)}{1-\mathbf{P}(s,[s]_{\approx_{\mathrm{w}}})}$ for $i \in \{k_{\mathrm{sil}}, \ldots, k-1\}$ solve the following linear equation system:

$$\sum_{i=k_{\mathrm{sil}}}^{k-1} x_i Pr(B_i, [s]_{AP}, C) = Pr([s]_{\approx_{\mathrm{w}}}, [s]_{AP}, C) \quad \text{for all } C \in \{B_{k_{\mathrm{sil}}}, \ldots, B_{k-1}\} \tag{1}$$

We claim that Equation System (1) has at most one solution. Therefore, for each non-$\approx_{\mathrm{w}}$-silent state $t \in [s]_{\approx_{\mathrm{w}}}$, we have $\mathbf{P}(t, B_i \mid \text{non-}\approx_{\mathrm{w}}\text{-inert}) = \mathbf{P}(s, B_i \mid \text{non-}\approx_{\mathrm{w}}\text{-inert})$, which proves this lemma.

We now prove the claim that Equation System (1) has at most one solution. We first prove that the matrix $\mathbf{A} = (Pr(B_j, [s]_{AP}, B_i))_{i,j \in \{1,\ldots,k\}}$ is regular. For every B_i (with $i \geq 1$), fix some state $s_i \in B_i$ that is non-$\approx_{\mathrm{w}}$-silent. Let

$$\mathbf{C} = (c_{ij})_{i,j \in \{1,\ldots,k\}} \quad \text{where } c_{ij} = \begin{cases} \mathbf{P}(s_j, B_i) & \text{if } s_j \in [s]_{AP} \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{E} = \begin{pmatrix} 1 - e_1 & & 0 \\ & \ddots & \\ 0 & & 1 - e_k \end{pmatrix} \quad \text{where } e_i = \begin{cases} \sum_{h=1}^{k} \mathbf{P}(s_i, B_h) Pr(B_h, [s]_{AP}, B_i) & \text{if } s_i \in [s]_{AP} \\ 0 & \text{otherwise.} \end{cases}$$

Let $\mathbf{D} = (d_{ij})_{i,j \in \{1,\ldots,k\}} = \mathbf{A} \cdot \mathbf{C} + \mathbf{E}$. We show that $\mathbf{D} = \mathbf{A}$. For the diagonal elements d_{ii} with $s_i \in [s]_{AP}$, we have:

$$d_{ii} = \sum_{h=1}^{k} Pr(B_h, [s]_{AP}, B_i) \mathbf{P}(s_i, B_h) + 1 - e_i = 1 = Pr(B_i, [s]_{AP}, B_i)$$

and for the off-diagonal elements d_{ij} with $s_j \in [s]_{AP}$, we have:

$$Pr(B_j, [s]_{AP}, B_i) = Pr(s_j, [s]_{AP}, B_i) = \sum_{h=1}^{k} Pr(B_h, [s]_{AP}, B_i) \mathbf{P}(s_j, B_h) = d_{ij}$$

Elements d_{ij} with $s_j \notin [s]_{AP}$ are trivial.

This proves that $\mathbf{A} \cdot \mathbf{C} + \mathbf{E} = \mathbf{A}$, so also $\mathbf{E} = \mathbf{A} \cdot (\mathbf{I} - \mathbf{C})$. Next we show that all $e_i < 1$. This is trivial if $s_i \notin [s]_{AP}$, so we now assume that $B_i \subseteq [s]_{AP}$.

- Assume that there is some $B_{h_0} \neq B_i$ with $Pr(B_{h_0}, [s]_{AP}, B_i) \neq 0$. Note that if some state s' satisfies $Pr(s', [s]_{AP}, B_i) = 1$ then $s' \in B_i$ (here we use that we started with $\approx_{\mathrm{w}}$ and not just any weak bisimulation); therefore, $Pr(B_{h_0}, [s]_{AP}, B_i) < 1$. Then

$$e_i \leq \sum_{\substack{h=1 \\ h \neq h_0}}^{k} \mathbf{P}(s_i, B_h) + \mathbf{P}(s_i, B_{h_0}) Pr(B_{h_0}, [s]_{AP}, B_i) < \mathbf{P}(s_i, S) \leq 1.$$

- If for all $B_i \neq B_j$ we have $Pr(B_j, [s]_{AP}, B_i) = 0$, then

$$e_i = \mathbf{P}(s_i, B_i) Pr(B_i, [s]_{AP}, B_i) = \mathbf{P}(s_i, B_i) < 1.$$

Therefore, $\mathbf{E}$ is regular, and hence $\mathbf{A}$ is regular because $\mathbf{A}^{-1} = (\mathbf{I} - \mathbf{C}) \cdot \mathbf{E}^{-1}$.

Now we prove that the extended matrix $\mathbf{A}^0 = (Pr(B_j, [s]_{AP}, B_i))_{i,j \in \{k_{\mathrm{sil}}, \ldots, k\}}$ is regular. Note that if $i \neq j \leq 0$, then $Pr(B_j, [s]_{AP}, B_i) = 0$. So, we can write $\mathbf{A}^0$ as follows

$$\mathbf{A}^0 = \begin{pmatrix} \mathbf{I} & * \\ 0 & \mathbf{A} \end{pmatrix}.$$

Hence, $\mathbf{A}^0$ is regular as well. The matrix of Equation System (1) is the matrix $\mathbf{A}^0$ without the last column and row, i.e., $(Pr(B_j, [s]_{AP}, B_i))_{i,j \in \{k_{\mathrm{sil}}, \ldots, k-1\}}$. Let $\mathbf{A'}$ be $\mathbf{A}^0$ without the last row, and consider the equation system $\mathbf{A'x} = \mathbf{b}$, where $\mathbf{b}_i = Pr([s]_{\approx_{\mathrm{w}}}, [s]_{AP}, B_i)$ for $i = k_{\mathrm{sil}}, \ldots, k-1$. If some $\mathbf{x}$ solves this equation system, then $(\mathbf{x}_i)_{i=k_{\mathrm{sil}}, \ldots, k-1}$ solves (1) iff $\mathbf{x}_k = 0$. We argue that there is at most one such $\mathbf{x}$.

As $\mathbf{A}^0$ is regular, the solution space of $\mathbf{A'x} = \mathbf{b}$ is (at most) one-dimensional and can therefore be described by $\{\mathbf{a} + t \cdot \mathbf{c} \mid t \in \mathbb{R}\}$ for some vectors $\mathbf{a}, \mathbf{c} \in \mathbb{R}^{k - k_{\mathrm{sil}} + 1}$.

To come to a contradiction, we assume $\mathbf{a}_k = \mathbf{c}_k = 0$ and $\mathbf{c} \neq \mathbf{0}$; in that case, there would be more than one solution of (1). We know $\mathbf{A'}(\mathbf{a} + t\mathbf{c}) = \mathbf{b}$ for all $t \in \mathbb{R}$, so

$$\sum_{h=k_{\mathrm{sil}}}^{k-1} Pr(B_h, [s]_{AP}, B_i)(\mathbf{a}_h + t\mathbf{c}_h) = Pr([s]_{\approx_{\mathrm{w}}}, [s]_{AP}, B_i) \quad \text{for } i = k_{\mathrm{sil}}, \ldots, k-1.$$

Because this holds for all $t \in \mathbb{R}$, we must have that

$$\sum_{h=k_{\mathrm{sil}}}^{k-1} Pr(B_h, [s]_{AP}, B_i)\mathbf{c}_h = 0 \quad \text{for } i = k_{\mathrm{sil}}, \ldots, k-1.$$

Because $\mathbf{A}^0$ is regular, we cannot have $\mathbf{A}^0\mathbf{c} = \mathbf{0}$, therefore

$$c := \sum_{h=1}^{k-1} Pr(B_h, [s]_{AP}, B_k)\mathbf{c}_h \neq 0.$$

By these equations, the vector $(\frac{1}{c}\mathbf{c}_i)_{i=1,\ldots,k}$ is equal to the last row of $\mathbf{A}^{-1}$. We have $0 = \frac{1}{c}\mathbf{c}_k = (\mathbf{A}^{-1})_{kk} = (\mathbf{I} - \mathbf{C})_{kk}/(1 - e_k) = (1 - \mathbf{P}(s_k, B_k))/(1 - e_k) \neq 0$. Contradiction! ◀

Characterizing Consensus in the Heard-Of Model

A. R. Balasubramanian
Technical University of Munich, Germany

Igor Walukiewicz
CNRS, LaBRI, University of Bordeaux, France

Abstract

The Heard-Of model is a simple and relatively expressive model of distributed computation. Because of this, it has gained a considerable attention of the verification community. We give a characterization of all algorithms solving consensus in a fragment of this model. The fragment is big enough to cover many prominent consensus algorithms. The characterization is purely syntactic: it is expressed in terms of some conditions on the text of the algorithm.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Software and its engineering → Software verification; Theory of computation → Logic and verification

Keywords and phrases consensus problem, Heard-Of model, verification

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.9

Related Version A full version of this paper is available at `https://arxiv.org/abs/2004.09621`.

Funding *A. R. Balasubramanian*: UMI ReLaX, ERC Advanced Grant 787367 (PaVeS).
Igor Walukiewicz: ANR FREDDA ANR-17-CE40-0013.

1 Introduction

Most distributed algorithms solving problems like consensus, leader election, set agreement, or renaming are essentially one iterated loop. Yet, their behavior is difficult to understand due to unbounded number of processes, asynchrony, failures, and other aspects of the execution model. The general context of this work is to be able to say what happens when we change some of the parameters: modify an algorithm or the execution model. Ideally we would like to characterize the space of all algorithms solving a particular problem.

To approach this kind of questions, one needs to restrict to a well defined space of all distributed algorithms and execution contexts. In general this is an impossible requirement. Yet the distributed algorithms community has come up with some settings that are expressive enough to represent interesting cases and limited enough to start quantifying over "all possible" distributed algorithms [11, 40, 1].

In this work we consider the consensus problem in the Heard-Of model [11]. *Consensus problem* is a central problem in the field of distributed algorithms; it requires that all correct processes eventually decide on one of the initial values. *The Heard-Of model* is a round- and message-passing-based model. It can represent many intricacies of various execution models and yet is simple enough to attempt to analyze it algorithmically [9, 14, 15, 28, 27]. Initially, our goal was to continue the quest from [28] of examining what is algorithmically possible to verify in the Heard-Of model. While working on this problem we have realized that a much more ambitious goal can be achieved: to give a simple, and in particular decidable, characterization of all consensus algorithms in well-defined fragments of the Heard-Of model.

The Heard-Of model is an open ended model: it does not specify what operations processes can perform and what kinds of communication predicates are allowed. Communication predicates in the Heard-Of model capture in an elegant way both synchrony degree and failure model. In this work we fix the set of atomic communication predicates and atomic operations. We opted for a set sufficient to express most prominent consensus algorithms (cf. Section 7), but we do not cover all operations found in the literature on the Heard-Of model.

Our characterization of algorithms that solve consensus is expressed in terms of syntactic conditions both on the text of the algorithm, and on the constraints in the communication predicate. It exhibits an interesting way all consensus algorithms should behave. One could imagine that there can be a consensus algorithm that makes processes gradually converge to a consensus: more and more processes adopting the same value. This is not the case. A consensus algorithm, in models we study here, should have a fixed number of crucial rounds where precise things are guaranteed to happen. Special rounds have been identified for existing algorithms [33], but not their distribution over different phases. Additionally, here we show that all algorithms should have this structure.

As an application of our characterization we can think of using it as an intermediate step in analysis of more complicated settings than the Heard-Of model. An algorithm in a given setting can be abstracted to an algorithm in the Heard-Of model, and then our characterization can be applied. Instead of proving the original algorithm correct it is enough to show that the abstraction is sound. For example, an approach reducing asynchronous semantics to round based semantics under some conditions is developed in [8]. A recent paper [13] gives a reduction methodology in a much larger context, and shows its applicability. The goal language of the reduction is an extension of the Heard-Of model that is not covered by our characterization. As another application, our characterization can be used to quickly see if an algorithm can be improved by taking a less constrained communication predicate, by adapting threshold constants, or by removing parts of code (c.f. Section 7).

Organization of the paper. In the next section we introduce the Heard-Of model and formulate the consensus problem. In the four consecutive sections we present the characterizations for the core model as well as for the extensions with timestamps, coordinators, and with both timestamps and coordinators at the same time. We then give examples of algorithms that are covered by these characterizations. Proofs can be found in the appendix, as well as in the full version of the paper [3].

Related work

The celebrated FLP result [18] states that consensus is impossible to achieve in an asynchronous system in presence of failures, even in the presence of one crash failure. There is a considerable literature investigating the models in which the consensus problem is solvable. Even closer in spirit to the present paper are results on weakest failure detectors required to solve the problem [6, 19]. Another step closer are works providing generic consensus algorithms that can be instantiated to give several known concrete algorithms [31, 22, 21, 5, 34, 33]. The present paper considers a relatively simple model, but gives a characterization result of all possible consensus algorithms.

The cornerstone idea of the Heard-Of model is to represent both asynchrony and failures by the constraints on the message loss expressed by communication predicates. This greatly simplifies the model, that in turn is very useful for a kind of characterizations we present here. Unavoidably, not all aspects of partial synchrony [17, 12] or failures [7] are covered by the model. For example, after a crash it may be difficult for a process to get into initial state, or in terms of the Heard-of model, do the same round as other processes [38, 8]. Faults are not malicious: a sent value may be lost, but the value may not be modified during transmission. These observations just underline that there is no universal model for distributed algorithms. There exists several other proposals of relatively simple and expressible models [20, 40, 1, 32]. The Heard-Of model, while not perfect, is in our opinion representative enough to merit a more detailed study.

On the verification side there are at least three approaches to analysis of the Heard-Of or similar models. One is to use automatic theorem provers like Isabelle [10, 9, 14]. Another is deductive verification methods applied to annotated programs [16, 15]. The closest to this work is a model-checking approach [37, 28, 27, 2]. Particularly relevant here is the work of Maric et al. [28]. who show cut-off results for a fragment of the Heard-Of model and then perform verification on a resulting finite state system. Our fragment of the Heard-Of model is incomparable with the one from that work, and arguably it has less restrictions coming from purely technical issues in proofs. While trying to extend the scope of automatic verification methods along the lines in the above papers, we have realized that in our case it is possible to obtain a characterization result.

Of course there are also other models of distributed systems that are considered in the context of verification. For example there has been big progress on verification of threshold automata [26, 24, 25, 35, 4]. There are also other methods, as automatically generating invariants for distributed algorithms [23, 39, 36], or verification in Coq proof assistant [41, 42].

2 Heard-Of model and the consensus problem

In the Heard-Of model a certain number of processes execute the same code synchronously. An algorithm consists of a sequence of *rounds*, every process executes the same round at the same time. The sequence of rounds, called *phase*, is repeated forever. In a round every process sends the value of one of its variables to a communication medium, receives a multiset of values, and uses it to adopt a new value (cf. Figure 1).

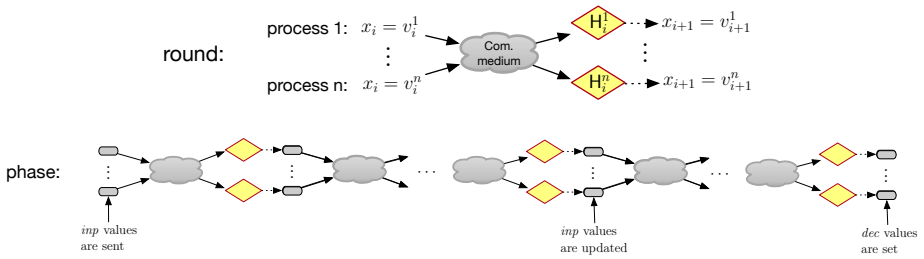


Figure 1 A schema of an execution of a round and of a phase. In every round i every process sends a value of its variable x_i, and sets its variable x_{i+1} depending on the received multiset of values: H_i^j. At the beginning of the phase the value of *inp* is sent, at some round *inp* may be updated; we use **ir** for the index of this round. In the last round *dec* may be set. Both *inp* and *dec* are not updated if the value is ?, standing for undefined.

At the beginning every process has its initial value in variable *inp*. Every process is expected to eventually set its decision variable *dec*. Every round is communication closed meaning that a value sent in a round can only be received in the same round; if it is not received it is lost. A *communication predicate* is used to express a constraint on acceptable message losses. Algorithm 1 is a concrete simple example of a 2-round algorithm with two rounds. In the first round the value of *inp* is send, in the second the value of x_1. We will explain the algorithm later in the text.

We proceed with a description of the syntax and semantics of Heard-Of algorithms. Next we define the consensus problem. In later sections we will extend the core language with timestamps and coordinators.

Algorithm 1 Parametrized OneThird algorithm [11], thr_1, thr_2 are constants from $(0, 1)$.

```
send (inp)
    if uni(H) ∧ |H| > thr_1 · |Π| then x_1 := inp := smor(H);
    if mult(H) ∧ |H| > thr_1 · |Π| then x_1 := inp := smor(H);
send x_1
    if uni(H) ∧ |H| > thr_2 · |Π| then dec := smor(H);
Communication predicate: F(ψ^1 ∧ Fψ^2)
where: ψ^1 := (φ_= ∧ φ_thr1, true)   and   ψ^2 := (φ_thr1, φ_thr2)
```

Syntax

An algorithm has one *phase* that consists of two or more rounds. In the first round each process sends the value of *inp* variable, in the last round it can set the value of *dec* variable. A phase is repeated forever, all processes execute the same round at the same time. A round i is a send statement followed by a sequence of conditionals:

```
send x_{i-1}
    if cond_i^1(H) then x_i := op_i^1(H);
    ⋮
    if cond_i^l(H) then x_i := op_i^l(H);
```

The variables are used in a sequence: first x_0, which is *inp*, is sent and x_1 is set, then x_1 is sent and x_2 is set, etc. (cf. Figure 1). There should be exactly one round (before the last round) where *inp* is updated; the conditional lines in this round are:

$$\texttt{if } cond^j_{\mathbf{ir}}(\mathsf{H}) \texttt{ then } x_{\mathbf{ir}} := inp := \mathtt{op}^j_{\mathbf{ir}}(\mathsf{H})$$

Since this is a special round, we use the index **ir** to designate this round number. In the last round, only instructions setting variable *dec* can be present:

$$\texttt{if } cond^j_r(\mathsf{H}) \texttt{ then } dec := \mathtt{op}^j_r(\mathsf{H})$$

Because of this special form of the last round, a phase needs to have at least two rounds. Of course one can also have a syntax and a characterization for one round algorithms, but unifying the two hinders readability. Our fragment roughly corresponds to the fragment from [28], without extra restrictions but with a less liberty at the fork point.

The intuition behind the syntax is that in the i^{th} round, after a process sends its value of the x_{i-1} variable and receives a multiset H, it finds the first instruction whose condition is satisfied by H and performs the corresponding assignment. Hence, even if multiple conditions are satisfied by a multi-set, only the first such condition is executed.

As an example, consider Algorithm 1. It has two rounds, each begins with a `send` statement. In the first round both x_1 and *inp* are set, in the second round *dec* is set. The conditions talk about properties of the received H multiset; we describe them below.

In round i every process first sends the value of variable x_{i-1}, and then receives a multiset of values H that it uses to set the value of the variable x_i. The possible tests on the received set H are `uni`, `mult`, and $|\mathsf{H}| > thr \cdot |\Pi|$ saying respectively that: the multiset has only one value; has more than one value; and that is of size $> thr \cdot n$ where n is the number of processes and $0 \leq thr < 1$. The possible operations are $\min(\mathsf{H})$ resulting in the minimal value in H, and $\mathrm{smor}(\mathsf{H})$ resulting in the minimal most frequent value in H. For example, the first conditional line in Algorithm 1 tests if there is only one value in H, and if this value has

multiplicity at least $thr_1 \cdot n$ in H; if so *inp* and x_1 are set to this value, it does not matter if min or smor operation is used in this case. The test in the second line holds when received H set has at least two values and is of size at least $thr_1 \cdot n$. In this case x_1 is set to the smallest most frequent value in H.

In addition to description of rounds, an algorithm has also a communication predicate putting constraints on the behavior of the communication medium. A *communication predicate for a phase* with r rounds is a tuple $\psi = (\psi_1, \ldots, \psi_r)$, where each ψ_l is a conjunction of atomic communication predicates that we specify later. A *communication predicate for an algorithm* is

$$(\mathsf{G}\overline{\psi}) \wedge (\mathsf{F}(\psi^1 \wedge \mathsf{F}(\psi^2 \wedge \ldots (\mathsf{F}\psi^k) \ldots)))$$

where $\overline{\psi}$ and ψ^i are communication predicates for a phase. Predicate $\overline{\psi}$ is a *global predicate*, and $\psi^1 \ldots, \psi^k$ are *sporadic predicates*. So the global predicate specifies constraints on every phase of execution, while sporadic predicates specify a *sequence* of special phases that should happen: first ψ_1, followed later by ψ_2, etc. We have two types of atomic communication predicates: $\varphi_=$ says that every process receives the same multiset; φ_{thr} says that every process receives a multiset of size at least $thr \cdot n$ where n is the number of processes. In Algorithm 1 the global predicate is trivial, and we require two special phases. In the first of them, in its first round every process should receive exactly the same H multiset, and the multiset should contain values from at least thr_1 fraction of all processes.

Semantics

The values of variables come from a fixed linearly ordered set D. Additionally, we take a special value $? \notin D$ standing for undefined. We write $D_?$ for $D \cup \{?\}$.

We describe the semantics of an algorithm for n processes. A *state of an algorithm* is a pair of n-tuples of values; denoted (f, d). Intuitively, f specifies the value of the *inp* variable for each process, and d specifies the value of the *dec* variable. The value of *inp* can never be ?, while initially the value of *dec* is ? for every process. We denote by $mset(f)$ the multiset of values appearing in the tuple f. Only values of *inp* and *dec* survive between phases. All the other variables are reset to ? at the beginning of each phase.

There are two kinds of transitions:

$(f, d) \xrightarrow{\psi} (f', d')$	a phase transition
$f \xRightarrow{\varphi}_i f'$	a transition for round i

Phase transitions will be defined as a composition of round transitions. In a transition for round i, tuple f describes the values of x_{i-1}, and f' the values of x_i Phase transition is labeled with a phase communication predicate, while a round transition has a round number and a conjunction of atomic predicates as labels.

Before defining these transitions we need to describe the semantics of communication predicates. At every round processes send values of their variable to a communication medium, and then receive a multiset of values from the medium (cf. Figure 1). Communication medium is not assumed to be perfect, it can send a different multiset of values to every process, provided it is a sub-multiset of received values. An atomic communication predicate puts constraints on multisets that every process receives. In other words, such a predicate specifies constraints on a tuple of multisets $\vec{\mathsf{H}} = (\mathsf{H}_1, \ldots, \mathsf{H}_n)$. Predicate $\varphi_=$ requires that all the multisets are the same. Predicate φ_{thr} requires that every multiset is bigger than $thr \cdot n$ for some number $0 \leq thr < 1$. Predicate *true* does not put any restrictions. We write $\vec{\mathsf{H}} \vDash \varphi$ when the tuple of multisets $\vec{\mathsf{H}}$ satisfies the conjunction of atomic predicates φ.

Once a process p receives a multiset H_p, it uses it to do an update of one of its variables. For this it finds the first condition that H_p satisfies and performs the operation from the corresponding assignment.

Recall that a *condition* is a conjunction of atomic conditions: `uni`, `mult`, $|\mathsf{H}| > thr \cdot |\Pi|$. A multiset H satisfies `uni` when it contains just one value; it satisfies `mult` if it contains more than one value. A multiset H satisfies $|\mathsf{H}| > thr \cdot |\Pi|$ when the size of H is bigger than $thr \cdot n$, where n is the number of processes. Observe that only predicates of the last type take into account possible repetitions of the same value.

We can now define the *update value* $\texttt{update}_i(\mathsf{H})$, describing to which value the process sets its variable in round i upon receiving the multiset H. For this the process finds the first conditional statement in the sequence of instructions for round i whose condition is satisfied by $\mathsf{H} - \{?\}$ and looks at the operation in the statement:

- if it is $x := \min(\mathsf{H})$ then $\texttt{update}_i(\mathsf{H})$ is the minimal value in $\mathsf{H} - \{?\}$;
- if it is $x := \mathrm{smor}(\mathsf{H})$ then $\texttt{update}_i(\mathsf{H})$ is the smallest most frequent value in $\mathsf{H} - \{?\}$;
- if no condition is satisfied then $\texttt{update}_i(\mathsf{H}) = ?$.

A transition $f \overset{\varphi}{\Longrightarrow}_i f'$ is possible when there exists a tuple of multisets $(\mathsf{H}_1, \ldots, \mathsf{H}_n) \vDash \varphi$ such that for all $p = 1, \ldots, n$: $\mathsf{H}_p \subseteq mset(f)$, and $f'(p) = \texttt{update}_i(\mathsf{H}_p)$. Observe that ? value in H_p is ignored by the `update` function, but not by the communication predicate.

A transition $(f, d) \xrightarrow{\psi} (f', d')$, for $\psi = (\varphi_1, \ldots, \varphi_n)$, is possible when there is a sequence:

$$f_0 \overset{\varphi_1}{\Longrightarrow}_1 f_1 \overset{\varphi_2}{\Longrightarrow}_2 \cdots \overset{\varphi_{r-1}}{\Longrightarrow}_{r-1} f_{r-1} \overset{\varphi_r}{\Longrightarrow}_r f_r \qquad \text{where}$$

- $f_0 = f$;
- $f'(p) = f_{\mathbf{ir}}(p)$ if $f_{\mathbf{ir}}(p) \neq ?$, and $f'(p) = f_{\mathbf{ir}}(p)$ otherwise;
- $d'(p) = d(p)$ if $d(p) \neq ?$, and $d'(p) = f_r(p)$ otherwise.

This means that if in round $\mathbf{ir}$, the value of $f_{\mathbf{ir}}(p)$ was ?, then the process p retains its value of the *inp* onto the next phase; otherwise the process p updates its value of *inp* to $f_{\mathbf{ir}}(p)$. The value of *dec* cannot be updated, it can only be set if it has not been set before. For setting the value of *dec*, the value from the last round is used.

An *execution* is a sequence of phase transitions. An *execution of an algorithm respecting a communication predicate* $(\mathsf{G}\overline{\psi}) \wedge (\mathsf{F}(\psi^1 \wedge \mathsf{F}(\psi^2 \wedge \ldots (\mathsf{F}\psi^k) \ldots)))$ is an infinite sequence:

$$(f_0, d_0) \xrightarrow{\overline{\psi}}{}^{*} (f_1, d_1) \xrightarrow{\psi \wedge \psi^1} (f'_1, d'_1) \cdots \xrightarrow{\overline{\psi}}{}^{*} (f_k, d_k) \xrightarrow{\psi \wedge \psi^k} (f'_k, d'_k) \xrightarrow{\overline{\psi}}{}^{\omega} \cdots$$

where $\xrightarrow{\overline{\psi}}{}^{*}$ stands for a finite sequence of $\xrightarrow{\overline{\psi}}$ transitions, and $\xrightarrow{\overline{\psi}}{}^{\omega}$ for an infinite sequence. For every execution there is some fixed n determining the number of processes, f_0 is any n-tuple of values without ?, and d_0 is the n-tuple of ? values. Observe that the size of the first tuple determines the size of every other tuple. By definition of transitions, there is always a transition from every configuration, so an execution cannot block. Thus we can think of every execution as being infinite.

▶ **Definition 1** (Consensus problem). *An algorithm has* agreement property *if for every number of processes n, and for every state (f, d) reachable by an execution of the algorithm, for all processes p_1 and p_2, either $d(p_1) = d(p_2)$ or one of the two values is ?. An algorithm has* termination property *if for every n, and for every execution there is a state (f, d) on this execution with $d(p) \neq ?$ for all $p = 1, \ldots, n$. An algorithm* solves consensus *if it has agreement and termination properties.*

▶ Remark 2. Normally, the consensus problem also requires irrevocability and integrity properties, but these are always guaranteed by the semantics: once set, a process cannot change its *dec* value, and a variable can be set only to one of the values that has been received.

▶ Remark 3. The original definition of the Heard-Of model is open ended: it does not limit possible forms of a communication predicate, conditions, or operations. Clearly, for the kind of result we present here, we need to fix them.

▶ Remark 4. In the original definition processes are allowed to have identifiers. We do not need them for the set of operations we consider. Later we will add coordinators without referring to identifiers. This is a relatively standard way of avoiding identifiers while having reasonable expressivity.

3 A characterization for the core language

We present a characterization of all the algorithms in our language that solve consensus. In later sections we will extend it to include timestamps and coordinators. As it will turn out, for our analysis we will need to consider only two values a, b with a fixed order between them: we take a smaller than b. This order influences the semantics of instructions: the result of min is a on a multiset containing at least one a; the result of smor is a on a multiset with the same number of a's and b's. Because of this asymmetry we mostly focus on the number of b's in a tuple. In our analysis we will consider tuples of the form $bias(\theta)$ for $\theta < 1$, i.e., a tuple where we have n processes (for some large enough n), out of which $\theta \cdot n$ of them have their value set to b; and the remaining ones to a. The tuple containing only b's (resp. only a's) is called $solo$ (resp. $solo^a$).

We show that there is essentially one way to solve consensus. The text of the algorithm together with the form of the global predicate determines a threshold $\overline{thr}$. We prove that in the language we consider here, there should be a *unifier phase* which guarantees that the tuple of *inp* values after the phase belongs to one of the following four types: $solo$, $solo^a$, $bias(\theta)$, or $bias(1-\theta)$ where $\theta \geq \overline{thr}$. Intuitively, this means that there is a dominant value in the tuple. This phase should be followed by a *decider phase* which guarantees that if the tuple of *inp* values is of one of the above mentioned types, then all the processes decide. While this ensures termination, agreement is ensured by proving that some structural properties on the algorithm should always hold.

Before stating the characterization, we will make some observations that allow us to simplify the structure of an algorithm, and in consequence simplify the statements.

It is easy to see that in our languge we can assume that the list of conditional instructions in each round can have at most one `uni` conditional followed by a sequence of `mult` conditionals with non-increasing thresholds:

$$\texttt{if uni}(\mathsf{H}) \wedge |\mathsf{H}| > thr^i_u \cdot |\Pi| \texttt{ then } x := \texttt{op}^i_u(\mathsf{H})$$
$$\texttt{if mult}(\mathsf{H}) \wedge |\mathsf{H}| > thr^{i,1}_m \cdot |\Pi| \texttt{ then } x := \texttt{op}^i_m(\mathsf{H})$$
$$\vdots$$
$$\texttt{if mult}(\mathsf{H}) \wedge |\mathsf{H}| > thr^{i,k}_m \cdot |\Pi| \texttt{ then } x := \texttt{op}^i_m(\mathsf{H})$$

We use superscript i to denote the round number: so thr^1_u is a threshold associated to `uni` instruction in the first round, etc. If round i does not have a `uni` instruction, then thr^1_u will be -1. For the sake of brevity, $thr^{i,k}_m$ will always denote the minimal threshold appearing in any of the `mult` instructions in round i and -1 if no `mult` instructions exist in round i.

We fix a *communication predicate*:

$$(\mathsf{G}\overline{\psi}) \wedge (\mathsf{F}(\psi^1 \wedge \mathsf{F}(\psi^2 \wedge \ldots (\mathsf{F}\psi^k)\ldots))) \tag{1}$$

Without loss of generality we can assume that every sporadic predicate implies the global predicate; in consequence, $\overline{\psi} \wedge \psi^i$ is equivalent to ψ^i. Recall that each of $\overline{\psi}, \psi^1, \ldots, \psi^k$ is an r-tuple of conjunctions of atomic predicates. We write $\psi\lfloor_i$ for the i-th element of the tuple and so ψ is $(\psi\lfloor_1, \ldots, \psi\lfloor_r)$. By $thr_i(\psi)$ we denote the threshold constant appearing in the predicate $\psi\lfloor_i$, i.e., if $\psi\lfloor_i$ has φ_{thr} as a conjunct, then $thr_i(\psi) = thr$, if it has no such conjunct then $thr_i(\psi) = -1$. We call $\psi\lfloor_i$ an *equalizer* if it has $\varphi_=$ as a conjunct. In this case we also say that ψ has an equalizer.

Recall (cf. page 2) that a transition $f \overset{\psi}{\Longrightarrow}_i f'$ for a round i under a phase predicate ψ is possible when there is a tuple of multisets $(\mathsf{H}_1, \ldots, \mathsf{H}_n) \vDash \psi\lfloor_i$ such that for all $p = 1, \ldots, n$: $\mathsf{H}_p \in mset(f)$ and $f'(p) = \mathtt{update}_i(\mathsf{H}_p)$.

▶ **Definition 5.** *A round i is* preserving *w.r.t. ψ iff one of the three conditions hold: (i) it does not have an* `uni` *instruction, (ii) it does not have a* `mult` *instruction, or (iii) $thr_i(\psi) < \max(thr^i_u, thr^{i,k}_m)$. Otherwise the round is* non-preserving. *The round is* solo safe *w.r.t. ψ if $0 \leq thr^i_u \leq thr_i(\psi)$.*

If i is a preserving round, then there exists a tuple f such that $? \notin mset(f)$ and such that a transition $f \overset{\psi}{\Longrightarrow}_i f'$ is possible for f' a tuple consisting solely of ?. The consequence of such a transition is that *inp* is not updated in the phase, i.e., old values of *inp* are *preserved*. On the other extreme, if all transitions in the phase are non-preserving then all *inp* values are necessarily updated by the phase. Finally, a solo safe round cannot alter the *solo* state, i.e., *solo* $\overset{\psi}{\Longrightarrow}_i$ *solo* is the only transition possible from *solo*.

▶ Remark 6. Suppose rounds $1, \ldots, i-1$ are non-preserving under $\overline{\psi}$, the global predicate. In this situation, since $? \notin mset(f)$, if $f \overset{\overline{\psi}\lfloor_1}{\Longrightarrow}_1 f_1 \overset{\overline{\psi}\lfloor_2}{\Longrightarrow}_2 \ldots \overset{\overline{\psi}\lfloor_{i-1}}{\Longrightarrow}_{i-1} f_{i-1}$ then $? \notin mset(f_{i-1})$. Hence, no heard-of multi-set H constructed from f_{i-1} can have ? value. Notice that every process is bound to receive a heard-of set of size at least $thr_i(\overline{\psi})$ in round i. For a sake of example, suppose $thr_i(\overline{\psi}) > thr^{i,2}_m$. The semantics then guarantees that every heard-of set sent during the i^{th} round either satisfies the `uni` instruction, or one of the first two `mult` instructions, or no instruction at all. Hence, in such a case all the `mult` instructions except the first two can be removed from the description of round i as they will be never executed. This implies that we can adopt the following assumption.

▶ **Assumption 1.** *For every round i, if rounds $1, \ldots, i-1$ are non-preserving under $\overline{\psi}$ then*

$$\begin{cases} thr^i_u \geq thr_i(\overline{\psi}) & \text{if round } i \text{ has } \mathtt{uni} \text{ instruction} \\ thr^{i,k}_m \geq thr_i(\overline{\psi}) & \text{if round } i \text{ has } \mathtt{mult} \text{ instruction} \end{cases} \tag{2}$$

We put some restrictions on the form of algorithms we consider in our characterization. They greatly simplify the statements, and as we argue, are removing cases that are not that interesting anyway.

▶ **Proviso 1.** *We adopt the following additional syntactic restrictions:*

- *We require that the global predicate does not have an equalizer.*
- *We assume that there is no* `mult` *instruction in the round* $\mathbf{ir} + 1$.

Concerning the first of the above requirements, if the global predicate has an equalizer then it is quite easy to construct an algorithm for consensus because equalizer guarantees that in a given round all the processes receive the same value. The characterization below

can be extended to this case but would require to mention it separately in all the statements. Concerning the second requirement, We can show that if such a `mult` instruction exists then either the algorithm violates consensus, or the instruction will never be fired in any execution of the algorithm and so it can be removed without making an algorithm incorrect.

In order to state our characterization we need to give formal definitions of concepts we have discussed at the beginning of the section.

▶ **Definition 7.** *The* border threshold *is* $\overline{thr} = \max(1 - thr_u^1, 1 - thr_m^{1,k}/2)$.

▶ **Definition 8.** *A predicate* ψ *is a*

- Decider*, if all rounds are solo safe w.r.t.* ψ
- Unifier*, if the three conditions hold:*
  - $thr_1(\psi) \geq thr_m^{1,k}$ *and either* $thr_1(\psi) \geq thr_u^1$ *or* $thr_1(\psi) \geq \overline{thr}$*,*
  - *there exists* i *such that* $1 \leq i \leq \mathbf{ir}$ *and* $\psi|_i$ *is an equalizer,*
  - *rounds* $2, \ldots, i$ *are non-preserving w.r.t.* ψ *and rounds* $i+1, \ldots \mathbf{ir}$ *are solo-safe w.r.t.* ψ

Finally, we list some syntactic properties of algorithms that, as we will see later, imply the agreement property.

▶ **Definition 9.** *An algorithm is* syntactically safe *when:*

1. *First round has a* `mult` *instruction.*
2. *Every round has a* `uni` *instruction.*
3. *In the first round the operation in every* `mult` *instruction is* smor*.*
4. $thr_m^{1,k}/2 \geq 1 - thr_u^{\mathbf{ir}+1}$*, and* $thr_u^1 \geq 1 - thr_u^{\mathbf{ir}+1}$*.*

Recall that $\psi^1, \ldots, \psi^k$ are the set of sporadic predicates from the communication predicate. Without loss of generality we can assume that there is at least one sporadic predicate; at a degenerate case it is always possible to take a sporadic predicate that is the same as the global predicate. With these definitions we can state our characterization:

▶ **Theorem 10.** *Consider algorithms in the core language satisfying syntactic constraints from Assumption 1 and Proviso 1. An algorithm solves consensus iff it is syntactically safe according to Definition 9, and it satisfies the condition:*

T *There is* $i \leq j$ *such that* ψ^i *is a unifier and* ψ^j *is a decider.*

A two value principle is a corollary from the proof of the above theorem: an algorithm solves consensus iff it solves consensus for two values. Indeed, it turns out that it is enough to work with three values a, b, and ? standing for undefined. The proof considers separately safety and liveness aspects of the consensus problem. Notice that the properties from Definition 9 intervene also in the proof of termination.

▶ **Lemma 11.** *An algorithm violating structural properties from Definition 9 cannot solve consensus. An algorithm with the structural properties has the agreement property.*

▶ **Lemma 12.** *An algorithm with the structural properties from Definition 9 has the termination property iff it satisfies condition T from Theorem 10.*

4 A characterization for algorithms with timestamps

We extend our characterization to algorithms with timestamps. Now, variable *inp* stores not only the value but also a timestamp, that is the number of the last phase at which *inp* was updated. These timestamps are used in the first round, as a process considers only values

with the most recent timestamp. The syntax is the same as before except that we introduce a new operation, called maxts, that must be used in the first round and nowhere else. So the form of the first round becomes:

```
send (inp, ts)
    if cond_1^1(H) then x_1 := maxts(H);
    ⋮
    if cond_1^l(H) then x_1 := maxts(H);
```

The semantics of transitions for rounds and phases needs to take into account timestamps. The semantics changes only for the first round; its form becomes $(f,t) \xRightarrow{\varphi} f'$, where t is a vector of timestamps (n-tuple of natural numbers). Timestamps are ignored by communication predicates and conditions, but are used in the update operation. The operation maxts(H) returns the smallest among values with the most recent timestamp in H.

The form of a phase transition changes to $(f,t,d) \xrightarrow{\psi} (f',t',d')$. Value $t(p)$ is the timestamp of the last update of *inp* of process p (whose value is $f(p)$). We do not need to keep timestamps for d since the value of *dec* can be set only once. Phase transitions are defined as before, taking into account the above mentioned change for the first round transition, and the fact that in the round **ir** when *inp* is updated then so is its timestamp. Some examples of algorithms with timestamps are presented in Section 7.

As in the case of the core language, without loss of generality we can assume conditions from Assumption 1. Concerning Proviso 1, we assume almost the same conditions, but now the second one refers to the round **ir** and not to the round **ir** + 1, and is a bit stronger.

▶ **Proviso 2.** *We adopt the following syntactic restrictions:*

- *We require that the global predicate does not have an equalizer.*
- *We assume that there is no* `mult` *instruction in the round* **ir**, *and that* $thr_u^{\mathbf{ir}} \geq 1/2$.

The justification for the first restriction is as before. Concerning the second restriction, we can prove that if these two assumptions do not hold then either the algorithm violates consensus, or we can remove the `mult` instruction and increase $thr_u^{\mathbf{ir}}$ without making an algorithm incorrect.

Our characterization resembles the one for the core language. The structural conditions get slightly modified: the condition on constants is weakened, and there is no need to talk about smor operations in the fist round.

▶ **Definition 13.** *An algorithm is* syntactically t-safe *when:*

1. *Every round has a* `uni` *instruction.*
2. *First round has a* `mult` *instruction.*
3. $thr_m^{1,k} \geq 1 - thr_u^{\mathbf{ir}+1}$ *and* $thr_u^1 \geq 1 - thr_u^{\mathbf{ir}+1}$.

We consider the same shape of a communication predicate as in the case of the core language (1). A characterization for the case with timestamps uses a stronger version of a unifier that we define now. The intuition is that we do not have $\overline{thr}$ constant because of maxts operations in the first round. In other words, the conditions are the same as before but when taking $\overline{thr} > 1$.

▶ **Definition 14.** *A predicate ψ is a* strong unifier *ψ if it is a unifier in a sense of Definition 8 and $thr_u^1 \leq thr_1(\psi)$.*

Modulo the above two changes, the characterization stays the same.

▶ **Theorem 15.** *Consider algorithms in the language with timestamps satisfying syntactic constraints from Assumption 1 and Proviso 2. An algorithm satisfies consensus iff it is syntactically t-safe according to Definition 13, and it satisfies:*
sT *There are $i \leq j$ such that ψ^i is a strong unifier and ψ^j is a decider.*

5 A characterization for algorithms with coordinators

We consider algorithms equipped with coordinators. The novelty is that we can now have rounds where there is a unique process that receives values from other processes, as well as rounds where there is a unique process that sends values to other processes. For this we extend the syntax by introducing a round type that can be: `every`, `lr` (leader receive), or `ls` (leader-send):

- A round of type `every` behaves as before.
- In a round of type `lr` only one arbitrarily selected process receives values.
- In a round of type `ls`, the process selected in the immediately preceding `lr` round sends its value to all other processes.

If an `ls` round is not preceded by an `lr` round then an arbitrarily chosen process sends its value. We assume that every `lr` round is immediately followed by an `ls` round, because otherwise the `lr` round would be useless. We also assume that *inp* and *dec* are not updated during `lr` rounds, as only one process is active in these rounds.

For `ls` rounds we introduce a new communication predicate. The predicate $\varphi_{\texttt{ls}}$ says that the leader successfully sends its message to everybody; it makes sense only for `ls` rounds.

These extensions of the syntax are reflected in the semantics. For convenience we introduce two new names for tuples: one^b is a tuple where all the entries are ? except for one entry which is b; similarly for one^a. Abusing the notation we also write $one^?$ for $solo^?$, namely the tuple consisting only of ? values.

Let us define the semantics of `lr` and `ls` rounds. If i-th round is of type `lr`, we have a transition $f \overset{\psi}{\Longrightarrow}_i one^d$ for every $d \in \mathrm{fire}_i(f, \psi)$. In particular, if $? \in \mathrm{fire}_i(f, \psi)$ then $f \overset{\varphi}{\Longrightarrow}_i solo^?$ is possible.

Suppose i-th round is of type `ls`. If $\psi|_i$ contains $\varphi_{\texttt{ls}}$ as a conjunct then

$$one^d \overset{\psi}{\Longrightarrow}_i solo^d \qquad \text{if round } (i-1) \text{ is of type } \texttt{lr}$$
$$f \overset{\psi}{\Longrightarrow}_i solo^d \quad \text{for } d \in set(f) \qquad \text{otherwise}$$

When $\psi|_i$ does not contain $\varphi_{\texttt{ls}}$ then independently of the type of the round $(i-1)$ we have $f \overset{\psi}{\Longrightarrow}_i f'$ for every $d \in set(f)$ and f' such that $set(f') \subseteq \{d, ?\}$.

We consider the same shape of a communication predicate as in the case of the core language (1).

The semantics allows us to adopt some more simplifying assumptions about the syntax of the algorithm, and the form of the communication predicate.

▶ **Assumption 2.** *We assume that* `ls` *rounds do not have a* `mult` *instruction. Indeed, from the above semantics it follows that* `mult` *instruction is never used in a round of type* `ls`*. It also does not make much sense to use $\varphi_{\texttt{ls}}$ in rounds other than of type* `ls`*. So to shorten some definitions we require that $\varphi_{\texttt{ls}}$ can appear only in communication predicates for* `ls`*-rounds. For similar reasons we require that $\varphi_=$ predicate is not used in* `ls`*-rounds. As we have observed in the first paragraph, we can assume that neither round* **ir** *nor the last round are of type* `lr`*.*

The notions of preserving and solo-safe rounds get adapted to the new syntax.

▶ **Definition 16.** *A round of type* `ls` *is* c-solo-safe *w.r.t.* ψ *if* ψ_i *has* $\varphi_{\texttt{ls}}$ *as a conjunct, it is* c-preserving *otherwise. A round of type other than* `ls` *is* c-preserving *or* c-solo-safe *w.r.t* ψ *if it is so in the sense of Definition 5.*

▶ **Definition 17.** *A* c-equalizer *is a conjunction containing a term of the form* $\varphi_=$ *or* $\varphi_{\texttt{ls}}$.

▶ **Proviso 3.** *We assume the same conditions as in Proviso 8, but using the concepts of c-equalizers instead of equalizers.*

To justify the proviso we prove that `mult` instruction in round $\mathbf{ir}+1$ cannot be useful. Assumption 1 is also updated to using the notion of c-preserving instead of preserving. We restate it for convenience.

▶ **Assumption 3.** *For every round* i*, if rounds* $1,\ldots,i-1$ *are non-c-preserving under* $\overline{\psi}$ *then*

$$\begin{cases} thr_u^i \geq thr_i(\overline{\psi}) & \text{if round } i \text{ has } \texttt{uni} \text{ instruction} \\ thr_m^{i,k} \geq thr_i(\overline{\psi}) & \text{if round } i \text{ has } \texttt{mult} \text{ instruction} \end{cases} \tag{3}$$

Finally, the above modifications imply modifications of terms from Definition 8.

▶ **Definition 18.** *A predicate* ψ *is called a*

- c-decider*, if all rounds are c-solo safe w.r.t.* ψ*.*
- c-unifier*, if*
  - $thr_1(\psi) \geq thr_m^{1,k}$ *and either* $thr_1(\psi) \geq thr_u^1$ *or* $thr_1(\psi) \geq \overline{thr}$*,*
  - *there exists* i *such that* $1 \leq i \leq \mathbf{ir}$ *and* $\psi|_i$ *is an c-equalizer,*
  - *rounds* $2,\ldots,i$ *are non-c-preserving w.r.t.* ψ *and rounds* $i+1,\ldots\mathbf{ir}$ *are c-solo-safe w.r.t.* ψ*.*

With these modifications, we get an analog of Theorem 10 for the case with coordinators subject to the modified provisos as explained above.

▶ **Theorem 19.** *Consider algorithms in the language with timestamps satisfying syntactic constraints from Assumptions 2, 3 and Proviso 3. An algorithm satisfies consensus iff the first round and the* $(\mathbf{ir}+1)^{th}$ *round are not of type* `ls`*, it is syntactically safe according to Definition 9, and it satisfies the condition:*
cT *There are* $i \leq j$ *such that* ψ^i *is a c-unifier and* ψ^j *is a c-decider.*

6 A characterization for algorithms with coordinators and timestamps

Finally, we consider an extension of the core language with both coordinators and timestamps. Formally, we extend the coordinator model with timestamps in the same way we have extended the core model. So now *inp* variables store pairs (value, timestamp), and all the instructions in the first round are maxts (cf. page 10).

▶ **Proviso 4.** *We assume the same proviso as for timestamps; namely, Proviso 2, but using the notion of c-equalizer.*

As in the previous cases we justify our proviso by showing that the algorithm violating the second condition would not be correct or the condition could be removed.

The characterization is a mix of conditions from timestamps and coordinator cases.

▶ **Definition 20.** *A predicate ψ is a strong c-unifier if it is a c-unifier (cf. Definition 14) and $thr_u^1 \leq thr_1(\psi)$.*

▶ **Theorem 21.** *Consider algorithms in the language with timestamps satisfying syntactic constraints from Assumptions 2, 3 and Proviso 4. An algorithm satisfies consensus iff the first round and the $(\mathbf{ir}+1)^{th}$ round are not of type* `ls`*, it has the structural properties from Definition 13, and it satisfies:*

scT *There are $i \leq j$ such that ψ^i is a strong c-unifier and ψ^j is a c-decider.*

7 Examples

We apply the characterizations from the previous sections to some consensus algorithms studied in the literature, and their variants.

First, we can revisit the parametrized Algorithm 1 from page 4. This is an algorithm in the core language, and it depends on two thresholds. Theorem 10 implies that it solves consensus iff $thr_1/2 \geq 1 - thr_2$. In case of $thr_1 = thr_2 = 2/3$ we obtain the well known OneThird algorithm. But, for example, $thr_1 = 1/2$ and $thr_2 = 3/4$ are also possible solutions for this inequality. So Algorithm 1 solves consensus for these values of thresholds.

Because of the conditions on constants, $thr_m^{1,k}/2 \geq 1 - thr_u^{\mathbf{ir}+1}$ coming from Definition 9, it is not possible to have an algorithm in the core language where all constants are at most $1/2$. This answers a question from [11] for the language we consider here.

The above condition on constants is weakened to $thr_m^{1,k} \geq 1 - thr_u^{\mathbf{ir}+1}$ when we have timestamps. In this case indeed it is possible to use only $1/2$ thresholds [27].

When we have both timestamps and coordinators, we get variants of Paxos algorithm.

Algorithm 2 Paxos algorithm.

`send` (inp, ts) `lr`
  `if uni`(H) $\wedge$ |H| $> 1/2 \cdot |\Pi|$ `then` $x_1 :=$ maxts(H);
  `if mult`(H) $\wedge$ |H| $> 1/2 \cdot |\Pi|$ `then` $x_1 :=$ maxts(H);
`send` x_1 `ls`
  `if uni`(H) `then` $x_2 := inp :=$ smor(H);
`send` x_2 `lr`
  `if uni`(H) $\wedge$ |H| $> 1/2 \cdot |\Pi|$ `then` $x_3 :=$ smor(H);
`send` x_3 `ls`
  `if uni`(H) `then` $dec :=$ smor(H);
`Communication predicate:` $\mathsf{F}(\psi^1)$ where $\psi^1 := (\varphi_{1/2},\ \varphi_{\mathtt{ls}},\ \varphi_{1/2},\ \varphi_{\mathtt{ls}})$

The algorithm is correct by Theorem 21. One can observe that without modifying the code there is not much room for improvement in this algorithm. A decider phase is needed to solve consensus, and ψ_1 is a minimal requirement for a decider phase. A possible modification is to change the thresholds in the first round to, say, $1/3$ and in the third round to $2/3$ (both in the algorithm and in the communication predicate).

Algorithm 3 Three round Paxos algorithm.

`send` (inp, ts) `lr`
  `if` $\mathtt{uni}(\mathsf{H}) \wedge |\mathsf{H}| > 1/2 \cdot |\Pi|$ `then` $x_1 := \mathrm{maxts}(\mathsf{H})$;
  `if` $\mathtt{mult}(\mathsf{H}) \wedge |\mathsf{H}| > 1/2 \cdot |\Pi|$ `then` $x_1 := \mathrm{maxts}(\mathsf{H})$;
`send` x_1 `ls`
  `if` $\mathtt{uni}(\mathsf{H})$ `then` $x_2 := inp := \mathrm{smor}(\mathsf{H})$;
`send` x_2 `every`
  `if` $\mathtt{uni}(\mathsf{H}) \wedge |\mathsf{H}| > 1/2 \cdot |\Pi|$ `then` $dec := \mathrm{smor}(\mathsf{H})$;
`Communication predicate:` $\mathsf{F}(\psi^1)$ where $\psi^1 := (\varphi_{1/2},\ \varphi_{\mathtt{ls}},\ \varphi_{1/2})$

The three round Paxos presented above is also correct by Theorem 21. Once again it is possible to change constants in the first round to 1/3 and in the last round to 2/3 (both in the algorithm and in the communication predicate).

One can also wander about algorithms with coordinators but without timestamps. Here is a possibility that resembles three round Paxos:

Algorithm 4 Three round coordinator algorithm.

`send` (inp) `lr`
  `if` $\mathtt{uni}(\mathsf{H}) \wedge |\mathsf{H}| > 2/3 \cdot |\Pi|$ `then` $x_1 := \mathrm{smor}(\mathsf{H})$;
  `if` $\mathtt{mult}(\mathsf{H}) \wedge |\mathsf{H}| > 2/3 \cdot |\Pi|$ `then` $x_1 := \mathrm{smor}(\mathsf{H})$;
`send` x_1 `ls`
  `if` $\mathtt{uni}(\mathsf{H})$ `then` $x_2 := inp := \mathrm{smor}(\mathsf{H})$;
`send` x_2 `every`
  `if` $\mathtt{uni}(\mathsf{H}) \wedge |\mathsf{H}| > 2/3 \cdot |\Pi|$ `then` $dec := \mathrm{smor}(\mathsf{H})$;
`Communication predicate:` $\mathsf{F}(\psi)$ where $\psi := (\varphi_{2/3}, \varphi_{\mathtt{ls}}, \varphi_{2/3})$

The algorithm solves consensus by Theorem 19. The constants are bigger than in Paxos because we do not have timestamps: the constraints on constants come from Definition 9, and not from Definition 13. The advantage is that we do not need time-stamps, while keeping the same structure as for three-round Paxos.

It is possible to introduce more parameters in these algorithms to analyze for which choices of parameters they solve consensus.

8 Conclusions

We have characterized all algorithms solving consensus in a fragment of the Heard-Of model. We have aimed at a fragment that can express most important algorithms while trying to avoid ad hoc restrictions (c.f. proviso on page 8). The fragment covers algorithms considered in the context of verification [28, 10] with a notable exception of algorithms sending more than one variable. In this work we have considered only single phase algorithms while originally the model permits also to have initial phases. We believe that it is possible to extend the characterization to incorporate the initial phases, but this would further complicate the results and there are no well-know algorithms that use such phases. More severe and technically important restriction is that we allow to use only one variable at a time. In particular, it is not possible to send pairs of variables.

One curious direction of further research would be to list all "best" consensus algorithms under some external constraints; for example the constraints can come from some properties of an execution platform external to the Heard-Of model. This problem assumes that there

is some way to compare two algorithms. One guiding principle for such a measure could be efficient use of knowledge [30, 29]: at every step the algorithm does maximum it can do, given its knowledge of the state of the system.

This research is on the borderline between distributed computing and verification. From a distributed computing side it considers quite a simple model, but gives a characterization result. From a verification side, the systems are complicated because the number of processes is unbounded, there are timestamps, and interactions are based on a fraction of processes having a particular value. We do not advance on verification methods for such a setting. Instead, we observe that in the context considered here verification may be avoided. We believe that a similar phenomenon can appear also for other problems than consensus. It is also an intriguing question to explore how much we can enrich the current model and still get a characterization. We conjecture that a characterization is possible for an extension with randomness covering at least the Ben-Or algorithm. Of course, formalization of proofs, either in Coq or Isabelle, for such extensions would be very helpful.

References

1 Marcos K. Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Partial synchrony based on set timeliness. *Distributed Computing*, 25(3):249–260, 2012. doi:10.1007/s00446-012-0158-8.

2 Benjamin Aminof, Sasha Rubin, Ilina Stoilkovska, Josef Widder, and Florian Zuleger. Parameterized model checking of synchronous distributed algorithms by abstraction. In Isil Dillig and Jens Palsberg, editors, *Verification, Model Checking, and Abstract Interpretation - 19th International Conference, VMCAI 2018*, volume 10747 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2018. doi:10.1007/978-3-319-73721-8_1.

3 A.R. Balasubramanian and Igor Walukiewicz. Characterizing consensus in the heard-of model. arXiv:2004.09621.

4 Nathalie Bertrand, Igor Konnov, Marijana Lazic, and Josef Widder. Verification of randomized consensus algorithms under round-rigid adversaries. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory*, volume 140 of *LIPIcs*, pages 33:1–33:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.33.

5 Martin Biely, Josef Widder, Bernadette Charron-Bost, Antoine Gaillard, Martin Hutle, and André Schiper. Tolerating corrupted communication. In Indranil Gupta and Roger Wattenhofer, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007*, pages 244–253. ACM, 2007. doi:10.1145/1281100.1281136.

6 Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, 1996. doi:10.1145/234533.234549.

7 Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996. doi:10.1145/226643.226647.

8 Mouna Chaouch-Saad, Bernadette Charron-Bost, and Stephan Merz. A reduction theorem for the verification of round-based distributed algorithms. In Olivier Bournez and Igor Potapov, editors, *Reachability Problems, 3rd International Workshop, RP 2009*, volume 5797 of *Lecture Notes in Computer Science*, pages 93–106. Springer, 2009. doi:10.1007/978-3-642-04420-5_10.

9 Bernadette Charron-Bost, Henri Debrat, and Stephan Merz. Formal verification of consensus algorithms tolerating malicious faults. In Xavier Défago, Franck Petit, and Vincent Villain, editors, *Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, SSS 2011*, volume 6976 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 2011. doi:10.1007/978-3-642-24550-3_11.

10 Bernadette Charron-Bost and Stephan Merz. Formal verification of a consensus algorithm in the heard-of model. *Int. J. Software and Informatics*, 3(2-3):273–303, 2009. URL: http://www.ijsi.org/ch/reader/view_abstract.aspx?file_no=273&flag=1.

11 Bernadette Charron-Bost and André Schiper. The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 2009.

12 Flaviu Cristian and Christof Fetzer. The timed asynchronous distributed system model. *IEEE Trans. Parallel Distrib. Syst.*, 10(6):642–657, 1999. doi:10.1109/71.774912.

13 Andrei Damian, Cezara Dragoi, Alexandru Militaru, and Josef Widder. Communication-closed asynchronous protocols. In *CAV (2)*, volume 11562 of *Lecture Notes in Computer Science*, pages 344–363. Springer, 2019. doi:10.1007/978-3-030-25543-5_20.

14 Henri Debrat and Stephan Merz. Verifying fault-tolerant distributed algorithms in the heard-of model. *Archive of Formal Proofs*, 2012, 2012. URL: https://www.isa-afp.org/entries/Heard_Of.shtml.

15 Cezara Dragoi, Thomas A. Henzinger, Helmut Veith, Josef Widder, and Damien Zufferey. A logic-based framework for verifying consensus algorithms. In Kenneth L. McMillan and Xavier Rival, editors, *Verification, Model Checking, and Abstract Interpretation VMCAI 2014*, volume 8318 of *Lecture Notes in Computer Science*, pages 161–181. Springer, 2014. doi:10.1007/978-3-642-54013-4_10.

16 Cezara Dragoi, Thomas A. Henzinger, and Damien Zufferey. Psync: a partially synchronous language for fault-tolerant distributed algorithms. In Rastislav Bodík and Rupak Majumdar, editors, *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016*, pages 400–415. ACM, 2016. doi:10.1145/2837614.2837650.

17 Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988. doi:10.1145/42282.42283.

18 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985. doi:10.1145/3149.214121.

19 Felix C. Freiling, Rachid Guerraoui, and Petr Kuznetsov. The failure detector abstraction. *ACM Comput. Surv.*, 43(2):9:1–9:40, 2011. doi:10.1145/1883612.1883616.

20 Eli Gafni. Round-by-round fault detectors: Unifying synchrony and asynchrony (extended abstract). In Brian A. Coan and Yehuda Afek, editors, *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98*, pages 143–152. ACM, 1998. doi:10.1145/277697.277724.

21 Rachid Guerraoui and Michel Raynal. The alpha of indulgent consensus. *Comput. J.*, 50(1):53–67, 2007. doi:10.1093/comjnl/bxl046.

22 Michel Hurfin, Achour Mostéfaoui, and Michel Raynal. A versatile family of consensus protocols based on chandra-toueg's unreliable failure detectors. *IEEE Trans. Computers*, 51(4):395–408, 2002. doi:10.1109/12.995450.

23 Igor Konnov, Helmut Veith, and Josef Widder. SMT and POR beat counter abstraction: Parameterized model checking of threshold-based distributed algorithms. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015*, volume 9206 of *Lecture Notes in Computer Science*, pages 85–102. Springer, 2015. doi:10.1007/978-3-319-21690-4_6.

24 Igor V. Konnov, Marijana Lazic, Helmut Veith, and Josef Widder. A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017*, pages 719–734. ACM, 2017. URL: http://dl.acm.org/citation.cfm?id=3009860.

25 Igor V. Konnov, Helmut Veith, and Josef Widder. On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. *Inf. Comput.*, 252:95–109, 2017. doi:10.1016/j.ic.2016.03.006.

26 Jure Kukovec, Igor Konnov, and Josef Widder. Reachability in parameterized systems: All flavors of threshold automata. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018*, volume 118 of *LIPIcs*, pages 19:1–19:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.19.

27 Ognjen Maric. *Formal Verification of Fault-Tolerant Systems*. PhD thesis, ETH Zurich, 2017.

28 Ognjen Maric, Christoph Sprenger, and David A. Basin. Cutoff bounds for consensus algorithms. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017*, volume 10427 of *Lecture Notes in Computer Science*, pages 217–237. Springer, 2017. doi:10.1007/978-3-319-63390-9_12.

29 Yoram Moses. Knowledge in distributed systems. In *Encyclopedia of Algorithms*, pages 1051–1055. Springer, 2016. doi:10.1007/978-1-4939-2864-4_606.

30 Yoram Moses and Sergio Rajsbaum. A layered analysis of consensus. *SIAM J. Comput.*, 31(4):989–1021, 2002. doi:10.1137/S0097539799364006.

31 Achour Mostéfaoui and Michel Raynal. Solving consensus using chandra-toueg's unreliable failure detectors: A general quorum-based approach. In Prasad Jayanti, editor, *Distributed Computing, 13th International Symposium*, volume 1693 of *Lecture Notes in Computer Science*, pages 49–63. Springer, 1999. doi:10.1007/3-540-48169-9_4.

32 Michel Raynal and Julien Stainer. Synchrony weakened by message adversaries vs asynchrony restricted by failure detectors. In Panagiota Fatourou and Gadi Taubenfeld, editors, *ACM Symposium on Principles of Distributed Computing, PODC '13*, pages 166–175. ACM, 2013. doi:10.1145/2484239.2484249.

33 Olivier Rütti, Zarko Milosevic, and André Schiper. Generic construction of consensus algorithms for benign and byzantine faults. In *Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2010*, pages 343–352. IEEE Computer Society, 2010. doi:10.1109/DSN.2010.5544299.

34 Yee Jiun Song, Robbert van Renesse, Fred B. Schneider, and Danny Dolev. The building blocks of consensus. In Shrisha Rao, Mainak Chatterjee, Prasad Jayanti, C. Siva Ram Murthy, and Sanjoy Kumar Saha, editors, *Distributed Computing and Networking, 9th International Conference, ICDCN 2008*, volume 4904 of *Lecture Notes in Computer Science*, pages 54–72. Springer, 2008. doi:10.1007/978-3-540-77444-0_5.

35 Ilina Stoilkovska, Igor Konnov, Josef Widder, and Florian Zuleger. Verifying safety of synchronous fault-tolerant algorithms by bounded model checking. In Tomás Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019*, volume 11428 of *Lecture Notes in Computer Science*, pages 357–374. Springer, 2019. doi:10.1007/978-3-030-17465-1_20.

36 Marcelo Taube, Giuliano Losa, Kenneth L. McMillan, Oded Padon, Mooly Sagiv, Sharon Shoham, James R. Wilcox, and Doug Woos. Modularity for decidability of deductive verification with applications to distributed systems. In Jeffrey S. Foster and Dan Grossman, editors, *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018*, pages 662–677. ACM, 2018. doi:10.1145/3192366.3192414.

37 Tatsuhiro Tsuchiya and André Schiper. Verification of consensus algorithms using satisfiability solving. *Distributed Computing*, 23(5-6):341–358, 2011. doi:10.1007/s00446-010-0123-3.

38 Robbert van Renesse, Nicolas Schiper, and Fred B. Schneider. Vive la différence: Paxos vs. viewstamped replication vs. zab. *IEEE Trans. Dependable Sec. Comput.*, 12(4):472–484, 2015. doi:10.1109/TDSC.2014.2355848.

39 Klaus von Gleissenthall, Nikolaj Bjørner, and Andrey Rybalchenko. Cardinalities and universal quantifiers for verifying parameterized systems. In Chandra Krintz and Emery Berger, editors, *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016*, pages 599–613. ACM, 2016. doi:10.1145/2908080.2908129.

40 Josef Widder and Ulrich Schmid. The theta-model: achieving synchrony without clocks. *Distributed Computing*, 22(1):29–47, 2009. doi:10.1007/s00446-009-0080-x.

41 James R. Wilcox, Doug Woos, Pavel Panchekha, Zachary Tatlock, Xi Wang, Michael D. Ernst, and Thomas E. Anderson. Verdi: a framework for implementing and formally verifying distributed systems. In David Grove and Steve Blackburn, editors, *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 357–368. ACM, 2015. doi:10.1145/2737924.2737958.

42 Doug Woos, James R. Wilcox, Steve Anton, Zachary Tatlock, Michael D. Ernst, and Thomas E. Anderson. Planning for change in a formal verification of the raft consensus protocol. In Jeremy Avigad and Adam Chlipala, editors, *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, pages 154–165. ACM, 2016. doi:10.1145/2854065.2854081.

A Classification of Weak Asynchronous Models of Distributed Computing

Javier Esparza
Technical University of Munich, Germany
esparza@in.tum.de

Fabian Reiter
LIGM, Université Gustave Eiffel, Marne-la-Vallée, France
fabian.reiter@gmail.com

Abstract

We conduct a systematic study of asynchronous models of distributed computing consisting of identical finite-state devices that cooperate in a network to decide if the network satisfies a given graph-theoretical property. Models discussed in the literature differ in the detection capabilities of the agents residing at the nodes of the network (detecting the set of states of their neighbors, or counting the number of neighbors in each state), the notion of acceptance (acceptance by halting in a particular configuration, or by stable consensus), the notion of step (synchronous move, interleaving, or arbitrary timing), and the fairness assumptions (non-starving, or stochastic-like). We study the expressive power of the combinations of these features, and show that the initially twenty possible combinations fit into seven equivalence classes. The classification is the consequence of several equi-expressivity results with a clear interpretation. In particular, we show that acceptance by halting configuration only has non-trivial expressive power if it is combined with counting, and that synchronous and interleaving models have the same power as those in which an arbitrary set of nodes can move at the same time. We also identify simple graph properties that distinguish the expressive power of the seven classes.

2012 ACM Subject Classification Theory of computation → Automata extensions; Theory of computation → Concurrency; Theory of computation → Distributed computing models

Keywords and phrases Asynchrony, Concurrency theory, Weak models of distributed computing

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.10

Related Version A full version with appendix is available at `https://arxiv.org/abs/2007.03291v1`.

Funding This work was supported by the ERC project PaVeS (Advanced Grant 787367).

Acknowledgements The authors thank Ahmed Bouajjani for many interesting discussions, and several anonymous reviewers for their helpful feedback.

1 Introduction

Distributed computing is increasingly interested in the study of networks of natural or artificial devices, like molecules, cells, microorganisms, or nano-robots. These devices have very limited computational and communication capabilities, and are indistinguishable. In particular, a device cannot recognize whether its current communication partner is the same as a past one. This stands in stark contrast to the devices of standard computer networks, which has motivated researchers to question the suitability of traditional distributed computing models for the study of these networks, and to propose new ones. Examples include population protocols [3, 1], chemical reaction networks [13], networked finite state machines [6], the weak models of distributed computing of [8], and the beeping model [5]. A survey discussing many of them, and more, can be found in [11].

All these models share several common features, introduced to capture the limitations of the devices [6]: the network can have an arbitrary topology; all nodes of the network have a finite number of states, independent of the size of the network or its topology; all nodes run the same protocol; and state changes only depend on the states of a bounded number of neighbors, again independent of the size of the network.

Unfortunately, despite this very substantial common ground, the models still differ in many aspects, which makes it hard to compare results across them, or decide which features are essential for a particular result. A study of the models allows one to identify four specific junctions at which they choose different paths:

- *Detection.* In some models, agents can only detect the *existence* of neighbors in a certain state [8]. In others, they can *count* their number, up to a fixed threshold [6, 8]. For example, in biological models, cells communicate by emitting special kinds of proteins, and detecting them; in some models the cells may detect the presence of the protein when its concentration exceeds a given threshold, while in others they are able to detect different concentration levels.
- *Acceptance.* Some models compute by *stable consensus*, which requires all nodes to eventually agree on the outcome of the computation (but the nodes do not need to *know* that consensus has been reached) [3, 1, 13], while others require the nodes to reach a consensus in a halting configuration [8]. Acceptance by stable consensus is computationally powerful, since it permits the algorithm designer to concentrate on ensuring that every bad input is eventually rejected; declaring all non-rejecting states accepting ensures that every good input is eventually accepted.
- *Selection.* In some models, at each step a scheduler chooses an arbitrary set of nodes to make a step [6, 12], while in others it is exactly one, or exactly one pair of neighboring nodes [3, 1, 13]. We call the latter *exclusive* or *interleaving* models. Intuitively, interleaving models are useful when it can be assumed that process steps are much faster than the time interval between them, while the former policy does not need this assumption. In addition, they help the algorithm designer, who can assume that agents act in mutual exclusion. (Examples where this is useful can be found in the proofs of Propositions 16 and 20.) Another common option for selection is the *synchronous* execution model [8], where all nodes are selected in each step. Again this can be helpful for designing algorithms, but it is incompatible with exclusive selection.
- *Fairness.* Some models use fairness assumptions designed to model or approximate stochastic behavior [3, 1, 13], while others choose minimal notions, like "all nodes make a step infinitely often", which only assume the absence of crash faults (see, e.g., [7, 9]). Stochastic-like assumptions are reasonable for biological or chemical models, but can be too strong for networks of artificial nodes, which may follow non-random execution policies. Stochastic models may be able to solve problems that cannot be solved with weaker fairness assumptions.

The goal of this paper is to explore the space of models spanned by the above parameters, and compare their computational power within a specific framework. For this we use *distributed automata*, a generic formalism for the description of finite-state distributed algorithms. Such an automaton consists of a set of rules that tell the nodes of a graph how to change their state depending on the states of their neighbors. Intuitively, the automaton describes an algorithm that allows the nodes of an input graph to decide, in a distributed way, whether the graph satisfies a given property. The computational power of a class of distributed automata is then given by the class of graph languages recognized by the automata in the class, or, in other words, by the graph properties that the class of automata can decide.

We start with twenty classes of distributed automata, and show that with respect to their computational power, they fall into seven different classes. This reduction is a consequence of two results presented in this paper: (1) acceptance by halting configuration only has non-trivial expressive power if it is combined with counting; (2) both interleaving and synchronous selection have the same power as liberal selection where arbitrarily many nodes can move at the same time (and therefore, one can design an automaton in an interleaving or synchronous model, which is less error prone, and then translate it to a liberal model). Some of the simulations we design to prove the results are of independent interest. In particular, we give explicit constructions showing how to simulate interleaving models by non-interleaving ones.

The paper is organized as follows. Section 2 introduces distributed automata and their variants. Sections 3 to 5 show that the variants collapse to at most the seven equivalence classes mentioned above. Section 6 contains separation results showing that the seven classes are different. Finally, Section 7 presents further results on their expressive power. Proofs missing or only sketched can be found in the appendix of the arXiv version.

2 A taxonomy of distributed automata

Given sets X, Y, we denote by 2^X the power set of X, and by X^Y the set of functions $Y \to X$. We define $[m:n] \coloneqq \{i \in \mathbb{Z} \mid m \leq i \leq n\}$ and $[n] \coloneqq [0:n]$, for any $m, n \in \mathbb{Z}$ such that $m \leq n$. Angle brackets indicate excluded endpoints, e.g., $\langle m:n] \coloneqq [m-1:n]$ and $[n\rangle \coloneqq [0:n-1]$.

Let Λ be a finite set. A *(Λ-labeled, undirected) graph* is a triple $G = (V, E, \lambda)$, where V is a finite nonempty set of *nodes*, E is a set of undirected *edges* of the form $e = \{u, v\} \subseteq V$ such that $u \neq v$, and $\lambda \colon V \to \Lambda$ is a *labeling*. Isomorphic graphs are considered to be equal. **Convention**: Throughout the paper, all graphs have at least two nodes and are connected.

2.1 Distributed automata

Distributed automata take a graph as input, and either accept or reject it. To define them we first introduce distributed machines.

Distributed machines. Let Λ be a finite set of symbols and let $\beta \in \mathbb{N}_+$. A *(distributed) machine* with *input alphabet* Λ and *counting bound* β is a tuple $M = (Q, \delta_0, \delta, Y, N)$, where Q is a finite set of *states*, $\delta_0 \colon \Lambda \to Q$ is an *initialization function*, $\delta \colon Q \times [\beta]^Q \to Q$ is a *transition function*, and $Y, N \subseteq Q$ are two sets of *accepting* and *rejecting* states, respectively. The function δ updates the state of a node v based on the number of neighbors v has in each state, but it can only detect if v has $0, 1, \ldots, (\beta - 1)$, or at least β neighbors in a given state.

Selections, schedules, configurations, runs, and acceptance. A *selection* of a Λ-labeled graph $G = (V, E, \lambda)$ is a set $S \subseteq V$, and a *schedule* of G is an infinite sequence of selections $\sigma = (S_0, S_1, S_2, \ldots) \in (2^V)^\omega$. Intuitively, the selection S_t is the set of nodes activated by the scheduler at time t.

Let $M = (Q, \delta_0, \delta, Y, N)$ be a distributed machine with input alphabet Λ. A *configuration* of M on G is a mapping $C \colon V \to Q$. Given a configuration C and a node $v \in V$, we let $N_v^C \colon Q \to [\beta]$ denote the function that assigns to each state q the number of neighbors of v that are in state q up to threshold β, i.e., $\min\{\beta, \mathrm{card}(\{u \mid \{u, v\} \in E \wedge C(u) = q\})\}$. We call N_v^C the *β-bounded multiset* of states of v's neighbors.

For any selection S, we define the *successor configuration* of C via S to be the configuration $succ_\delta(C, S)$ that one obtains from C if all nodes in S evaluate the transition function δ simultaneously while the remaining nodes keep their current state. Formally, for all $v \in V$,

$$succ_\delta(C, S)(v) = \begin{cases} C(v) & \text{if } v \notin S \\ \delta\big(C(v), N_v^C\big) & \text{if } v \in S. \end{cases}$$

This brings us directly to the notion of a run. Given a schedule $\sigma = (S_0, S_1, S_2, \dots)$, the *run* of M on G scheduled by σ is the infinite sequence $\rho = (C_0, C_1, C_2, \dots)$ of configurations that are defined inductively as follows, where $\circ$ denotes function composition, and $t \in \mathbb{N}$:

$$C_0 = \delta_0 \circ \lambda \qquad \text{and} \qquad C_{t+1} = succ_\delta(C_t, S_t).$$

A configuration C is *accepting* if $C(v) \in Y$ for every $v \in V$, and *rejecting* if $C(v) \in N$ for every $v \in V$. A run $\rho = (C_0, C_1, C_2, \dots)$ of M on G is *accepting* if there is a time $t \in \mathbb{N}$ such that $C_{t'}$ is accepting for every $t' \geq t$. In other words, a run is accepting if from some time on it only visits accepting configurations. Similarly, ρ is *rejecting* if eventually all visited configurations are rejecting. Following [3], we call this *acceptance by stable consensus*.

Distributed automata. Not every schedule of a distributed machine models an execution; for example, schedules in which a node is never activated are usually considered illegal. We assume that distributed machines are controlled by a scheduler that ensures that the machine executes a legal run. Formally, a *scheduler* is a pair $\Sigma = (s, f)$, where s is a *selection constraint* that assigns to every graph $G = (V, E, \lambda)$ a set $s(G) \subseteq 2^V$ of *permitted selections* such that every node $v \in V$ occurs in at least one selection $S \in s(G)$, and f is a *fairness constraint* that assigns to every graph G a set $f(G) \subseteq s(G)^\omega$ of *fair schedules* of G. We call the runs with schedules in $f(G)$ *fair runs* (with respect to Σ).

A *distributed automaton* is a pair $A = (M, \Sigma)$, where M is a machine and Σ is a scheduler satisfying the *consistency condition*: for every graph G, either all fair runs of M on G are accepting, or all fair runs of M on G are rejecting. Intuitively, the machine is "immune" to the scheduler because its answer is independent of the scheduler's choices. This formalizes the standard notion of "asynchronous distributed algorithm". Notice that the consistency condition is a very strong *semantic* requirement. Although we will not do so in this paper, one can prove that it is undecidable whether a given pair (M, Σ) satisfies it.

A accepts G if every fair run of A on G is accepting, and *rejects G* otherwise. The language $L(A)$ recognized by A is the set of graphs it accepts. Two automata are equivalent if they recognize the same language.

2.2 Classifying distributed automata

We classify automata according to four criteria: detection capabilities, acceptance condition, selection constraint, and fairness constraint. The first two criteria concern the distributed machine, and the other two the scheduler. For each criterion, we investigate some of the major options that have been considered in the literature.

Detection. In some models, agents can only detect the existence of neighbors in a certain state. This corresponds to *non-counting machines*, i.e., machines with counting bound $\beta = 1$. Other models can detect the number of neighbors up to a higher bound [8].

Acceptance. As mentioned above, distributed machines accept by stable consensus. This is the acceptance condition of population protocols and chemical reaction networks [3, 1, 13]. Other models consider a notion of acceptance where each node explicitly decides to accept or reject [8]. This notion is captured by halting automata. A machine M is *halting* if its transition function does not allow the nodes to leave accepting or rejecting states, i.e., if $\delta(q, P) = q$ for every $q \in Y \cup N$ and every β-bounded multiset $P \in [\beta]^Q$. In halting machines, each node knows whether the input graph will be accepted the moment it enters an accepting or rejecting state. Indeed, by the consistency condition, in every fair run, eventually either all nodes occupy accepting states, or all nodes occupy rejecting states. Since nodes can never leave an accepting state once they enter it, each node that enters such a state knows that all other nodes will eventually do likewise. The same applies to rejecting states.

Selection. A scheduler $\Sigma = (s, f)$ is *synchronous* on $G = (V, E, \lambda)$ if $s(G) = \{V\}$. Intuitively, at every step all nodes make a move. Σ is *exclusive* or *interleaving-based* on G if $s(G) = \{\{v\} \mid v \in V\}$. Intuitively, at every step exactly one node makes a move, i.e., nodes execute steps in mutual exclusion. Finally, Σ is *liberal* on G if $s(G) = 2^V$. Intuitively, at every step an arbitrary subset of nodes makes a move. A scheduler is called *synchronous* if it is synchronous on every graph. Exclusive and liberal schedulers are defined analogously.

Fairness. A schedule $\sigma = (S_0, S_1, \ldots)$ of a graph G is *weakly fair* if for every node v of G, there exist infinitely many indices t such that $v \in S_t$. In other words, a schedule is weakly fair if every node is active infinitely often. A scheduler $\Sigma = (s, f)$ is *weakly fair* if $f(G)$ contains precisely the weakly-fair schedules of $s(G)^\omega$ for every graph G. This is the weakest fairness constraint one can impose on distributed automata; it only excludes runs in which a node crashes, and does not participate in the computation anymore.

With respect to a given selection constraint s, a schedule $\sigma = (S_0, S_1, \ldots) \in s(G)^\omega$ of a graph G is *strongly fair* if for every finite sequence $(T_0, \ldots, T_n) \in s(G)^*$ there exist infinitely many indices t such that $(S_t, S_{t+1}, \ldots, S_{t+n}) = (T_0, T_1, \ldots, T_n)$. Intuitively, strong fairness requires that every possible finite sequence of selections is scheduled infinitely often. If every node is selected independently with positive probability, stochastic schedules are almost surely strongly fair. A scheduler $\Sigma = (s, f)$ is *strongly fair* if for every graph G, the set $f(G)$ contains precisely the strongly-fair schedules of $s(G)^\omega$.

▶ Remark 1. Whether a schedule σ of a graph $G = (V, E, \lambda)$ is strongly fair or not depends on $s(G)$. For example, if $s(G) = \{V\}$, then the synchronous schedule V^ω is strongly fair, but if $s(G) = 2^V$, then it is not.

Our notion of strong fairness implies an apparently stronger one, used frequently in the literature, stating that in a strongly fair run, a sequence of configurations that is enabled infinitely often must occur infinitely often:

▶ **Lemma 2.** *Let A be a strongly fair automaton and $(D_0, \ldots, D_n)$ be a sequence of configurations of A such that D_{i+1} is the successor configuration of D_i via some selection S_i permitted by A, for $i \in [0 : n\rangle$. For any fair run $\rho = (C_0, C_1, \ldots)$ of A, if $C_i = D_0$ for infinitely many indices $i \in \mathbb{N}$, then $(C_j, \ldots, C_{j+n}) = (D_0, \ldots, D_n)$ for infinitely many indices $j \in \mathbb{N}$.*

The classification above yields 24 classes of automata (four classes of machines and six classes of schedulers). To assign mnemonics to them, we use lowercase letters for the most restrictive machine variants (i.e., non-counting and halting), and the same letters in uppercase for the other variants. With schedulers we proceed the other way round, assigning lowercase letters to the most liberal variants (i.e., liberal selection and weak fairness). Intuitively, due

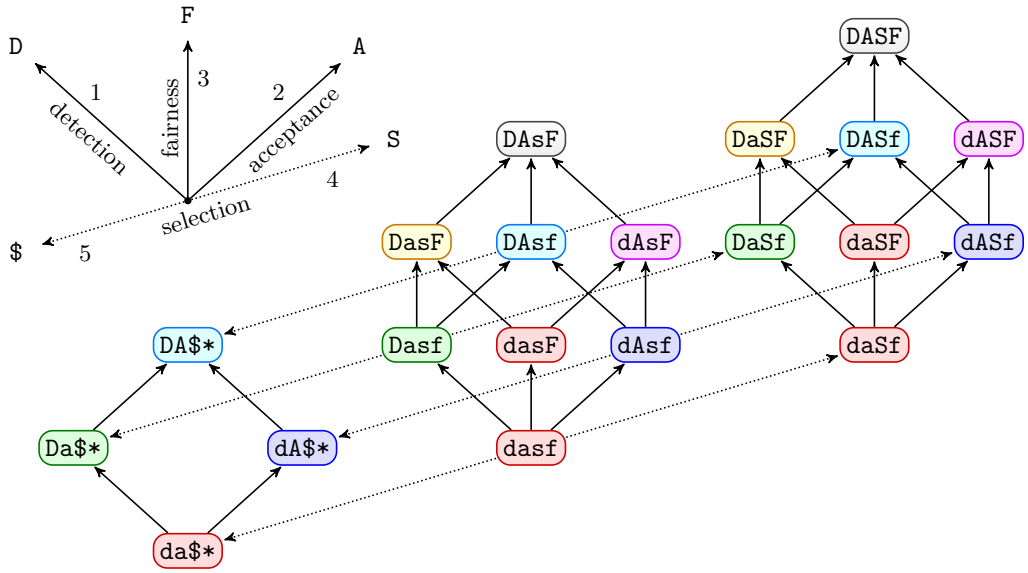


Figure 1 Initial classification of the models according to the class of graph languages they recognize. Arrows indicate inclusion between classes of languages. The diagram can be thought of as lying in four-dimensional space, where each dimension represents one of our four parameters. The vectors of the "coordinate system" are labeled with the statement number of Lemma 3 that proves the inclusions in the corresponding direction. In the coming sections, classes are shown to be equal if and only if they have the same color, reducing the 20 classes to 7, as shown in Figure 4. This means in particular that we completely eliminate the dimension of selection (shown in dotted lines), leaving us with only three dimensions.

to the consistency condition, the more liberal a scheduler, the harder it is for an automaton to recognize a graph language, because more runs have to yield the same result. So, loosely speaking, we expect the expressive power to increase with the number of uppercase letters.

Detection	*Acceptance*	*Selection*	*Fairness*
`d`: non-counting	`a`: halting	`s`: liberal	`f`: weak
`D`: counting	`A`: stable consensus	`S`: exclusive	`F`: strong
		`$`: synchronous	

We denote each class of automata by a string $wxyz \in \{\mathtt{d},\mathtt{D}\} \times \{\mathtt{a},\mathtt{A}\} \times \{\mathtt{s},\mathtt{S},\mathtt{\$}\} \times \{\mathtt{f},\mathtt{F}\}$. The class of languages recognized by $wxyz$-automata is denoted $\mathcal{G}(wxyz)$. The following lemma states all relations between language classes that follow directly from the definitions. Statement 1 abbreviates "$\mathcal{G}(\mathtt{d}xyz) \subseteq \mathcal{G}(\mathtt{D}xyz)$ for all $x \in \{\mathtt{a},\mathtt{A}\}$, $y \in \{\mathtt{s},\mathtt{S},\mathtt{\$}\}$, $z \in \{\mathtt{f},\mathtt{F}\}$". We use the same convention in Statements 2 to 5, and throughout the paper. That is, any statement with four-letter strings containing the wildcard symbol `*` must be expanded into the list of all statements that can be obtained by replacing identically positioned occurrences of `*` with the same letter.

▶ **Lemma 3.** *1.* $\mathcal{G}(\mathtt{d***}) \subseteq \mathcal{G}(\mathtt{D***})$, *2.* $\mathcal{G}(\mathtt{*a**}) \subseteq \mathcal{G}(\mathtt{*A**})$, *3.* $\mathcal{G}(\mathtt{***f}) \subseteq \mathcal{G}(\mathtt{***F})$, *4.* $\mathcal{G}(\mathtt{**sf}) \subseteq \mathcal{G}(\mathtt{**Sf})$, *5.* $\mathcal{G}(\mathtt{**sf}) \subseteq \mathcal{G}(\mathtt{**\$f})$, *6.* $\mathcal{G}(\mathtt{**\$F}) \subseteq \mathcal{G}(\mathtt{**\$f})$.

Lemma 3 leads to the diagram in Figure 1, showing 20 automata classes (we have $\mathcal{G}(\mathtt{**\$f}) = \mathcal{G}(\mathtt{**\$F})$ by Statements 3 and 6). An arrow between two classes means that every graph language recognized by the source class is also recognized by the target class.

The reader probably finds Figure 1 very complicated. We also do, and this was the motivation for the present paper. How many of these classes are really different? In the next sections we show that classes with the same color have the same expressivity, and thus that the diagram of Figure 1 collapses to the one of Figure 4, which contains only seven classes.

3 The weakest classes have no expressiveness

We prove that `das*`-automata have no expressive power, and the results in Sections 4 and 5 will generalize this to `da**`-automata. Intuitively, if agents cannot count their neighbors, and must reach a halting configuration, then they cannot distinguish any two graphs. Formally, a graph property is *trivial* if either every graph satisfies it, or no graph satisfies it. We have:

▶ **Theorem 4.** *Every* `das*`*-automaton recognizes a trivial graph property.*

Proof sketch. By Statement 3 of Lemma 3, it suffices to prove the claim for `dasF`-automata. So let A be a `dasF`-automaton, and let G and H be two graphs (connected and with at least two nodes by convention). Assume that A accepts G but rejects H. By the consistency condition, all fair runs of A on G accept, and all fair runs on H reject. Now let ρ^G and ρ^H be any such runs, and let $t \in \mathbb{N}$ be a time at which all nodes in ρ^G and ρ^H have halted. We define a new graph K that consists of t copies $\{G_i\}_{i\in[1:t]}$ and $\{H_i\}_{i\in[1:t]}$ of G and H, with additional edges defined as follows. For each node w^X of the original graph $X \in \{G, H\}$, we denote its copy in X_i by w_i^X, where $i \in [1:t]$. Let u^G and v^G be two *adjacent* nodes of G, and u^H and v^H be two *adjacent* nodes of H. We add the connecting edges $\{u_i^X, v_{i+1}^X\}$ for all $i \in [1:t\rangle$ and $X \in \{G, H\}$, as well as the edge $\{u_t^G, u_t^H\}$. This is illustrated in Figure 2.

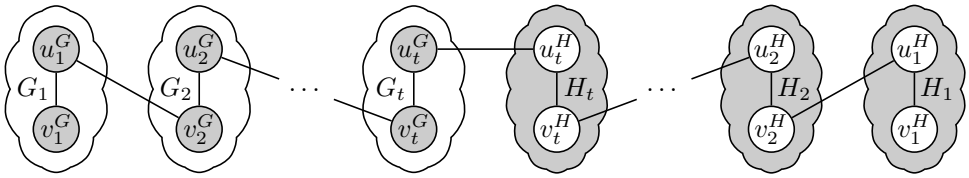


Figure 2 Graph K used in the proof of Theorem 4.

We show that there is a fair run ρ of A on K that neither accepts nor rejects. It follows that A does not satisfy the consistency condition, contradicting the hypothesis. Since A is a non-counting automaton, initially every node w_i^X except for u_t^G and u_t^H "sees" the same neighborhood as the corresponding node w^X in the original graph X. Only the two nodes u_t^G and u_t^H may have a different neighborhoods than u^G and u^H, and this might affect their behavior starting at time 1. Their different behavior can be propagated to other nodes in subsequent rounds, but it takes time before it reaches every node. We exploit this to construct ρ in such a way that some nodes of K (those of G_1) reach an accepting state, while others (those of H_1) reach a rejecting state. Since A is a halting automaton, these nodes will never change their state again, and so the run is neither accepting nor rejecting. ◀

4 Synchronicity can always be simulated

We show that every class with synchronous selection is equivalent to the corresponding class with liberal selection. Albeit non-trivial, this is easy to prove by a standard technique of distributed computing known as *alpha synchronizer*. (The term was introduced in [4], but a similar idea appeared earlier in cellular automata theory [10].) Given a machine

$M = (Q, \delta_0, \delta, Y, N)$, we define a machine $\tilde{M} = (\tilde{Q}, \tilde{\delta_0}, \tilde{\delta}, \tilde{Y}, \tilde{N})$ such that for every graph G, the unique synchronous run of M on G accepts (rejects) iff every weakly fair run ρ of $\tilde{M}$ on G accepts (rejects). The gadget achieving this is called a "synchronizer", because it ensures that the nodes of G behave "as in the synchronous case", even when selection is liberal.

The set of states of $\tilde{M}$ is $\tilde{Q} := Q \times Q \times \{0, 1, 2\}$. Given $(q, q', i) \in \tilde{Q}$, we call q the *past M-state*, q' the *current M-state*, and i the *phase*. The initialization function is given by $\tilde{\delta}_0(a) := (\delta_0(a), \delta_0(a), 0)$. In order to define the transition function $\tilde{\delta}$, let v be a node in state (q, q', i). If v is selected by the scheduler, its next state is determined as follows:

- If at least one neighbor of v is in phase $(i - 1) \bmod 3$, then v does not change state. Intuitively, if some neighbor is still one phase behind, then v waits for it to "catch up".
- If every neighbor of v is in phase i or $(i+1) \bmod 3$, then v moves to $(q', q'', (i+1) \bmod 3)$, where q'' is defined as follows. Let N_v be the set of neighbors of v, and for each $u \in N_v$, let (q_u, q'_u, i_u) be the state of u. Further, let $q''_u := q'_u$ if $i_u = i$, and $q''_u := q_u$ if $i_u = (i + 1) \bmod 3$, and let $\mathcal{M}$ be the multiset over Q containing for each $u \in N_v$ a copy of the state q''_u. (Loosely speaking, $\mathcal{M}$ contains the current M-states of the neighbors of v that are in the same phase as v, and the past M-states of the neighbors that are one phase ahead, i.e., the states they had when they were in the same phase as v). Let $\mathcal{M}_\beta$ be given by $\mathcal{M}_\beta(q) = \min\{\beta, \mathcal{M}_\beta(q)\}$. We define $q'' := \delta(q, \mathcal{M}_\beta)$; loosely speaking, v moves to the state it would move to in M if all its neighbors were in the same phase.

Let $\tilde{\rho}$ be any weakly-fair run of $\tilde{M}$ on a graph G. Fix a node v of G, and extract from $\tilde{\rho}$ the sequence $q'_{10}q'_{11}q'_{12}\,q'_{20}q'_{21}q'_{22}\ldots q'_{i0}q'_{i1}q'_{i2}\ldots$, where q'_{ij} denotes the current M-state of v immediately after entering phase j for the i-th time. Now, let ρ be the unique synchronous run of M on G, and let $q'_0q'_1q'_2\ldots$ be the sequence obtained by projecting ρ onto the states of v. It is easy to see that these two sequences coincide. By the definition of stable acceptance, $\tilde{\rho}$ accepts iff ρ accepts, and rejects iff ρ rejects. Using this construction, we obtain:

▶ **Theorem 5.** *For every* `**$*`*-automaton there is an equivalent* `**s*`*-automaton.*

5 Exclusivity does not increase expressiveness

In this section, we obtain the rather surprising result that the computational power of a class of automata does not increase if we restrict its schedulers to interleaving ones (which guarantees that agents act in mutual exclusion with all other agents).

5.1 Exclusivity under strong fairness

We start by considering strongly fair models, i.e., we compare a class of the form `**sF` with the corresponding class `**SF`. On an intuitive level, their equivalence might be less surprising than the subsequent result presented in Section 5.2 because strong fairness provides a way to break symmetry, which can be exploited to simulate exclusivity. Nevertheless, neither class trivially subsumes the other, so we have to prove inclusions in both directions.

▶ **Theorem 6.** *For every* `**sF`*-automaton there is an equivalent* `**SF`*-automaton.*

Proof sketch. Given a `**sF`-automaton A, we construct a `**SF`-automaton B such that for all input graphs G, every strongly fair run of B on G simulates a strongly fair run of A on G. The difficulty lies in the fact that A and B do not share the same notion of strong fairness because they have different selection constraints. While A's liberal scheduler guarantees that arbitrary sequences of selections will occur infinitely often, B's exclusive scheduler can select only one node at a time. To simulate A's behavior with B, we adapt the synchronizer

from Section 4. Just like there, nodes keep track of their previous and current state in A, as well as the current phase number modulo 3. However, instead of updating their state in every phase, they only do so if an additional *activity flag* is set. Thus, we can simulate an arbitrary selection S by raising the flags of exactly those nodes that lie in S. The outcome of a phase simulated in this way will be the same as if all the nodes in S made a transition simultaneously. The main issue is how to set the activity flags in each phase in such a way that every finite sequence $(S_1, \ldots, S_n)$ of selections is guaranteed to occur infinitely often. We show that this is possible, exploiting the fact that B's scheduler is strongly fair. ◀

▶ **Theorem 7.** *For every* `**SF`*-automaton there is an equivalent* `**sF`*-automaton.*

Proof sketch. First, we note that the only way exclusivity could possibly be useful is to break symmetry between adjacent nodes. This is because for an independent set (i.e., a set of pairwise non-adjacent nodes), the order of activation is irrelevant: whether the scheduler activates them all at once or one by one in some arbitrary order, the outcome will always be the same. Consequently, to simulate a run with exclusivity, it suffices to simulate a run where no two adjacent nodes are active at the same time. We provide a simple protocol that makes use of the strong fairness constraint (in an environment with liberal selection) to ensure that if a node wants to execute a transition, then it will eventually be able to do so while all its neighbors remain passive. ◀

5.2 Exclusivity under weak fairness

We now show that even in the absence of strong fairness, the restriction to interleaving schedulers does not increase expressive power. At first sight, this may be quite surprising because exclusivity inherently breaks symmetry, whereas an automaton with liberal selection and weak fairness can always be assumed to run synchronously and thus be incapable of breaking symmetry. In fact, it is easy to come up with examples of automata that exploit exclusivity to ensure termination.

▶ **Proposition 8.** *For every* `**sf`*-automaton, there exists a* `**Sf`*-automaton that recognizes the same graph language but makes use of exclusive selection to ensure termination. If run synchronously, it never terminates (and hence it is not a valid* `**sf`*-automaton).*

However, although the automata described in Proposition 8 make use of exclusivity, they do not really benefit from it; they only recognize languages that can also be recognized by liberal automata. As we will see in Theorem 11, this observation can be generalized to arbitrary `**Sf`-automata. Intuitively, since exclusivity does not add any expressive power, it can in a certain sense be simulated without needing to break symmetry.

The proof of Theorem 11 is based on the notion of Kronecker cover. The *Kronecker cover* (also known as *bipartite double cover*) of a graph $G = (V, E, \lambda)$ is the bipartite graph $G' = (V', E', \lambda')$ where $V' = V \times \{0, 1\}$, $E' = \bigcup_{\{u,v\} \in E} \{\, \{(u, 0), (v, 1)\}, \{(u, 1), (v, 0)\} \,\}$, and $\lambda'((v, i)) = \lambda(v)$ for all $(v, i) \in V'$. An example is provided in Figure 3.

The Kronecker cover in Figure 3 is connected because the nodes in $\{u, v, w\} \times \{0, 1\}$ form a cycle. The following lemma generalizes this observation.

▶ **Lemma 9.** *The Kronecker cover of a connected graph G is connected if and only if G contains a cycle of odd length, (i.e., if and only if G is non-bipartite).*

If a Kronecker cover is connected, then it constitutes a legal input for a distributed automaton. The next key lemma shows that, in this case, a weakly fair automaton cannot even distinguish between a graph and its Kronecker cover.

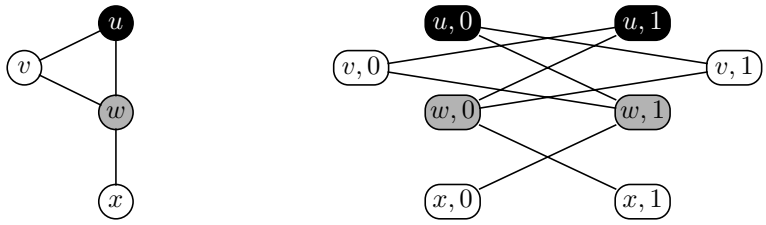


Figure 3 A graph (on the left) and its Kronecker cover (on the right).

▶ **Lemma 10.** *For every* `***f`*-automaton A with input alphabet Λ and every non-bipartite Λ-labeled graph G, A accepts G if and only if it accepts the Kronecker cover of G.*

We can now prove the main technical result of this section:

▶ **Theorem 11.** *For every* `**Sf`*-automaton there is an equivalent* `**sf`*-automaton.*

Proof sketch. Given a `**Sf`-automaton A, we construct an equivalent `**$f`-automaton B (i.e., a synchronous automaton). This is sufficient to prove the claim, because we know from Theorem 5 that B can always be simulated by a `**sf`-automaton using a synchronizer.

Let G be an input graph for A. If we were guaranteed that the labels of G define a proper vertex coloring (i.e., edges connect nodes of different colors), then the task would be straightforward. Indeed, since each color of a proper coloring represents an independent set, B could simply operate in cyclically repeating phases, each one activating precisely the nodes of one of the colors. As explained in the proof of Theorem 7, such a run is equivalent to a run of an exclusive scheduler that activates the nodes of each independent set one by one (in some arbitrary order).

This approach can be adapted to bipartite graphs because a bipartite graph has exactly two possible 2-colorings. However, computing one of the two 2-colorings would require to break symmetry, which a `**$f`-automaton cannot do. So instead, the states of automaton B have two components, one corresponding to each coloring, and nodes update both components when they are activated.

Using these ideas, we construct B in such a way that it recognizes the same bipartite graphs as A. Then we use Lemmas 9 and 10 to prove that $L(A) = L(B)$. Indeed, if G is not bipartite, then by Lemma 9, its Kronecker cover G' is connected and therefore constitutes a legal input for a distributed automaton. By Lemma 10, B accepts G if and only if it accepts G'. Since Kronecker covers are bipartite by definition, we know from the above discussion that B accepts G' if and only if A accepts G'. Finally, again by Lemma 10, A accepts G' if and only if it accepts G. From this chain of equivalences, we can conclude that G is accepted by B if and only if it is accepted by A. ◀

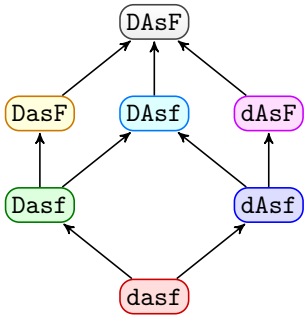
6 Separations

In Sections 3, 4 and 5 we have shown that the classes of graph languages in Figure 1 collapse to *at most* the seven classes shown on the left of Figure 4. In this section we show that the seven classes are all different. For this we examine four graph languages, and determine which classes are expressive enough to recognize them:

- $\mathcal{B}$: The language of graphs with set of labels $\{black, white\}$ having at least one black node.
- $\mathcal{S}$: The language of star graphs, i.e., the set of all connected, unlabeled graphs in which one node (the *center*) has degree at least 2, and all others (the *leaves*) have degree 1.

- $\mathcal{C}_3$: The language containing one single graph, namely the cycle C_3 with three nodes labeled by 0, 1, and 2, respectively.
- $\mathcal{S}_{even}$: The language of even stars, i.e., the graphs of $\mathcal{S}$ with an even number of leaves.

The results are summarized on the right of Figure 4.



Class	$\mathcal{B}$	$\mathcal{S}$	$\mathcal{C}_3$	$\mathcal{S}_{even}$
DAsF	✓	✓	✓	✓
DasF	✗	✓	✗	✓
DAsf	✓	✓	✗	✗
dAsF	✓	✓	✓	✗
Dasf	✗	✓	✗	✗
dAsf	✓	✗	✗	✗
dasf	✗	✗	✗	✗

Figure 4 On the left, quotient of the classification of Figure 1. On the right, four graph languages, and the automata models capable of recognizing them.

Recognizing properties of labeled graphs: the language $\mathcal{B}$

The main difference between the two types of acceptance is that halting automata cannot recognize properties that require nodes to wait an unlimited amount of time for some information that may never arrive, while even the simplest class of automata accepting by stable consensus can recognize some of those properties, such as $\mathcal{B}$.

▶ **Proposition 12.** *$\mathcal{B}$ is recognizable by a* `dAsf`*-automaton, but not by any* `*a**`*-automaton.*

Proof sketch. The `dAsf`-automaton has two states, called *black* and *white*. The initial state of a node is given by its label. Black nodes remain always black, and white nodes with a black neighbor become black. Since graphs are connected by assumption, if a graph contains some black node then eventually all nodes are black, otherwise all nodes stay white.

For the second part, one can show that `DasF`-automata cannot distinguish between an entirely white cycle and a sufficiently long path graph whose nodes are all white except for two black nodes at the endpoints. (The argument is similar to the proof of Theorem 4.) ◀

Recognizing properties of unlabeled graphs: the language $\mathcal{S}$

We show in Proposition 13 that `dAsf`-automata cannot recognize any non-trivial property of *unlabeled* graphs (which we identify with the labeled graphs whose nodes all carry the same label). That is, while `dAsf`-automata can recognize properties of the labeling of a graph, they cannot recognize any non-trivial property of its *structure*. Then we show in Proposition 14 that the strong fairness of `dAsF`-automata allows them to recognize $\mathcal{S}$.

▶ **Proposition 13.** `dAsf`*-automata can only recognize trivial properties of* unlabeled *graphs. In particular, $\mathcal{S}$ is not recognizable by a* `dAsf`*-automaton.*

Proof. Let A be a `dAsf`-automaton, and let $\rho = (C_0, C_1, \ldots)$ be the synchronous run of A on an unlabeled graph $G = (V, E)$, i.e., the run scheduled by V^ω. We show that A either accepts all unlabeled graphs, or rejects all unlabeled graphs. Since V^ω is a weakly fair schedule, ρ is a fair run, and so by the consistency condition A accepts G iff ρ is accepting. Since G is unlabeled, in C_0 every node of G is in the same state q_0, which is independent of G. Moreover,

since ρ is synchronous and A is non-counting, in each configuration C_i every node of G is in the same state q_i, which is also independent of G. So the states visited by ρ are independent of G, and so A either accepts all unlabeled graphs, or rejects all unlabeled graphs. ◀

▶ **Proposition 14.** *$\mathcal{S}$ is recognizable by a* `dAsF`*-automaton and by a* `Dasf`*-automaton.*

Proof sketch. We give a `dAsF`-automaton that recognizes $\mathcal{S}$. The states of the automaton are pairs (d, c), where $d \in \{$*leaf*, *center*, *unknown*, *neither*$\}$ is the *estimate* of v, and $c \in \{0, 1\}$ is its *color*. Every time a node is selected it flips its color. When a node with estimate *unknown* sees two neighbors with different colors, it switches to *center*, and if from then on it sees a neighbor with estimate *center*, it moves to *neither*. Strong fairness is crucial for correctness: by Lemma 2, it ensures that a node that is not a leaf will eventually be selected in a configuration in which at least two of its neighbors have different colors.

Now we give a `Dasf`-automaton with $\beta = 2$ that recognizes $\mathcal{S}$. Since $\beta = 2$, a node can determine for each state q if it has 0, 1, or at least 2 neighbors in q. The automaton's states are $\{$*init*, *leaf*, *non-leaf*, *accept*, *reject*$\}$. Initially all nodes are in state *init*. The nodes update their estimates depending on the number of neighbors (0, 1, or at least 2) in each state. ◀

Symmetry breaking: the language $\mathcal{C}_3$

We show that the language $\mathcal{C}_3$ requires both acceptance by stable consensus and strong fairness to be recognizable. Intuitively, both of them are required to distinguish C_3 from arbitrarily long cycles that repeat the labeling of C_3 cyclically.

▶ **Proposition 15.** *$\mathcal{C}_3$ is recognizable by a* `dAsF`*-automaton, but neither by* `DA*f`*-automata nor by* `Da*F`*-automata.*

Proof sketch. Our `dAsF`-automaton for $\mathcal{C}_3$ checks two conditions: first, that the input graph is a cycle with cyclic labeling $0{-}1{-}2$, and second, that it contains exactly one node labeled by 2 (which implies that the cycle has length 3). For both conditions, we use a similar trick as in Proposition 14, relying on acceptance by stable consensus and strong fairness to eventually break symmetry between otherwise indistinguishable nodes. To verify the second condition, each node labeled by 2 successively sends signals in both directions through the cycle, and checks that those signals always come back from the expected direction.

For the second part of the claim, we show that `DA*f-` and `Da*F`-automata cannot distinguish C_3 from C_6, the hexagon whose nodes are labeled by $0{-}1{-}2{-}0{-}1{-}2$ (and back to 0). To do so, given a fair run ρ_3 of such an automaton on C_3, we construct a fair run ρ_6 on C_6 that "duplicates" the behavior of ρ_3. In the case of `Da*F`-automata, this duplication is performed only until ρ_3 has reached a halting configuration (because otherwise ρ_6 would violate the strong fairness constraint). ◀

Counting neighbors modulo a number: the language $\mathcal{S}_{even}$

Since counting automata can only count up to a threshold β, no node can directly observe that it has an even number of neighbors. This makes the language $\mathcal{S}_{even}$ rather difficult to recognize. We now show that the combination of counting and strong fairness can do the job. The proof also provides a good example where exclusivity helps to design an algorithm.

▶ **Proposition 16.** *$\mathcal{S}_{even}$ is recognizable by a* `DasF`*-automaton.*

Proof sketch. In Proposition 14 we have exhibited a `Dasf`-automaton A recognizing $\mathcal{S}$. We now give a `DaSF`-automaton B that uses counting, exclusivity, and strong fairness to further decide if the number of leaves is even. Loosely speaking, B first executes A; if A rejects, then B rejects, because the graph is not even a star. If A accepts, then B enters a new phase during which it counts the number of leaves modulo 2. By Theorem 7, B is equivalent to a `DasF`-automaton.

We can assume that when A accepts, all nodes are labeled with either *leaf* or *center* (the unique non-leaf). We give an informal description of B. Leaves can be in states *visible*, *invisible*, *dead*, *even*, or *odd*. While leaves have not been counted by the center, they alternate between the states *visible* and *invisible*. The center only increments its modulo-2 counter if exactly one leaf is *visible*. After a leaf is counted, it moves to *dead*. When all leaves become dead, i.e., when they have all been counted, the center decides whether to accept or reject; the leaves read the decision from the counter, and move to *even* or *odd* accordingly. ◀

The next two results show that recognizing $\mathcal{S}_{even}$ needs both counting and strong fairness.

▶ **Proposition 17.** *$\mathcal{S}_{even}$ is not recognizable by `DA*f`-automata.*

Proof. We show that for every `DA*f`-automaton A there exist stars G and G' such that exactly one of G and G' belongs to S_{even}, but A either accepts both of them or rejects both of them. Let $\beta \geq 1$ be A's counting bound, and let G and G' be the stars with $\beta + 1$ and $\beta + 2$ leaves, respectively. Now consider the synchronous runs ρ and ρ' of A on G and G'. By symmetry, and since the number of leaves exceeds β in both G and G', at every time $t \in \mathbb{N}$, the center is in the same state in ρ and ρ', and likewise all leaves are in the same state. So the sequences of states visited by the center and the leaves are the same in both ρ and ρ', and therefore ρ is accepting iff ρ' is accepting. ◀

▶ **Proposition 18.** *$\mathcal{S}_{even}$ is not recognizable by `dA*F`-automata.*

Proof sketch. Given a `dA*F`-automaton A, the proof identifies an even number n, depending on A, such that if A accepts the star with n leaves, then it cannot reject the star with $n + 1$ leaves. The proof is involved, and can be found in the appendix of the arXiv version. ◀

7 Expressive power

As a first application of our results, we investigate the expressivity of our models for graph languages that depend only on the labeling function of a graph, and not on its topology.

Given a Λ-labeled graph $G = (V, E, \lambda)$, where $\Lambda = \{\ell_1, \ldots, \ell_k\}$, let $\#_G \colon \Lambda \to \mathbb{N}$ be the mapping that assigns to each label ℓ the number $\#_G(\ell)$ of nodes of V such that $\lambda(v) = \ell$. A language is *Presburger-definable* if there is a formula $\varphi(x_1, \ldots, x_k)$ of Presburger arithmetic such that a Λ-labeled graph G belongs to the language if and only if $\varphi(\#_G(\ell_1), \ldots, \#_G(\ell_k))$ holds. An example of such a language is $\mathcal{B}$, the set of graphs that contain a black node.

We show that `DAsF`-automata recognize all Presburger languages, but none of the other six classes do. The negative part of the result follows easily from the table in Figure 4.

▶ **Proposition 19.** *There exist Presburger-definable languages that are not recognizable by `d***-`, `*a**-`, or `***f`-automata.*

Proof. By Proposition 12, `*a**`-automata cannot recognize the language $\mathcal{B}$, which is Presburger-definable. Furthermore, by Propositions 14, 17 and 18, `dA*F-` and `DA*f`-automata can recognize the language $\mathcal{S}$ of star graphs but not the language $\mathcal{S}_{even}$ of stars with an

even number of leaves. This implies that dA*F- and DA*f-automata cannot recognize the Presburger-definable language of graphs with an odd number of nodes, because the intersection of this language with $\mathcal{S}$ is equal to $\mathcal{S}_{even}$, and languages recognizable by distributed automata are closed under intersection (by a standard product construction). ◀

For the positive part, we proceed in three steps: First, following [1] and Section 5 of [3], we introduce *graph population protocols*, a graph variant of the well-known population protocol model introduced in [2, 3]. Then we recall a result of [3] showing that graph population protocols recognize all Presburger-definable languages. Finally, we show that every graph population protocol can be simulated by a DAsF-automaton.

Our definition of graph population protocols is equivalent to that of [1, 3], but reuses the notation of Section 2 as far as possible. A *graph population protocol* $\Pi = (Q, \delta_0, \delta, Y, N)$ is defined like a DASF-automaton with machine $M = \Pi$, except for the following differences:

- The transition function is of the form $\delta\colon Q^2 \to Q^2$.
- A selection of a graph $G = (V, E, \lambda)$ is an ordered pair $S = (u, v) \in V^2$ of *adjacent* nodes (instead of a singleton $\{u\} \subseteq V$), and the selection constraint on G is $\{(u, v) \mid \{u, v\} \in E\}$.
- $C_t(v)$ is defined inductively as follows, for $t \in \mathbb{N}$ and $v \in V$:

$$C_0(v) = \delta_0(\lambda(v)) \quad \text{and} \quad C_{t+1}(v) = \begin{cases} \delta\big(C_t(v), C_t(u)\big)_{\text{fst}} & \text{if } S_t = (v, u) \text{ for some } u, \\ \delta\big(C_t(u), C_t(v)\big)_{\text{snd}} & \text{if } S_t = (u, v) \text{ for some } u, \\ C_t(v) & \text{otherwise,} \end{cases}$$

where P_{fst} and P_{snd} denote the first and second component of a pair P.

So, intuitively, the scheduler selects two adjacent nodes, which update their states according to δ. The definitions of all other relevant notions remain the same. This holds in particular for acceptance by stable consensus and strong fairness (which are baked into the model), and the consistency condition. Standard population protocols correspond to graph population protocols on complete graphs, where every pair of distinct nodes is connected by an edge.

It is shown in [3] that standard population protocols recognize all Presburger-definable languages. Further, Theorem 7 of [3] shows that every language recognized by population protocols is also recognized by graph population protocols. Loosely speaking, given a population protocol, one constructs the protocol on graphs in which, when an edge of the graph is selected, either the two nodes connected by it interact as in the population protocol, or they swap their states. By strong fairness, the states of the nodes can "move around the graph", and any pair of states eventually interacts infinitely often. The choice between interacting or swapping is nondeterministic, but it can be simulated by deterministic transitions (see [3]). Therefore, in order to show that DA*F-automata recognize all Presburger-definable languages, it suffices to simulate graph population protocols with distributed automata. As in the proof of Proposition 16, we make use of exclusivity to simplify the construction.

▶ **Proposition 20.** *For every graph population protocol there is an equivalent* DA*F*-automaton.*

Proof sketch. We present a simulation that runs a population protocol on a distributed automaton. To this end, the automaton has to simulate a scheduler that selects ordered pairs of adjacent nodes instead of arbitrary sets of nodes. For any pair (u, v) that is selected to perform a transition, let us call u the *initiator* and v the *responder* of the transition. By Theorem 7, we may assume that the automaton's scheduler selects a single node in each step.

The main idea is as follows: When a node u is selected and sees that it can become the initiator of a transition, it declares its intention to do so by raising the flag "?". Then u waits until some neighbor v is selected and raises the flag "!", which signals that v wants to become

the responder of a transition. If this happens, the next time u is selected, it computes its new state according to the state of v and the transition function of the population protocol, but also keeps its old state in memory so that v can still see it. After that, v also updates its state, and finally u deletes its old state, which completes the transition. Throughout this protocol, the nodes verify that they have exactly one partner during each transition. If this condition is violated, they raise the error flag "$\perp$" and abort their current transition. ◀

▶ **Corollary 21.** `DA*F`*-automata recognize all Presburger-definable languages.*

8 Conclusions

We have conducted an extensive comparative analysis of the expressive power of weak asynchronous models of distributed computing. Our analysis has reduced the initial "jungle" of twenty different models to only seven. This reduction in complexity is achieved by Theorems 4, 5, 6, 7, and 11, all of which have a clear and intuitive interpretation.

We have also shown that the seven classes are distinct, and have identified inclusions and non-inclusions between them. However, two inclusions remain open: Are `Dasf` or `DAsf` included in `dAsF`? Intuitively, this asks if strong fairness and acceptance by stable consensus can be used to simulate counting. We can provide a positive answer for graphs of bounded degree (a limitation common in practice), because in this case even `dA*F` and `DAsF` coincide.

▶ **Proposition 22.** *For every* `DA*F`*-automaton A and every $k \in \mathbb{N}$ there is a* `dA*F`*-automaton B equivalent to A on graphs of maximum degree k.*

However, for arbitrary graphs we conjecture that neither `Dasf` nor `DAsf` are included in `dAsF`.

Finally, we have made a first step towards characterizing the graph languages recognizable by the different classes, by transferring a characterization for population protocols.

As a last note, observe that our results hold for *decision* problems on *undirected* graphs that can be solved by consensus in the framework of distributed automata. Several of our constructions (e.g., those in Theorems 5 and 7) rely on bidirectional communication, which is not guaranteed on directed graphs. Furthermore, exclusive selection leads to higher computational power for non-decision problems. For instance, it can be used to solve the vertex coloring problem on graphs of bounded degree (by a standard greedy algorithm), which, for symmetry reasons, is impossible in a model with synchronous selection.

References

1. Dana Angluin, James Aspnes, Melody Chan, Michael J Fischer, Hong Jiang, and René Peralta. Stably computable properties of network graphs. In *International Conference on Distributed Computing in Sensor Systems*, pages 63–74. Springer, 2005.
2. Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *PODC*, pages 290–299. ACM, 2004.
3. Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.
4. Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, 1985. `doi:10.1145/4221.4227`.
5. Alejandro Cornejo and Fabian Kuhn. Deploying wireless networks with beeps. In *DISC*, volume 6343 of *Lecture Notes in Computer Science*, pages 148–162. Springer, 2010.
6. Yuval Emek and Roger Wattenhofer. Stone age distributed computing. In *PODC*, pages 137–146. ACM, 2013.

7 Nissim Francez. *Fairness*. Texts and Monographs in Computer Science. Springer, 1986.

8 Lauri Hella, Matti Järvisalo, Antti Kuusisto, Juhana Laurinharju, Tuomo Lempiäinen, Kerkko Luosto, Jukka Suomela, and Jonni Virtema. Weak models of distributed computing, with connections to modal logic. *Distributed Computing*, 28(1):31–53, 2015.

9 Daniel Lehmann, Amir Pnueli, and Jonathan Stavi. Impartiality, justice and fairness: The ethics of concurrent termination. In *ICALP*, volume 115 of *Lecture Notes in Computer Science*, pages 264–277. Springer, 1981.

10 Katsuhiko Nakamura. Synchronous to asynchronous transformation of polyautomata. *J. Comput. Syst. Sci.*, 23(1):22–37, 1981. doi:10.1016/0022-0000(81)90003-9.

11 Saket Navlakha and Ziv Bar-Joseph. Distributed information processing in biological and computational systems. *Commun. ACM*, 58(1):94–102, 2015. doi:10.1145/2678280.

12 Fabian Reiter. Asynchronous distributed automata: A characterization of the modal mu-fragment. In *ICALP*, volume 80 of *LIPIcs*, pages 100:1–100:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

13 David Soloveichik, Matthew Cook, Erik Winfree, and Jehoshua Bruck. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7(4):615–633, 2008.

Scalable Termination Detection for Distributed Actor Systems

Dan Plyukhin
University of Illinois at Urbana-Champaign, Urbana, IL, USA
daniilp2@illinois.edu

Gul Agha
University of Illinois at Urbana-Champaign, Urbana, IL, USA
agha@illinois.edu

Abstract

Automatic *garbage collection* (GC) prevents certain kinds of bugs and reduces programming overhead. GC techniques for sequential programs are based on *reachability analysis*. However, testing reachability from a root set is inadequate for determining whether an *actor* is garbage because an unreachable actor may send a message to a reachable actor. Instead, it is sufficient to check *termination* (sometimes also called *quiescence*): an actor is terminated if it is not currently processing a message and cannot receive a message in the future. Moreover, many actor frameworks provide all actors with access to file I/O or external storage; without inspecting an actor's internal code, it is necessary to check that the actor has terminated to ensure that it may be garbage collected in these frameworks. Previous algorithms to detect actor garbage require coordination mechanisms such as causal message delivery or nonlocal monitoring of actors for mutation. Such coordination mechanisms adversely affect concurrency and are therefore expensive in distributed systems. We present a low-overhead *reference listing* technique (called *DRL*) for termination detection in actor systems. DRL is based on asynchronous local snapshots and message-passing between actors. This enables a decentralized implementation and transient network partition tolerance. The paper provides a formal description of DRL, shows that all actors identified as garbage have indeed terminated (safety), and that all terminated actors–under certain reasonable assumptions–will eventually be identified (liveness).

2012 ACM Subject Classification Computing methodologies → Concurrent algorithms; Software and its engineering → Garbage collection

Keywords and phrases actors, concurrency, termination detection, quiescence detection, garbage collection, distributed systems

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.11

Funding This work was supported in part by the National Science Foundation under Grant No. SHF 1617401, and in part by the Laboratory Directed Research and Development program at Sandia National Laboratories, a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Acknowledgements We would like to thank Dipayan Mukherjee, Atul Sandur, Charles Kuch, Jerry Wu, and the anonymous referees for providing valuable feedback in earlier versions of this work.

1 Introduction

The actor model [1, 2] is a foundational model of concurrency that has been widely adopted for its scalability: for example, actor languages have been used to implement services at PayPal [19], Discord [27], and in the United Kingdom's National Health Service database [18]. In the actor model, stateful processes known as *actors* execute concurrently and communicate by sending asynchronous messages to other actors, provided they have a *reference* (also called

a *mail address* or *address* in the literature) to the recipient. Actors can also spawn new actors. An actor is said to be *garbage* if it can be destroyed without affecting the system's observable behavior.

Although a number of algorithms for automatic actor GC have been proposed [10, 13, 24, 25, 28, 30], actor languages and frameworks currently popular in industry (such as Akka [4], Erlang [5], and Orleans [8]) require that programmers garbage collect actors manually. We believe this is because the algorithms proposed thus far are too expensive to implement in distributed systems. In order to find applicability in real-world actor runtimes, we argue that a GC algorithm should satisfy the following properties:

1. (*Low latency*) GC should not restrict concurrency in the application.
2. (*High throughput*) GC should not impose significant space or message overhead.
3. (*Scalability*) GC should scale with the number of actors and nodes in the system.

To the best of our knowledge, no previous algorithm satisfies all three constraints. The first requirement precludes any global synchronization between actors, a "stop-the-world" step, or a requirement for causal order delivery of all messages. The second requirement means that the number of additional "control" messages imposed by the algorithm should be minimal. The third requirement precludes algorithms based on global snapshots, since taking a global snapshot of a system with a large number of nodes is infeasible.

To address these goals, we have developed a garbage collection technique called *DRL* for *Deferred Reference Listing*. The primary advantage of DRL is that it is decentralized and incremental: local garbage can be collected at one node without communicating with other nodes. Garbage collection can be performed concurrently with the application and imposes no message ordering constraints. We also expect DRL to be reasonably efficient in practice, since it does not require many additional messages or significant actor-local computation.

DRL works as follows. The *communication protocol* (Section 4) tracks information, such as references and message counts, and stores it in each actor's state. Actors periodically send out copies of their local state (called *snapshots*) to be stored at one or more designated *snapshot aggregator* actors. Each aggregator periodically searches its local store to find a subset of snapshots representing terminated actors (Section 6). Once an actor is determined to have terminated, it can be garbage collected by, for example, sending it a *self-destruct* message. Note that our termination detection algorithm itself is *location transparent*.

Since DRL is defined on top of the actor model, it is oblivious to details of a particular implementation (such as how sequential computations are represented). Our technique is therefore applicable to any actor framework and can be implemented as a library. Moreover, it can also be applied to open systems, allowing a garbage-collected actor subsystem to interoperate with an external actor system.

The outline of the paper is as follows. We provide a characterization of actor garbage in Section 2 and discuss related work in Section 3. We then provide a specification of the DRL protocol in Section 4. In Section 5, we describe a key property of DRL called the *Chain Lemma*. This lemma allows us to prove the safety and liveness properties, which are stated in Section 6. We then conclude in Section 7 with some discussion of future work and how DRL may be used in practice. To conserve space, all proofs have been relegated to the Appendix.

2 Preliminaries

An actor can only receive a message when it is *idle*. Upon receiving a message, it becomes *busy*. A busy actor can perform an unbounded sequence of *actions* before becoming idle. In [3], an action may be to spawn an actor, send a message, or perform a (local) computation. We will also assume that actors can perform effects, such as file I/O. The actions an actor performs in response to a message are dictated by its application-level code, called a *behavior*.

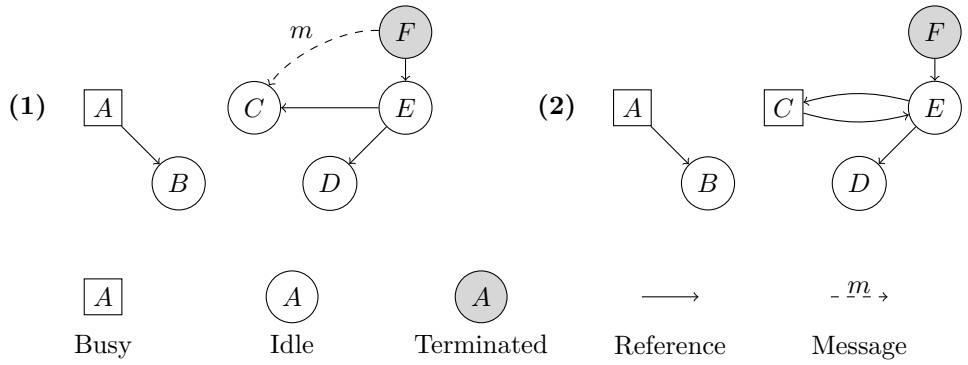


Figure 1 A simple actor system. The first configuration leads to the second after C receives the message m, which contains a reference to E. Notice that an actor can send a message and "forget" its reference to the recipient before the message is delivered, as is the case for actor F. In both configurations, E is a potential acquaintance of C, and D is potentially reachable from C. The only terminated actor is F because all other actors are potentially reachable from unblocked actors.

Actors can also receive messages from *external* actors (such as the user) by becoming *receptionists*. An actor A becomes a receptionist when its address is exposed to an external actor. Subsequently, any external actor can potentially obtain A's address and send it a message. It is not possible for an actor system to determine when all external actors have "forgotten" a receptionist's address. We will therefore assume that an actor can never cease to be a receptionist once its address has been exposed.

An actor is said to be garbage if it can be destroyed without affecting the system's observable behavior. However, without analyzing an actor's code, it is not possible to know whether it will have an effect when it receives a message. We will therefore restrict our attention to actors that can be guaranteed to be garbage without inspecting their behavior. According to this more conservative definition, any actor that might receive a message in the future should not be garbage collected because it could, for instance, write to a log file when it becomes busy. Conversely, any actor that is guaranteed to remain idle indefinitely can safely be garbage collected because it will never have any effects; such an actor is said to be *terminated*. Hence, garbage actors coincide with terminated actors in our model.

Terminated actors can be detected by looking at the global state of the system. We say that an actor B is a *potential acquaintance* of A (and A is a *potential inverse acquaintance* of B) if A has a reference to B or if there is an undelivered message to A that contains a reference to B. We define *potential reachability* to be the reflexive transitive closure of the potential acquaintance relation. If an actor is idle and has no undelivered messages, then it is *blocked*; otherwise it is *unblocked*. We then observe that an actor is terminated when it is only potentially reachable by blocked actors: Such an actor is idle, blocked, and can only potentially be sent a message by other idle blocked actors. Conversely, without analyzing actor code we cannot safely conclude that an actor is terminated if it is potentially reachable by an unblocked actor. Hence, we say that an actor is terminated if and only if it is blocked and all of its potential inverse acquaintances are terminated.

3 Related Work

Global Termination

Global termination detection (GTD) is used to determine when *all* processes have terminated [17, 16]. For GTD, it suffices to obtain global message send and receive counts. Most GTD algorithms also assume a fixed process topology. However, Lai gives an algorithm in [14] that supports dynamic topologies such as in the actor model. Lai's algorithm performs termination detection in "waves", disseminating control messages along a spanning tree (such as an actor supervisor hierarchy) so as to obtain consistent global message send and receive counts. Venkatasubramanian et al. take a similar approach to obtain a consistent global snapshot of actor states in a distributed system [25]. However, such an approach does not scale well because it is not incremental: garbage cannot be detected until all nodes in the system have responded. In contrast, DRL does not require a global snapshot, does not require actors to coordinate their local snapshots, and does not require waiting for all nodes before detecting local terminated actors.

Reference Tracking

We say that an idle actor is *simple garbage* if it has no undelivered messages and no other actor has a reference to it. Such actors can be detected with distributed reference counting [31, 6, 20] or with reference listing [21, 30] techniques. In reference listing algorithms, each actor maintains a partial list of actors that may have references to it. Whenever A sends B a reference to C, it also sends an `info` message informing C about B's reference. Once B no longer needs a reference to C, it informs C by sending a `release` message; this message should not be processed by C until all preceding messages from B to C have been delivered. Thus an actor is simple garbage when its reference listing is empty.

Our technique uses a form of *deferred reference listing*, in which A may also defer sending `info` messages to C until it releases its references to C. This allows `info` and `release` messages to be batched together, reducing communication overhead.

Cyclic Garbage

Actors that are transitively acquainted with one another are said to form cycles. Cycles of terminated actors are called *cyclic garbage* and cannot be detected with reference listing alone. Since actors are hosted on nodes and cycles may span across multiple nodes, detecting cyclic garbage requires sharing information between nodes to obtain a consistent view of the global topology. One approach is to compute a global snapshot of the distributed system [13] using the Chandy-Lamport algorithm [9]; but this requires pausing execution of all actors on a node to compute its local snapshot.

Another approach is to add edges to the actor reference graph so that actor garbage coincides with passive object garbage [24, 29]. This is convenient because it allows existing algorithms for distributed passive object GC, such as [23], to be reused in actor systems. However, such transformations require that actors know when they have undelivered messages, which requires some form of synchronization.

To avoid pausing executions, Wang and Varela proposed a reference listing based technique called the *pseudo-root* algorithm. The algorithm computes *approximate* global snapshots and is implemented in the SALSA runtime [30, 28]. The pseudo-root algorithm requires a high number of additional control messages and requires actors to write to shared memory if they migrate or release references during snapshot collection. Our protocol requires fewer control messages and no additional actions between local actor snapshots. Wang and Varela also explicitly address migration of actors, a concern orthogonal to our algorithm.

Our technique is inspired by *MAC*, a termination detection algorithm implemented in the Pony runtime [10]. In MAC, actors send a local snapshot to a designated cycle detector whenever their message queue becomes empty, and send another notification whenever it becomes non-empty. Clebsch and Drossopoulou prove that for systems with causal message delivery, a simple request-reply protocol is sufficient to confirm that the cycle detector's view of the topology is consistent. However, enforcing causal delivery in a distributed system imposes additional space and networking costs [11, 7]. DRL is similar to MAC, but does not require causal message delivery, supports decentralized termination detection, and actors need not take snapshots each time their message queues become empty. The key insight is that these limitations can be removed by tracking additional information at the actor level.

An earlier version of DRL appeared in [22]. In this paper, we formalize the description of the algorithm and prove its safety and liveness. In the process, we discovered that release acknowledgment messages are unnecessary and that termination detection is more flexible than we first thought: it is not necessary for GC to be performed in distinct "phases" where every actor takes a snapshot in each phase. In particular, once an idle actor takes a snapshot, it need not take another snapshot until it receives a fresh message.

4 A Two-Level Semantic Model

Our computation model is based on the two level approach to actor semantics [26], in which a lower *system-level* transition system interprets the operations performed by a higher, user-facing *application-level* transition system. In this section, we define the DRL communication protocol at the system level. We do not provide a transition system for the application level computation model, since it is not relevant to garbage collection (see [3] for how it can be done). What is relevant to us is that corresponding to each application-level action is a system-level transition that tracks references. We will therefore define *system-level configurations* and *transitions on system-level configurations*. We will refer to these, respectively, as configurations and transitions in the rest of the paper.

4.1 Overview

Actors in DRL use *reference objects* (abbreviated *refobs*) to send messages, instead of using plain actor addresses. Refobs are similar to unidirectional channels and can only be used by their designated *owner* to send messages to their *target*; thus in order for A to give B a reference to C, it must explicitly create a new refob owned by B. Once a refob is no longer needed, it should be *deactivated* by its owner and removed from local state.

The DRL communication protocol enriches each actor's state with a list of refobs that it currently owns and associated message counts representing the number of messages sent using each refob. Each actor also maintains a subset of the refobs of which it is the target, together with associated message receive counts. Lastly, actors perform a form of "contact tracing" by maintaining a subset of the refobs that they have created for other actors; we provide details about the bookkeeping later in this section.

The additional information above allows us to detect termination by inspecting actor snapshots. If a set of snapshots is consistent (in the sense of [9]) then we can use the "contact tracing" information to determine whether the set is *closed* under the potential inverse acquaintance relation (see Section 5). Then, given a consistent and closed set of snapshots, we can use the message counts to determine whether an actor is blocked. We can therefore find all the terminated actors within a consistent set of snapshots.

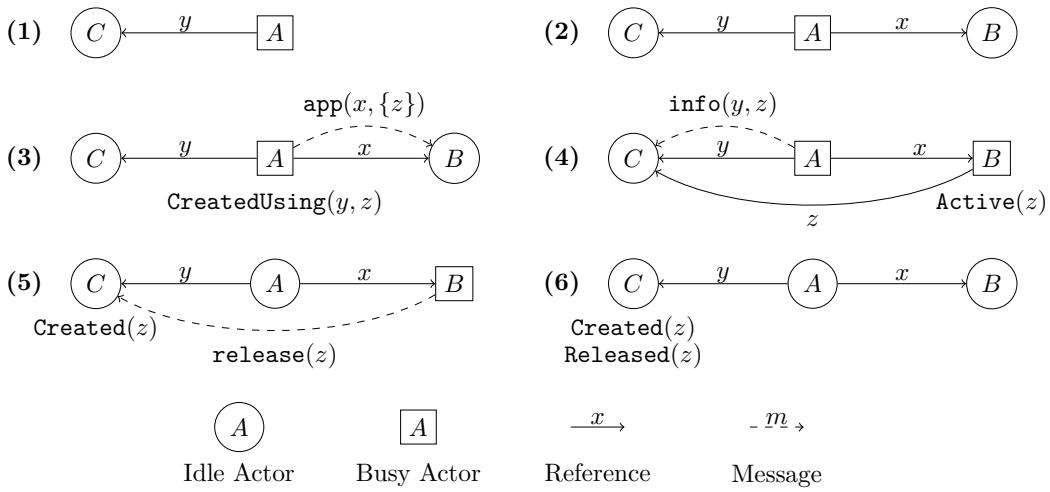


Figure 2 An example showing how refobs are created and destroyed. Below each actor we list all the "facts" related to z that are stored in its local state. Although not pictured in the figure, A also obtains facts $\texttt{Active}(x)$ and $\texttt{Active}(y)$ after spawning actors B and C, respectively. Likewise, actors B, C obtain facts $\texttt{Created}(x), \texttt{Created}(y)$, respectively, upon being spawned.

In fact, DRL satisfies a stronger property: any set of snapshots that "appears terminated" in the sense above is guaranteed to be consistent. Hence, given an arbitrary closed set of snapshots, it is possible to determine which of the corresponding actors have terminated. This allows a great deal of freedom in how snapshots are aggregated. For instance, actors could place their snapshots in a global eventually consistent store, with a garbage collection thread at each node periodically inspecting the store for local terminated actors.

Reference Objects

A refob is a triple (x, A, B), where A is the owner actor's address, B is the target actor's address, and x is a globally unique token. An actor can cheaply generate such a token by combining its address with a local sequence number, since actor systems already guarantee that each address is unique. We will stylize a triple (x, A, B) as $x : A \multimap B$. We will also sometimes refer to such a refob as simply x, since tokens act as unique identifiers.

When an actor A spawns an actor B (Fig. 2 (1, 2)) the DRL protocol creates a new refob $x : A \multimap B$ that is stored in both A and B's system-level state, and a refob $y : B \multimap B$ in B's state. The refob x allows A to send application-level messages to B. These messages are denoted $\texttt{app}(x, R)$, where R is the sett of refobs contained in the message that A has created for B. The refob y corresponds to the `self` variable present in some actor languages.

If A has active refobs $x : A \multimap B$ and $y : A \multimap C$, then it can create a new refob $z : B \multimap C$ by generating a token z. In addition to being sent to B, this refob must also temporarily be stored in A's system-level state and marked as "created using y" (Fig. 2 (3)). Once B receives z, it must add the refob to its system-level state and mark it as "active" (Fig. 2 (4)). Note that B can have multiple distinct refobs that reference the same actor in its state; this can be the result of, for example, several actors concurrently sending refobs to B. Transition rules for spawning actors and sending messages are given in Section 4.3.

Actor A may remove z from its state once it has sent a (system-level) `info` message informing C about z (Fig. 2 (4)). Similarly, when B no longer needs its refob for C, it can "deactivate" z by removing it from local state and sending C a (system-level) `release`

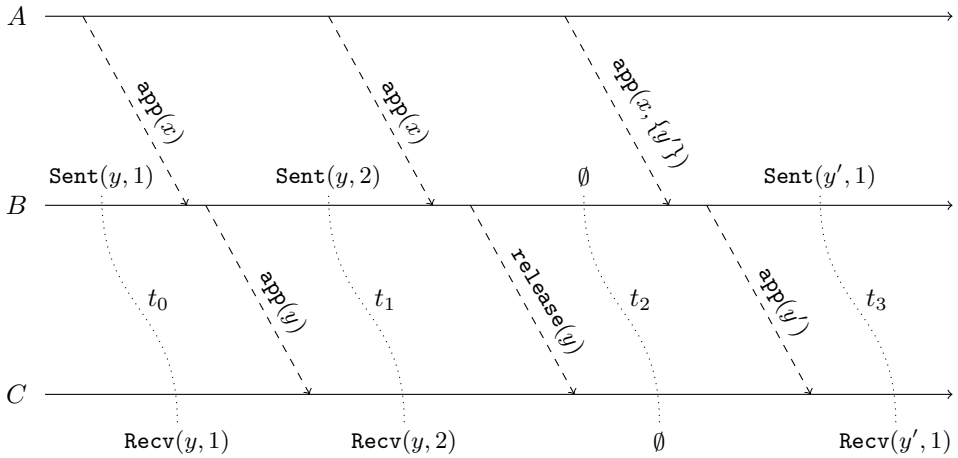


Figure 3 A time diagram for actors A, B, C, demonstrating message counts and consistent snapshots. Dashed arrows represent messages and dotted lines represent consistent cuts. In each cut above, B's message send count agrees with C's message receive count.

message (Fig. 2 (5)). Note that if B already has a refob $z : B \multimap C$ and then receives another $z' : B \multimap C$, then it can be more efficient to defer deactivating the extraneous z' until z is also no longer needed; this way, the `release` messages can be batched together.

When C receives an `info` message, it records that the refob has been created, and when C receives a `release` message, it records that the refob has been released (Fig. 2 (6)). Note that these messages may arrive in any order. Once C has received both, it is permitted to remove all facts about the refob from its local state. Transition rules for these reference listing actions are given in Section 4.4.

Once a refob has been created, it cycles through four states: pending, active, inactive, or released. A refob $z : B \multimap C$ is said to be *pending* until it is received by its owner B. Once received, the refob is *active* until it is *deactivated* by its owner, at which point it becomes *inactive*. Finally, once C learns that z has been deactivated, the refob is said to be *released*. A refob that has not yet been released is *unreleased*.

Slightly amending the definition we gave in Section 2, we say that B is a *potential acquaintance* of A (and A is a *potential inverse acquaintance* of B) when there exists an unreleased refob $x : A \multimap B$. Thus, B becomes a potential acquaintance of A as soon as x is created, and only ceases to be an acquaintance once it has received a `release` message for every refob $y : A \multimap B$ that has been created so far.

Message Counts and Snapshots

For each refob $x : A \multimap B$, the owner A counts the number of `app` and `info` messages sent along x; this count can be deleted when A deactivates x. Each message is annotated with the refob used to send it. Whenever B receives an `app` or `info` message along x, it correspondingly increments a receive count for x; this count can be deleted once x has been released. Thus the memory overhead of message counts is linear in the number of unreleased refobs.

A snapshot is a copy of all the facts in an actor's system-level state at some point in time. We will assume throughout the paper that in every set of snapshots Q, each snapshot was taken by a different actor. Such a set is also said to form a *cut*. Recall that a cut is consistent if no snapshot in the cut causally precedes any other [9]. Let us also say that Q is a set of *mutually quiescent* snapshots if there are no undelivered messages between actors

in the cut. That is, if $A \in Q$ sent a message to $B \in Q$ before taking a snapshot, then the message must have been delivered before B took its snapshot. Notice that if all snapshots in Q are mutually quiescent, then Q is consistent.

Notice also that in Fig. 3, the snapshots of B and C are mutually quiescent when their send and receive counts agree. This is ensured in part because each refob has a unique token: If actors associated message counts with actor names instead of tokens, then B's snapshots at t_0 and t_3 would both contain $\texttt{Sent}(C, 1)$. Thus, B's snapshot at t_3 and C's snapshot at t_0 would appear mutually quiescent, despite having undelivered messages in the cut.

We would like to conclude that snapshots from two actors A, B are mutually quiescent if and only if their send and receive counts are agreed for every refob $x : A \multimap B$ or $y : B \multimap A$. Unfortunately, this fails to hold in general for systems with unordered message delivery. It also fails to hold when, for instance, the owner actor takes a snapshot before the refob is activated and the target actor takes a snapshot after the refob is released. In such a case, neither knowledge set includes a message count for the refob and they therefore appear to agree. However, we show that the message counts can nevertheless be used to bound the number of undelivered messages for purposes of our algorithm (Lemma 12).

Definitions

We use the capital letters A, B, C, D, E to denote actor addresses. Tokens are denoted x, y, z, with a special reserved token `null` for messages from external actors.

A *fact* is a value that takes one of the following forms: $\texttt{Created}(x)$, $\texttt{Released}(x)$, $\texttt{CreatedUsing}(x, y)$, $\texttt{Active}(x)$, $\texttt{Unreleased}(x)$, $\texttt{Sent}(x, n)$, or $\texttt{Received}(x, n)$ for some refobs x, y and natural number n. Each actor's state holds a set of facts about refobs and message counts called its *knowledge set*. We use ϕ, ψ to denote facts and Φ, Ψ to denote finite sets of facts. Each fact may be interpreted as a *predicate* that indicates the occurrence of some past event. Interpreting a set of facts Φ as a set of axioms, we write $\Phi \vdash \phi$ when ϕ is derivable by first-order logic from Φ with the following additional rules:

- If $(\nexists n \in \mathbb{N},\ \texttt{Sent}(x, n) \in \Phi)$ then $\Phi \vdash \texttt{Sent}(x, 0)$
- If $(\nexists n \in \mathbb{N},\ \texttt{Received}(x, n) \in \Phi)$ then $\Phi \vdash \texttt{Received}(x, 0)$
- If $\Phi \vdash \texttt{Created}(x) \wedge \neg\texttt{Released}(x)$ then $\Phi \vdash \texttt{Unreleased}(x)$
- If $\Phi \vdash \texttt{CreatedUsing}(x, y)$ then $\Phi \vdash \texttt{Created}(y)$

For convenience, we define a pair of functions $\mathit{incSent}(x, \Phi)$, $\mathit{incRecv}(x, \Phi)$ for incrementing message send/receive counts, as follows: If $\texttt{Sent}(x, n) \in \Phi$ for some n, then $\mathit{incSent}(x, \Phi) = (\Phi \setminus \{\texttt{Sent}(x, n)\}) \cup \{\texttt{Sent}(x, n + 1)\}$; otherwise, $\mathit{incSent}(x, \Phi) = \Phi \cup \{\texttt{Sent}(x, 1)\}$. Likewise for *incRecv* and `Received`.

Recall that an actor is either *busy* (processing a message) or *idle* (waiting for a message). An actor with knowledge set Φ is denoted $[\Phi]$ if it is busy and (Φ) if it is idle.

Our specification includes both *system messages* (also called *control messages*) and *application messages*. The former are automatically generated by the DRL protocol and handled at the system level, whereas the latter are explicitly created and consumed by user-defined behaviors. Application-level messages are denoted $\texttt{app}(x, R)$. The argument x is the refob used to send the message. The second argument R is a set of refobs created by the sender to be used by the destination actor. Any remaining application-specific data in the message is omitted in our notation.

The DRL communication protocol uses two kinds of system messages. $\texttt{info}(y, z, B)$ is a message sent from an actor A to an actor C, informing it that a new refob $z : B \multimap C$ was created using $y : A \multimap C$. $\texttt{release}(x, n)$ is a message sent from an actor A to an actor B, informing it that the refob $x : A \multimap B$ has been deactivated and should be released.

A *configuration* $\langle\!\langle\ \alpha \mid \mu\ \rangle\!\rangle^{\rho}_{\chi}$ is a quadruple $(\alpha, \mu, \rho, \chi)$ where: α is a mapping from actor addresses to knowledge sets; μ is a mapping from actor addresses to multisets of messages; and ρ, χ are sets of actor addresses. Actors in $\mathrm{dom}(\alpha)$ are *internal actors* and actors in χ are *external actors*; the two sets may not intersect. The mapping μ associates each actor with undelivered messages to that actor. Actors in ρ are *receptionists*. We will ensure $\rho \subseteq \mathrm{dom}(\alpha)$ remains valid in any configuration that is derived from a configuration where the property holds (referred to as the locality laws in [12]).

Configurations are denoted by κ, κ', κ_0, etc. If an actor address A (resp. a token x), does not occur in κ, then the address (resp. the token) is said to be *fresh*. We assume a facility for generating fresh addresses and tokens.

In order to express our transition rules in a pattern-matching style, we will employ the following shorthand. Let $\alpha, [\Phi]_A$ refer to a mapping α' where $\alpha'(A) = [\Phi]$ and $\alpha = \alpha'|_{\mathrm{dom}(\alpha')\setminus\{A\}}$. Similarly, let $\mu, [A \triangleleft m]$ refer to a mapping μ' where $m \in \mu'(A)$ and $\mu = \mu'|_{\mathrm{dom}(\mu')\setminus\{A\}} \cup \{A \mapsto \mu'(A) \setminus \{m\}\}$. Informally, the expression $\alpha, [\Phi]_A$ refers to a set of actors containing both α and the busy actor A (with knowledge set Φ); the expression $\mu, [A \triangleleft m]$ refers to the set of messages containing both μ and the message m (sent to actor A).

The rules of our transition system define atomic transitions from one configuration to another. Each transition rule has a label l, parameterized by some variables $\vec{x}$ that occur in the left- and right-hand configurations. Given a configuration κ, these parameters functionally determine the next configuration κ'. Given arguments $\vec{v}$, we write $\kappa \xrightarrow{l(\vec{v})} \kappa'$ to denote a semantic step from κ to κ' using rule $l(\vec{v})$.

We refer to a label with arguments $l(\vec{v})$ as an *event*, denoted e. A sequence of events is denoted π. If $\pi = e_1, \ldots, e_n$ then we write $\kappa \xrightarrow{\pi} \kappa'$ when $\kappa \xrightarrow{e_1} \kappa_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} \kappa'$. If there exists π such that $\kappa \xrightarrow{\pi} \kappa'$, then κ' is *derivable* from κ. An *execution* is a sequence of events $e_1, \ldots, e_n$ such that $\kappa_0 \xrightarrow{e_1} \kappa_1 \xrightarrow{e_2} \ldots \xrightarrow{e_n} \kappa_n$, where κ_0 is the initial configuration (Section 4.2). We say that a property holds *at time* t if it holds in κ_t.

4.2 Initial Configuration

The initial configuration κ_0 consists of a single actor in a busy state:

$$\langle\!\langle\ [\Phi]_A \mid \emptyset\ \rangle\!\rangle^{\emptyset}_{\{E\}},$$

where $\Phi = \{\texttt{Active}(x : A \multimap E),\ \texttt{Created}(y : A \multimap A),\ \texttt{Active}(y : A \multimap A)\}$. The actor's knowledge set includes a refob to itself and a refob to an external actor E. A can become a receptionist by sending E a refob to itself. Henceforth, we will only consider configurations that are derivable from an initial configuration.

4.3 Standard Actor Operations

Fig. 4 gives transition rules for standard actor operations, such as spawning actors and sending messages. Each of these rules corresponds a rule in the standard operational semantics of actors [3]. Note that each rule is atomic, but can just as well be implemented as a sequence of several smaller steps without loss of generality because actors do not share state – see [3] for a formal proof.

The Spawn event allows a busy actor A to spawn a new actor B and creates two refobs $x : A \multimap B$, $y : B \multimap B$. B is initialized with knowledge about x and y via the facts $\texttt{Created}(x), \texttt{Created}(y)$. The facts $\texttt{Active}(x), \texttt{Active}(y)$ allow A and B to immediately begin sending messages to B. Note that implementing Spawn does not require a synchronization protocol between A and B to construct $x : A \multimap B$. The parent A can pass both its

$\textsc{Spawn}(x, A, B)$

$$\langle\!\langle\ \alpha, [\Phi]_A \mid \mu\ \rangle\!\rangle^{\rho}_{\chi} \to \langle\!\langle\ \alpha, [\Phi \cup \{\texttt{Active}(x : A \multimap B)\}]_A, [\Psi]_B \mid \mu\ \rangle\!\rangle^{\rho}_{\chi}$$

where x, y, B fresh
and $\Psi = \{\texttt{Created}(x : A \multimap B),\ \texttt{Created}(y : B \multimap B),\ \texttt{Active}(y : B \multimap B)\}$

$\textsc{Send}(x, \vec{y}, \vec{z}, A, B, \vec{C})$

$$\langle\!\langle\ \alpha, [\Phi]_A \mid \mu\ \rangle\!\rangle^{\rho}_{\chi} \to \langle\!\langle\ \alpha, [\mathit{incSent}(x, \Phi) \cup \Psi]_A \mid \mu, [B \triangleleft \texttt{app}(x, R)]\ \rangle\!\rangle^{\rho}_{\chi}$$

where $\vec{y}$ and $\vec{z}$ fresh and $n = |\vec{y}| = |\vec{z}| = |\vec{C}|$
and $\Phi \vdash \texttt{Active}(x : A \multimap B)$ and $\forall i \leq n,\ \Phi \vdash \texttt{Active}(y_i : A \multimap C_i)$
and $R = \{z_i : B \multimap C_i \mid i \leq n\}$ and $\Psi = \{\texttt{CreatedUsing}(y_i, z_i) \mid i \leq n\}$

$\textsc{Receive}(x, B, R)$

$$\langle\!\langle\ \alpha, (\Phi)_B \mid \mu, [B \triangleleft \texttt{app}(x, R)]\ \rangle\!\rangle^{\rho}_{\chi} \to \langle\!\langle\ \alpha, [\mathit{incRecv}(x, \Phi) \cup \Psi]_B \mid \mu\ \rangle\!\rangle^{\rho}_{\chi}$$

where $\Psi = \{\texttt{Active}(z) \mid z \in R\}$

$\textsc{Idle}(A)$

$$\langle\!\langle\ \alpha, [\Phi]_A \mid \mu\ \rangle\!\rangle^{\rho}_{\chi} \to \langle\!\langle\ \alpha, (\Phi)_A \mid \mu\ \rangle\!\rangle^{\rho}_{\chi}$$

Figure 4 Rules for standard actor interactions.

address and the freshly generated token x to the constructor for B. Since actors typically know their own addresses, this allows B to construct the triple (x, A, B). Since the `spawn` call typically returns the address of the spawned actor, A can also create the same triple.

The SEND event allows a busy actor A to send an application-level message to B containing a set of refobs $z_1, \ldots, z_n$ to actors $\vec{C} = C_1, \ldots, C_n$ – it is possible that $B = A$ or $C_i = A$ for some i. For each new refob z_i, we say that the message *contains* z_i. Any other data in the message besides these refobs is irrelevant to termination detection and therefore omitted. To send the message, A must have active refobs to both the target actor B and to every actor $C_1, \ldots, C_n$ referenced in the message. For each target C_i, A adds a fact $\texttt{CreatedUsing}(y_i, z_i)$ to its knowledge set; we say that A *created* z_i *using* y_i. Finally, A must increment its `Sent` count for the refob x used to send the message; we say that the message is sent *along* x.

The RECEIVE event allows an idle actor B to become busy by consuming an application message sent to B. Before performing subsequent actions, B increments the receive count for x and adds all refobs in the message to its knowledge set.

Finally, the IDLE event puts a busy actor into the idle state, enabling it to consume another message.

4.4 Release Protocol

Whenever an actor creates or receives a refob, it adds facts to its knowledge set. To remove these facts when they are no longer needed, actors can perform the *release protocol* defined in Fig. 5. All of these rules are not present in the standard operational semantics of actors.

SendInfo(y, z, A, B, C)

$$\langle\!\langle\ \alpha, [\Phi \cup \Psi]_A \mid \mu\ \rangle\!\rangle^{\rho}_{\chi} \to \langle\!\langle\ \alpha, [\mathit{incSent}(y, \Phi)]_A \mid \mu, [C \triangleleft \texttt{info}(y, z, B)]\ \rangle\!\rangle^{\rho}_{\chi}$$

where $\Psi = \{\texttt{CreatedUsing}(y : A \multimap C, z : B \multimap C)\}$

Info(y, z, B, C)

$$\langle\!\langle\ \alpha, (\Phi)_C \mid \mu, [C \triangleleft \texttt{info}(y, z, B)]\ \rangle\!\rangle^{\rho}_{\chi} \to \langle\!\langle\ \alpha, (\mathit{incRecv}(y, \Phi) \cup \Psi)_C \mid \mu\ \rangle\!\rangle^{\rho}_{\chi}$$

where $\Psi = \{\texttt{Created}(z : B \multimap C)\}$

SendRelease(x, A, B)

$$\langle\!\langle\ \alpha, [\Phi \cup \Psi]_A \mid \mu\ \rangle\!\rangle^{\rho}_{\chi} \to \langle\!\langle\ \alpha, [\Phi]_A \mid \mu, [B \triangleleft \texttt{release}(x, n)]\ \rangle\!\rangle^{\rho}_{\chi}$$

where $\Psi = \{\texttt{Active}(x : A \multimap B), \texttt{Sent}(x, n)\}$
and $\nexists y,\ \texttt{CreatedUsing}(x, y) \in \Phi$

Release(x, A, B)

$$\langle\!\langle\ \alpha, (\Phi)_B \mid \mu, [B \triangleleft \texttt{release}(x, n)]\ \rangle\!\rangle^{\rho}_{\chi} \to \langle\!\langle\ \alpha, (\Phi \cup \{\texttt{Released}(x)\})_B \mid \mu\ \rangle\!\rangle^{\rho}_{\chi}$$

only if $\Phi \vdash \texttt{Received}(x, n)$

Compaction(x, B, C)

$$\langle\!\langle\ \alpha, (\Phi \cup \Psi)_C \mid \mu\ \rangle\!\rangle^{\rho}_{\chi} \to \langle\!\langle\ \alpha, (\Phi)_C \mid \mu\ \rangle\!\rangle^{\rho}_{\chi}$$

where $\Psi = \{\texttt{Created}(x : B \multimap C), \texttt{Released}(x : B \multimap C), \texttt{Received}(x, n)\}$ for some $n \in \mathbb{N}$
or $\Psi = \{\texttt{Created}(x : B \multimap C), \texttt{Released}(x : B \multimap C)\}$ and $\forall n \in \mathbb{N},\ \texttt{Received}(x, n) \notin \Phi$

Snapshot(A, Φ)

$$\langle\!\langle\ \alpha, (\Phi)_A \mid \mu\ \rangle\!\rangle^{\rho}_{\chi} \to \langle\!\langle\ \alpha, (\Phi)_A \mid \mu\ \rangle\!\rangle^{\rho}_{\chi}$$

Figure 5 Rules for performing the release protocol.

The SendInfo event allows a busy actor A to inform C about a refob $z : B \multimap C$ that it created using y; we say that the `info` message is sent *along* y and *contains* z. This event allows A to remove the fact $\texttt{CreatedUsing}(y, z)$ from its knowledge set. It is crucial that A also increments its `Sent` count for y to indicate an undelivered `info` message sent to C: it allows the snapshot aggregator to detect when there are undelivered `info` messages, which contain refobs. This message is delivered with the Info event, which adds the fact $\texttt{Created}(z : B \multimap C)$ to C's knowledge set and correspondingly increments C's `Received` count for y.

When an actor A no longer needs $x : A \multimap B$ for sending messages, A can deactivate x with the SendRelease event; we say that the `release` is sent *along* x. A precondition of this event is that A has already sent messages to inform B about all the refobs it has created using x. In practice, an implementation may defer sending any `info` or `release` messages to a target B until all A's refobs to B are deactivated. This introduces a trade-off between the number of control messages and the rate of simple garbage detection (Section 5).

$\textsc{In}(A, R)$

$$\langle\!\langle\ \alpha \mid \mu\ \rangle\!\rangle^{\rho}_{\chi} \to \langle\!\langle\ \alpha \mid \mu, [A \triangleleft \texttt{app}(\texttt{null}, R)]\ \rangle\!\rangle^{\rho}_{\chi \cup \chi'}$$

where $A \in \rho$ and $R = \{x_1 : A \multimap B_1, \ldots, x_n : A \multimap B_n\}$ and $x_1, \ldots, x_n$ fresh
and $\{B_1, \ldots, B_n\} \cap \mathrm{dom}(\alpha) \subseteq \rho$ and $\chi' = \{B_1, \ldots, B_n\} \setminus \mathrm{dom}(\alpha)$

$\textsc{Out}(x, B, R)$

$$\langle\!\langle\ \alpha \mid \mu, [B \triangleleft \texttt{app}(x, R)]\ \rangle\!\rangle^{\rho}_{\chi} \to \langle\!\langle\ \alpha \mid \mu\ \rangle\!\rangle^{\rho \cup \rho'}_{\chi}$$

where $B \in \chi$ and $R = \{x_1 : B \multimap C_1, \ldots, x_n : B \multimap C_n\}$ and $\rho' = \{C_1, \ldots, C_n\} \cap \mathrm{dom}(\alpha)$

$\textsc{ReleaseOut}(x, B)$

$$\langle\!\langle\ \alpha \mid \mu, [B \triangleleft \texttt{release}(x, n)]\ \rangle\!\rangle^{\rho}_{\chi \cup \{B\}} \to \langle\!\langle\ \alpha \mid \mu\ \rangle\!\rangle^{\rho}_{\chi \cup \{B\}}$$

$\textsc{InfoOut}(y, z, A, B, C)$

$$\langle\!\langle\ \alpha \mid \mu, [C \triangleleft \texttt{info}(y, z, A, B)]\ \rangle\!\rangle^{\rho}_{\chi \cup \{C\}} \to \langle\!\langle\ \alpha \mid \mu\ \rangle\!\rangle^{\rho}_{\chi \cup \{C\}}$$

Figure 6 Rules for interacting with the outside world.

Each `release` message for a refob x includes a count n of the number of messages sent using x. This ensures that $\texttt{release}(x, n)$ is only delivered after all the preceding messages sent along x have been delivered. Once the RELEASE event can be executed, it adds the fact that x has been released to B's knowledge set. Once C has received both an `info` and `release` message for a refob x, it may remove facts about x from its knowledge set using the COMPACTION event.

Finally, the SNAPSHOT event captures an idle actor's knowledge set. For simplicity, we have omitted the process of disseminating snapshots to an aggregator. Although this event does not change the configuration, it allows us to prove properties about snapshot events at different points in time.

4.5 Composition and Effects

We give rules to dictate how internal actors interact with external actors in Fig. 6. The IN and OUT rules correspond to similar rules in the standard operational semantics of actors.

Since internal garbage collection protocols are not exposed to the outside world, all `release` and `info` messages sent to external actors are simply dropped by the RELEASEOUT and INFOOUT events. Likewise, only `app` messages can enter the system. Since we cannot statically determine when a receptionist's address has been forgotten by all external actors, we assume that receptionists are never terminated. The resulting "black box" behavior of our system is the same as the actor systems in [3]. Hence, in principle DRL can be gradually integrated into a codebase by creating a subsystem for garbage-collected actors.

The IN event allows an external actor to send an application-level message to a receptionist A containing a set of refobs R, all owned by A. Since external actors do not use refobs, the message is sent using the special `null` token. All targets in R that are not internal actors are added to the set of external actors.

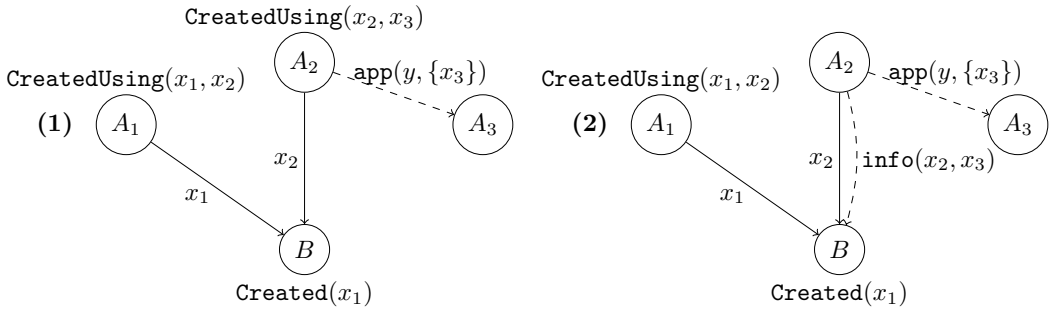


Figure 7 An example of a chain from B to x_3.

The OUT event delivers an application-level message to an external actor with a set of refobs R. All internal actors referenced in R become receptionists because their addresses have been exposed to the outside world.

4.6 Garbage

We can now operationally characterize actor garbage in our model. An actor A can *potentially receive a message* in κ if there is a sequence of events (possibly of length zero) leading from κ to a configuration κ' in which A has an undelivered message. We say that an actor is *terminated* if it is idle and cannot potentially receive a message.

An actor is *blocked* if it satisfies three conditions: (1) it is idle, (2) it is not a receptionist, and (3) it has no undelivered messages; otherwise, it is *unblocked*. We define *potential reachability* as the reflexive transitive closure of the potential acquaintance relation. That is, A_1 can potentially reach A_n if and only if there is a sequence of unreleased refobs $(x_1 : A_1 \multimap A_2), \ldots, (x_n : A_{n-1} \multimap A_n)$; recall that a refob $x : A \multimap B$ is unreleased if its target B has not yet received a `release` message for x.

Notice that an actor can potentially receive a message if and only if it is potentially reachable from an unblocked actor. Hence an actor is terminated if and only if it is only potentially reachable by blocked actors. A special case of this is *simple garbage*, in which an actor is blocked and has no potential inverse acquaintances besides itself.

We say that a set of actors S is *closed* (with respect to the potential inverse acquaintance relation) if, whenever $B \in S$ and there is an unreleased refob $x : A \multimap B$, then also $A \in S$. Notice that the closure of a set of terminated actors is also a set of terminated actors.

5 Chain Lemma

To determine if an actor has terminated, one must show that all of its potential inverse acquaintances have terminated. This appears to pose a problem for termination detection, since actors cannot have a complete listing of all their potential inverse acquaintances without some synchronization: actors would need to consult their acquaintances before creating new references to them. In this section, we show that the DRL protocol provides a weaker guarantee that will nevertheless prove sufficient: knowledge about an actor's refobs is *distributed* across the system and there is always a "path" from the actor to any of its potential inverse acquaintances.

Let us construct a concrete example of such a path, depicted by Fig. 7. Suppose that A_1 spawns B, gaining a refob $x_1 : A_1 \multimap B$. Then A_1 may use x_1 to create $x_2 : A_2 \multimap B$, which A_2 may receive and then use x_2 to create $x_3 : A_3 \multimap B$.

At this point, there are unreleased refobs owned by A_2 and A_3 that are not included in B's knowledge set. However, Fig. 7 shows that the distributed knowledge of B, A_1, A_2 creates a "path" to all of B's potential inverse acquaintances. Since A_1 spawned B, B knows the fact $\texttt{Created}(x_1)$. Then when A_1 created x_2, it added the fact $\texttt{CreatedUsing}(x_1, x_2)$ to its knowledge set, and likewise A_2 added the fact $\texttt{CreatedUsing}(x_2, x_3)$; each fact points to another actor that owns an unreleased refob to B (Fig. 7 (1)).

Since actors can remove `CreatedUsing` facts by sending `info` messages, we also consider (Fig. 7 (2)) to be a "path" from B to A_3. But notice that, once B receives the `info` message, the fact $\texttt{Created}(x_3)$ will be added to its knowledge set and so there will be a "direct path" from B to A_3. We formalize this intuition with the notion of a *chain* in a given configuration $\langle\!\langle\ \alpha \mid \mu\ \rangle\!\rangle^{\rho}_{\chi}$:

▶ **Definition 1.** *A* chain to $x : A \multimap B$ *is a sequence of unreleased refobs* $(x_1 : A_1 \multimap B), \ldots,$ $(x_n : A_n \multimap B)$ *such that:*

- $\alpha(B) \vdash \texttt{Created}(x_1 : A_1 \multimap B)$*;*
- *For all* $i < n$*, either* $\alpha(A_i) \vdash \texttt{CreatedUsing}(x_i, x_{i+1})$ *or the message* $[B \triangleleft \texttt{info}(x_i, x_{i+1})]$ *is in transit; and*
- $A_n = A$ *and* $x_n = x$*.*

We say that an actor B is *in the root set* if it is a receptionist or if there is an application message $\texttt{app}(x, R)$ in transit to an external actor with $B \in \text{targets}(R)$. Since external actors never release refobs, actors in the root set must never terminate.

▶ **Lemma 2** (Chain Lemma)**.** *Let* B *be an internal actor in* κ*. If* B *is not in the root set, then there is a chain to every unreleased refob* $x : A \multimap B$*. Otherwise, there is a chain to some refob* $y : C \multimap B$ *where* C *is an external actor.*

▶ Remark. When B is in the root set, not all of its unreleased refobs are guaranteed to have chains. This is because an external actor may send B's address to other receptionists without sending an `info` message to B.

An immediate application of the Chain Lemma is to allow actors to detect when they are simple garbage. If any actor besides B owns an unreleased refob to B, then B must have a fact $\texttt{Created}(x : A \multimap B)$ in its knowledge set where $A \neq B$. Hence, if B has no such facts, then it must have no nontrivial potential inverse acquaintances. Moreover, since actors can only have undelivered messages along unreleased refobs, B also has no undelivered messages from any other actor; it can only have undelivered messages that it sent to itself. This gives us the following result:

▶ **Theorem 3.** *Suppose* B *is idle with knowledge set* Φ*, such that:*

- Φ *does not contain any facts of the form* $\texttt{Created}(x : A \multimap B)$ *where* $A \neq B$*; and*
- *for all facts* $\texttt{Created}(x : B \multimap B) \in \Phi$*, also* $\Phi \vdash \texttt{Sent}(x, n) \wedge \texttt{Received}(x, n)$ *for some* n*.*

Then B *is simple garbage.*

6 Termination Detection

In order to detect non-simple terminated garbage, actors periodically sends a snapshot of their knowledge set to a snapshot aggregator actor. An aggregator in turn may disseminate snapshots it has to other aggregators. Each aggregator maintains a map data structure, associating an actor's address to its most recent snapshot; in effect, snapshot aggregators

maintain an eventually consistent key-value store with addresses as keys and snapshots as values. At any time, an aggregator can scan its local store to find terminated actors and send them a request to self-destruct.

Given an arbitrary set of snapshots Q, we characterize the *finalized subsets* of Q in this section. We show that the actors that took these finalized snapshots must be terminated. Conversely, the snapshots of any closed set of terminated actors are guaranteed to be finalized. (Recall that the closure of a set of terminated actors is also a terminated set of actors.) Thus, snapshot aggregators can eventually detect all terminated actors by periodically searching their local stores for finalized subsets. Finally, we give an algorithm for obtaining the maximum finalized subset of a set Q by "pruning away" the snapshots of actors that appear not to have terminated.

Recall that when we speak of a set of snapshots Q, we assume each snapshot was taken by a different actor. We will write $\Phi_A \in Q$ to denote A's snapshot in Q; we will also write $A \in Q$ if A has a snapshot in Q. We will also write $Q \vdash \phi$ if $\Phi \vdash \phi$ for some $\Phi \in Q$.

▶ **Definition 4.** *A set of snapshots Q is* closed *if, whenever $Q \vdash \texttt{Unreleased}(x : A \multimap B)$ and $B \in Q$, then also $A \in Q$ and $\Phi_A \vdash \texttt{Active}(x : A \multimap B)$.*

▶ **Definition 5.** *An actor $B \in Q$* appears blocked *if, for every $Q \vdash \texttt{Unreleased}(x : A \multimap B)$, then $\Phi_A, \Phi_B \in Q$ and $\Phi_A \vdash \texttt{Sent}(x, n)$ and $\Phi_B \vdash \texttt{Received}(x, n)$ for some n.*

▶ **Definition 6.** *A set of snapshots Q is* finalized *if it is closed and every actor in Q appears blocked.*

This definition corresponds to our characterization in Section 4.6: An actor is terminated precisely when it is in a closed set of blocked actors.

▶ **Theorem 7** (Safety)**.** *If Q is a finalized set of snapshots at time t_f then the actors in Q are all terminated at t_f.*

We say that the *final action* of a terminated actor is the last non-snapshot event it performs before becoming terminated. Notice that an actor's final action can only be an Idle, Info, or Release event. Note also that the final action may come *strictly before* an actor becomes terminated, since a blocked actor may only terminate after all of its potential inverse acquaintances become blocked.

The following lemma allows us to prove that DRL is eventually live. It also shows that an non-finalized set of snapshots must have an unblocked actor.

▶ **Lemma 8.** *Let S be a closed set of terminated actors at time t_f. If every actor in S took a snapshot sometime after its final action, then the resulting set of snapshots is finalized.*

▶ **Theorem 9** (Liveness)**.** *If every actor eventually takes a snapshot after performing an Idle, Info, or Release event, then every terminated actor is eventually part of a finalized set of snapshots.*

Proof. If an actor A is terminated, then the closure S of $\{A\}$ is a terminated set of actors. Since every actor eventually takes a snapshot after taking its final action, Lemma 8 implies that the resulting snapshots of S are finalized. ◀

We say that a refob $x : A \multimap B$ is *unreleased* in Q if $Q \vdash \texttt{Unreleased}(x)$. Such a refob is said to be *relevant* when $B \in Q$ implies $A \in Q$ and $\Phi_A \vdash \texttt{Active}(x)$ and $\Phi_A \vdash \texttt{Sent}(x, n)$ and $\Phi_B \vdash \texttt{Received}(x, n)$ for some n; intuitively, this indicates that B has no undelivered messages along x. Notice that a set Q is finalized if and only if all unreleased refobs in Q are relevant.

Observe that if $x : A \multimap B$ is unreleased and irrelevant in Q, then B cannot be in any finalized subset of Q. We can therefore employ a simple iterative algorithm to find the maximum finalized subset of Q: for each irrelevant unreleased refob $x : A \multimap B$ in Q, remove the target B from Q. Since this can make another unreleased refob $y : B \multimap C$ irrelevant, we must repeat this process until a fixed point is reached. In the resulting subset Q', all unreleased refobs are relevant. Since all actors in $Q \setminus Q'$ are not members of any finalized subset of Q, it must be that Q' is the maximum finalized subset of Q.

7 Conclusion and Future Work

We have shown how deferred reference listing and message counts can be used to detect termination in actor systems. The technique is provably safe (Theorem 7) and eventually live (Theorem 9). An implementation in Akka is presently underway.

We believe that DRL satisfies our three initial goals:

1. *Termination detection does not restrict concurrency in the application.* Actors do not need to coordinate their snapshots or pause execution during garbage collection.
2. *Termination detection does not impose high overhead.* The amortized memory overhead of our technique is linear in the number of unreleased refobs. Besides application messages, the only additional control messages required by the DRL communication protocol are `info` and `release` messages. These control messages can be batched together and deferred, at the cost of worse termination detection time.
3. *Termination detection scales with the number of nodes in the system.* Our algorithm is incremental, decentralized, and does not require synchronization between nodes.

Since it does not matter what order snapshots are collected in, DRL can be used as a "building block" for more sophisticated garbage collection algorithms. One promising direction is to take a *generational* approach [15], in which long-lived actors take snapshots less frequently than short-lived actors. Different types of actors could also take snapshots at different rates. In another approach, snapshot aggregators could *request* snapshots instead of waiting to receive them.

In the presence of faults, DRL remains safe but its liveness properties are affected. If an actor A crashes and its state cannot be recovered, then none of its refobs can be released and the aggregator will never receive its snapshot. Consequently, all actors potentially reachable from A can no longer be garbage collected. However, A's failure does not affect the garbage collection of actors it cannot reach. In particular, network partitions between nodes will not delay node-local garbage collection.

Choosing an adequate fault-recovery protocol will likely vary depending on the target actor framework. One option is to use checkpointing or event-sourcing to persist GC state; the resulting overhead may be acceptable in applications that do not frequently spawn actors or create refobs. Another option is to monitor actors for failure and infer which refobs are no longer active; this is a subject for future work.

Another issue that can affect liveness is message loss: If any messages along a refob $x : A \multimap B$ are dropped, then B can never be garbage collected because it will always appear unblocked. This is, in fact, the desired behavior if one cannot guarantee that the message will not be delivered at some later point. In practice, this problem might be addressed with watermarking.

References

1. Gul Agha. *ACTORS – a Model of Concurrent Computation in Distributed Systems*. MIT Press Series in Artificial Intelligence. MIT Press, 1990.
2. Gul Agha. Concurrent object-oriented programming. *Communications of the ACM*, 33(9):125–141, September 1990. doi:10.1145/83880.84528.
3. Gul A. Agha, Ian A. Mason, Scott F. Smith, and Carolyn L. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7(1):1–72, January 1997. doi:10.1017/S095679689700261X.
4. Akka. URL: https://akka.io/.
5. Joe Armstrong, Robert Virding, Claes Wikström, and Mike Williams. *Concurrent Programming in ERLANG*. Prentice Hall, Englewood Cliffs, New Jersey, second edition, 1996.
6. Di Bevan. Distributed garbage collection using reference counting. In G. Goos, J. Hartmanis, D. Barstow, W. Brauer, P. Brinch Hansen, D. Gries, D. Luckham, C. Moler, A. Pnueli, G. Seegmüller, J. Stoer, N. Wirth, J. W. Bakker, A. J. Nijman, and P. C. Treleaven, editors, *PARLE Parallel Architectures and Languages Europe*, volume 259, pages 176–187. Springer Berlin Heidelberg, Berlin, Heidelberg, 1987. doi:10.1007/3-540-17945-3_10.
7. Sebastian Blessing, Sylvan Clebsch, and Sophia Drossopoulou. Tree topologies for causal message delivery. In *Proceedings of the 7th ACM SIGPLAN International Workshop on Programming Based on Actors, Agents, and Decentralized Control - AGERE 2017*, pages 1–10, Vancouver, BC, Canada, 2017. ACM Press. doi:10.1145/3141834.3141835.
8. Sergey Bykov, Alan Geller, Gabriel Kliot, James R. Larus, Ravi Pandya, and Jorgen Thelin. Orleans: Cloud computing for everyone. In *Proceedings of the 2nd ACM Symposium on Cloud Computing - SOCC '11*, pages 1–14, Cascais, Portugal, 2011. ACM Press. doi:10.1145/2038916.2038932.
9. K. Mani Chandy and Leslie Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, February 1985. doi:10.1145/214451.214456.
10. Sylvan Clebsch and Sophia Drossopoulou. Fully concurrent garbage collection of actors on many-core machines. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications - OOPSLA '13*, pages 553–570, Indianapolis, Indiana, USA, 2013. ACM Press. doi:10.1145/2509136.2509557.
11. Colin J Fidge. Timestamps in message-passing systems that preserve the partial ordering. *Australian Computer Science Communications*, 10(1):56–66, February 1988.
12. Carl Hewitt and Henry G. Baker. Laws for communicating parallel processes. In Bruce Gilchrist, editor, *Information Processing, Proceedings of the 7th IFIP Congress 1977, Toronto, Canada, August 8-12, 1977*, pages 987–992. North-Holland, 1977.
13. D. Kafura, M. Mukherji, and D.M. Washabaugh. Concurrent and distributed garbage collection of active objects. *IEEE Transactions on Parallel and Distributed Systems*, 6(4):337–350, April 1995. doi:10.1109/71.372788.
14. Ten-Hwang Lai. Termination detection for dynamically distributed systems with non-first-in-first-out communication. *Journal of Parallel and Distributed Computing*, 3(4):577–599, December 1986. doi:10.1016/0743-7315(86)90015-8.
15. Henry Lieberman and Carl Hewitt. A real-time garbage collector based on the lifetimes of objects. *Commun. ACM*, 26(6):419–429, 1983. doi:10.1145/358141.358147.
16. Jeff Matocha and Tracy Camp. A taxonomy of distributed termination detection algorithms. *Journal of Systems and Software*, 43(3):207–221, November 1998. doi:10.1016/S0164-1212(98)10034-1.
17. Friedemann Mattern. Algorithms for distributed termination detection. *Distributed Computing*, 2(3):161–175, September 1987. doi:10.1007/BF01782776.
18. NHS to Deploy Riak for New IT Backbone With Quality of Care Improvements in Sight, October 2013. https://riak.com/nhs-to-deploy-riak-for-new-it-backbone-with-quality-of-care-improvements-in-sight.html.

19 PayPal Blows Past 1 Billion Transactions Per Day Using Just 8 VMs With Akka, Scala, Kafka and Akka Streams. https://www.lightbend.com/case-studies/paypal-blows-past-1-billion-transactions-per-day-using-just-8-vms-and-akka-scala-kafka-and-akka-streams.

20 José M. Piquer. Indirect Reference Counting: A Distributed Garbage Collection Algorithm. In Emile H. L. Aarts, Jan van Leeuwen, and Martin Rem, editors, *Parle '91 Parallel Architectures and Languages Europe*, volume 505, pages 150–165. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991. doi:10.1007/978-3-662-25209-3_11.

21 David Plainfossé and Marc Shapiro. A survey of distributed garbage collection techniques. In *Memory Management, International Workshop IWMM 95, Kinross, UK, September 27-29, 1995, Proceedings*, pages 211–249, 1995. doi:10.1007/3-540-60368-9_26.

22 Dan Plyukhin and Gul Agha. Concurrent garbage collection in the actor model. In *Proceedings of the 8th ACM SIGPLAN International Workshop on Programming Based on Actors, Agents, and Decentralized Control - AGERE 2018*, pages 44–53, Boston, MA, USA, 2018. ACM Press. doi:10.1145/3281366.3281368.

23 M. Schelvis. Incremental distribution of timestamp packets: A new approach to distributed garbage collection. In *Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications - OOPSLA '89*, pages 37–48, New Orleans, Louisiana, United States, 1989. ACM Press. doi:10.1145/74877.74883.

24 Abhay Vardhan and Gul Agha. Using passive object garbage collection algorithms for garbage collection of active objects. *ACM SIGPLAN Notices*, 38(2 supplement):106, February 2003. doi:10.1145/773039.512443.

25 Nalini Venkatasubramanian, Gul Agha, and Carolyn Talcott. Scalable distributed garbage collection for systems of active objects. In Yves Bekkers and Jacques Cohen, editors, *Memory Management*, volume 637, pages 134–147. Springer-Verlag, Berlin/Heidelberg, 1992. doi:10.1007/BFb0017187.

26 Nalini Venkatasubramanian and Carolyn Talcott. Reasoning about meta level activities in open distributed systems. In *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing - PODC '95*, pages 144–152, Ottowa, Ontario, Canada, 1995. ACM Press. doi:10.1145/224964.224981.

27 Stanislav Vishnevskiy. How Discord Scaled Elixir to 5,000,000 Concurrent Users, July 2017. URL: https://blog.discord.com/scaling-elixir-f9b8e1e7c29b.

28 Wei-Jen Wang. Conservative snapshot-based actor garbage collection for distributed mobile actor systems. *Telecommunication Systems*, June 2011. doi:10.1007/s11235-011-9509-1.

29 Wei-Jen Wang, Carlos Varela, Fu-Hau Hsu, and Cheng-Hsien Tang. Actor Garbage Collection Using Vertex-Preserving Actor-to-Object Graph Transformations. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Paolo Bellavista, Ruay-Shiung Chang, Han-Chieh Chao, Shin-Feng Lin, and Peter M. A. Sloot, editors, *Advances in Grid and Pervasive Computing*, volume 6104, pages 244–255. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. doi:10.1007/978-3-642-13067-0_28.

30 Wei-Jen Wang and Carlos A. Varela. Distributed Garbage Collection for Mobile Actor Systems: The Pseudo Root Approach. In Yeh-Ching Chung and José E. Moreira, editors, *Advances in Grid and Pervasive Computing*, volume 3947, pages 360–372. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. doi:10.1007/11745693_36.

31 Paul Watson and Ian Watson. An efficient garbage collection scheme for parallel computer architectures. In G. Goos, J. Hartmanis, D. Barstow, W. Brauer, P. Brinch Hansen, D. Gries, D. Luckham, C. Moler, A. Pnueli, G. Seegmüller, J. Stoer, N. Wirth, J. W. Bakker, A. J. Nijman, and P. C. Treleaven, editors, *PARLE Parallel Architectures and Languages Europe*, volume 259, pages 432–443. Springer Berlin Heidelberg, Berlin, Heidelberg, 1987. doi:10.1007/3-540-17945-3_25.

A Appendix

A.1 Basic Properties

▶ **Lemma 10.** *If B has undelivered messages along $x : A \multimap B$, then x is an unreleased refob.*

Proof. There are three types of messages: `app`, `info`, and `release`. All three messages can only be sent when x is active. Moreover, the RELEASE rule ensures that they must all be delivered before x can be released. ◀

▶ **Lemma 11.**

- *Once* $\texttt{CreatedUsing}(y : A \multimap C, z : B \multimap C)$ *is added to A's knowledge set, it will not be removed until after A has sent an* `info` *message containing z to C.*
- *Once* $\texttt{Created}(z : B \multimap C)$ *is added to C's knowledge set, it will not be removed until after C has received the (unique)* `release` *message along z.*
- *Once* $\texttt{Released}(z : B \multimap C)$ *is added to C's knowledge set, it will not be removed until after C has received the (unique)* `info` *message containing z.*

Proof. Immediate from the transition rules. ◀

▶ **Lemma 12.** *Consider a refob $x : A \multimap B$. Let t_1, t_2 be times such that x has not yet been deactivated at t_1 and x has not yet been released at t_2. In particular, t_1 and t_2 may be before the creation time of x.*

Suppose that $\alpha_{t_1}(A) \vdash \texttt{Sent}(x, n)$ and $\alpha_{t_2}(B) \vdash \texttt{Received}(x, m)$ and, if $t_1 < t_2$, that A does not send any messages along x during the interval $[t_1, t_2]$. Then the difference $\max(n - m, 0)$ is the number of messages sent along x before t_1 that were not received before t_2.

Proof. Since x is not deactivated at time t_1 and unreleased at time t_2, the message counts were never reset by the SENDRELEASE or COMPACTION rules. Hence n is the number of messages A sent along x before t_1 and m is the number of messages B received along x before t_2. Hence $\max(n - m, 0)$ is the number of messages sent before t_1 and *not* received before t_2. ◀

A.2 Chain Lemma

▶ **Lemma 2** (Chain Lemma)**.** *Let B be an internal actor in κ. If B is not in the root set, then there is a chain to every unreleased refob $x : A \multimap B$. Otherwise, there is a chain to some refob $y : C \multimap B$ where C is an external actor.*

Proof. We prove that the invariant holds in the initial configuration and at all subsequent times by induction on events $\kappa \xrightarrow{e} \kappa'$, omitting events that do not affect chains. Let $\kappa = \langle\!\langle\ \alpha \mid \mu\ \rangle\!\rangle^{\rho}_{\chi}$ and $\kappa' = \langle\!\langle\ \alpha' \mid \mu'\ \rangle\!\rangle^{\rho'}_{\chi'}$.

In the initial configuration, the only refob to an internal actor is $y : A \multimap A$. Since A knows $\texttt{Created}(y : A \multimap A)$, the invariant is satisfied.

In the cases below, let x, y, z, A, B, C be free variables, not referencing the variables used in the statement of the lemma.

- SPAWN(x, A, B) creates a new unreleased refob $x : A \multimap B$, which satisfies the invariant because $\alpha'(B) \vdash \texttt{Created}(x : A \multimap B)$.

- $\textsc{Send}(x, \vec{y}, \vec{z}, A, B, \vec{C})$ creates a set of refobs R. Let $(z : B \multimap C) \in R$, created using $y : A \multimap C$.
  If C is already in the root set, then the invariant is trivially preserved. Otherwise, there must be a chain $(x_1 : A_1 \multimap C), \ldots, (x_n : A_n \multimap C)$ where $x_n = y$ and $A_n = A$. Then $x_1, \ldots, x_n, z$ is a chain in κ', since $\alpha'(A_n) \vdash \texttt{CreatedUsing}(x_n, z)$.
  If B is an internal actor, then this shows that every unreleased refob to C has a chain in κ'. Otherwise, C is in the root set in κ'. To see that the invariant still holds, notice that $z : B \multimap C$ is a witness of the desired chain.
- $\textsc{SendInfo}(y, z, A, B, C)$ removes the $\texttt{CreatedUsing}(y, z)$ fact but also sends $\texttt{info}(y, z, B)$, so chains are unaffected.
- $\textsc{Info}(y, z, B, C)$ delivers $\texttt{info}(y, z, B)$ to C and adds $\texttt{Created}(z : B \multimap C)$ to its knowledge set.
  Suppose $z : B \multimap C$ is part of a chain $(x_1 : A_1 \multimap C), \ldots, (x_n : A_n \multimap C)$, i.e. $x_i = y$ and $x_{i+1} = z$ and $A_{i+1} = B$ for some $i < n$. Since $\alpha'(C) \vdash \texttt{Created}(x_{i+1} : A_{i+1} \multimap C)$, we still have a chain $x_{i+1}, \ldots, x_n$ in κ'.
- $\textsc{Release}(x, A, B)$ releases the refob $x : A \multimap B$. Since external actors never release their refobs, both A and B must be internal actors.
  Suppose the released refob was part of a chain $(x_1 : A_1 \multimap B), \ldots, (x_n : A_n \multimap B)$, i.e. $x_i = x$ and $A_i = A$ for some $i < n$. We will show that $x_{i+1}, \ldots, x_n$ is a chain in κ'.
  Before performing $\textsc{SendRelease}(x_i, A_i, B)$, A_i must have performed the $\textsc{Info}(x_i, x_{i+1}, A_{i+1}, B)$ event. Since the $\texttt{info}$ message was sent along x_i, Lemma 10 ensures that the message must have been delivered before the present $\textsc{Release}$ event. Furthermore, since x_{i+1} is an unreleased refob in κ', Lemma 11 ensures that $\alpha'(B) \vdash \texttt{Created}(x_{i+1} : A_{i+1} \multimap B)$.
- $\textsc{In}(A, R)$ adds a message from an external actor to the internal actor A. This event can only create new refobs that point to receptionists, so it preserves the invariant.
- $\textsc{Out}(x, B, R)$ emits a message $\texttt{app}(x, R)$ to the external actor B. Since all targets in R are already in the root set, the invariant is preserved. ◀

A.3 Termination Detection

Given a set of snapshots Q taken before some time t_f, we write Q_t to denote those snapshots in Q that were taken before time $t < t_f$. If $\Phi_A \in Q$, we denote the time of A's snapshot as t_A.

▶ **Lemma 8.** *Let S be a closed set of terminated actors at time t_f. If every actor in S took a snapshot sometime after its final action, then the resulting set of snapshots is finalized.*

Call this set of snapshots Q. First, we prove the following lemma.

▶ **Lemma 13.** *If $Q \vdash \texttt{Unreleased}(x : A \multimap B)$ and $B \in Q$, then x is unreleased at t_B.*

Proof. By definition, $Q \vdash \texttt{Unreleased}(x : A \multimap B)$ only if $Q \vdash \texttt{Created}(x) \wedge \neg\texttt{Released}(x)$. Since $Q \nvdash \texttt{Released}(x)$, we must also have $\Phi_B \nvdash \texttt{Released}(x)$. For $Q \vdash \texttt{Created}(x)$, there are two cases.

Case 1: $\Phi_B \vdash \texttt{Created}(x)$. Since $\Phi_B \nvdash \texttt{Released}(x)$, Lemma 11 implies that x is unreleased at time t_B.

Case 2: For some $C \in Q$ and some y, $\Phi_C \vdash \texttt{CreatedUsing}(y, x)$. Since C performed its final action before taking its snapshot, this implies that C will never send the $\texttt{info}$ message containing x to B.

Suppose then for a contradiction that x is released at time t_B. Since $\Phi_B \not\vdash \texttt{Released}(x)$, Lemma 11 implies that B received an `info` message containing x before its snapshot. But this is impossible because C never sends this message. ◀

Proof (Lemma 8). By strong induction on time t, we show that Q is closed and that every actor appears blocked.

Induction hypothesis: For all times $t' < t$, if $B \in Q_{t'}$ and $Q \vdash \texttt{Unreleased}(x : A \multimap B)$, then $A \in Q$, $Q \vdash \texttt{Active}(x)$, and $Q \vdash \texttt{Sent}(x, n)$ and $Q \vdash \texttt{Received}(x, n)$ for some n.

Since $Q_0 = \emptyset$, the induction hypothesis holds trivially in the initial configuration.

Now assume the induction hypothesis. Suppose that $B \in Q$ takes its snapshot at time t with $Q \vdash \texttt{Unreleased}(x : A \multimap B)$, which implies $Q \vdash \texttt{Created}(x) \wedge \neg\texttt{Released}(x)$.

$Q \vdash \texttt{Created}(x)$ implies that x was created before t_f. Lemma 13 implies that x is also unreleased at time t_f, since B cannot perform a RELEASE event after its final action. Hence A is in the closure of $\{B\}$ at time t_f, so $A \in Q$.

Now suppose $\Phi_A \not\vdash \texttt{Active}(x)$. Then either x will be activated after t_A or x was deactivated before t_A. The former is impossible because A would need to become unblocked to receive x. Since x is unreleased at time t_f and $t_A < t_f$, the latter implies that there is an undelivered `release` message for x at time t_f. But this is impossible as well, since B is blocked at t_f.

Finally, let n such that $\Phi_B \vdash \texttt{Received}(x, n)$; we must show that $\Phi_A \vdash \texttt{Sent}(x, n)$. By the above arguments, x is active at time t_A and unreleased at time t_B. Since both actors performed their final action before their snapshots, all messages sent before t_A must have been delivered before t_B. By Lemma 12, this implies $\Phi_A \vdash \texttt{Sent}(x, n)$. ◀

We now prove the safety theorem, which states that if Q is a finalized set of snapshots, then the corresponding actors of Q are terminated. We do this by showing that at each time t, all actors in Q_t are blocked and all of their potential inverse acquaintances are in Q.

Consider the first actor B in Q to take a snapshot. We show, using the Chain Lemma, that the closure of this actor is in Q. Then, since all potential inverse acquaintances of B take snapshots strictly after t_B, it is impossible for B to have any undelivered messages without appearing unblocked.

For every subsequent actor B to take a snapshot, we make a similar argument with an additional step: If B has any potential inverse acquaintances in Q_{t_B}, then they could not have sent B a message without first becoming unblocked.

▶ **Theorem 7** (Safety). *If Q is a finalized set of snapshots at time t_f then the actors in Q are all terminated at t_f.*

Proof. Proof by induction on events. The induction hypothesis consists of two clauses that must both be satisfied at all times $t \leq t_f$.

- **IH 1:** If $B \in Q_t$ and $x : A \multimap B$ is unreleased, then $Q \vdash \texttt{Unreleased}(x)$.
- **IH 2:** The actors of Q_t are all blocked.

Initial configuration. Since $Q_0 = \emptyset$, the invariant trivially holds.

Snapshot(B, Φ_B). Suppose $B \in Q$ takes a snapshot at time t. We show that if $x : A \multimap B$ is unreleased at time t, then $Q \vdash \texttt{Unreleased}(x)$ and there are no undelivered messages along x from A to B. We do this with the help of two lemmas.

▶ **Lemma 14.** *If $Q \vdash \texttt{Unreleased}(x : A \multimap B)$, then x is unreleased at time t and there are no undelivered messages along x at time t. Moreover, if $t_A > t$, then there are no undelivered messages along x throughout the interval $[t, t_A]$.*

Proof (Lemma). Since Q is closed, we have $A \in Q$ and $\Phi_A \vdash \texttt{Active}(x)$. Since B appears blocked, we must have $\Phi_A \vdash \texttt{Sent}(x, n)$ and $\Phi_B \vdash \texttt{Received}(x, n)$ for some n.

Suppose $t_A > t$. Since $\Phi_A \vdash \texttt{Active}(x)$, x is not deactivated and not released at t_A or t. Hence, by Lemma 12, every message sent along x before t_A was received before t. Since message sends precede receipts, each of those messages was sent before t. Hence there are no undelivered messages along x throughout $[t, t_A]$.

Now suppose $t_A < t$. Since $\Phi_A \vdash \texttt{Active}(x)$, x is not deactivated and not released at t_A. By IH 2, A was blocked throughout the interval $[t_A, t]$, so it could not have sent a `release` message. Hence x is not released at t. By Lemma 12, all messages sent along x before t_A must have been delivered before t. Hence, there are no undelivered messages along x at time t. ◀

▶ **Lemma 15.** *Let $x_1 : A_1 \multimap B, \ldots, x_n : A_n \multimap B$ be a chain to $x : A \multimap B$ at time t. Then $Q \vdash \texttt{Unreleased}(x)$.*

Proof (Lemma). Since all refobs in a chain are unreleased, we know $\forall i \leq n,\ \Phi_B \nvdash \texttt{Released}(x_i)$ and so $Q \nvdash \texttt{Released}(x_i)$. It therefore suffices to prove, by induction on the length of the chain, that $\forall i \leq n,\ Q \vdash \texttt{Created}(x_i)$.

Base case: By the definition of a chain, $\alpha_t(B) \vdash \texttt{Created}(x_1)$, so $\texttt{Created}(x_1) \in \Phi_B$.

Induction step: Assume $Q \vdash \texttt{Unreleased}(x_i)$, which implies $A_i \in Q$. Let t_i be the time of A_i's snapshot.

By the definition of a chain, either the message $[B \triangleleft \texttt{info}(x_i, x_{i+1})]$ is in transit at time t, or $\alpha_t(A_i) \vdash \texttt{CreatedUsing}(x_i, x_{i+1})$. But the first case is impossible by Lemma 14, so we only need to consider the latter.

Suppose $t_i > t$. Lemma 14 implies that A_i cannot perform the $\textsc{SendInfo}(x_i, x_{i+1}, A_{i+1}, B)$ event during $[t, t_i]$. Hence $\alpha_{t_i}(A_i) \vdash \texttt{CreatedUsing}(x_i, x_{i+1})$, so $Q \vdash \texttt{Created}(x_{i+1})$.

Now suppose $t_i < t$. By IH 2, A_i must have been blocked throughout the interval $[t_i, t]$. Hence A_i could not have created any refobs during this interval, so x_{i+1} must have been created before t_i. This implies $\alpha_{t_i}(A_i) \vdash \texttt{CreatedUsing}(x_i, x_{i+1})$ and therefore $Q \vdash \texttt{Created}(x_{i+1})$. ◀

Lemma 15 implies that B cannot be in the root set. If it were, then by the Chain Lemma there would be a refob $y : C \multimap B$ with a chain where C is an external actor. Since $Q \vdash \texttt{Unreleased}(y)$, there would need to be a snapshot from C in Q – but external actors do not take snapshots, so this is impossible.

Since B is not in the root set, there must be a chain to every unreleased refob $x : A \multimap B$. By Lemma 15, $Q \vdash \texttt{Unreleased}(x)$. By Lemma 14, there are no undelivered messages to B along x at time t. Since B can only have undelivered messages along unreleased refobs (Lemma 10), the actor is indeed blocked.

Send$(x, \vec{y}, \vec{z}, A, B, \vec{C})$. In order to maintain IH 2, we must show that if $B \in Q_t$ then this event cannot occur. So suppose $B \in Q_t$. By IH 1, we must have $Q \vdash \texttt{Unreleased}(x : A \multimap B)$, so $A \in Q$. By IH 2, we moreover have $A \notin Q_t$ – otherwise A would be blocked and unable to send this message. Since B appears blocked in Q, we must have $\Phi_A \vdash \texttt{Sent}(x, n)$ and $\Phi_B \vdash \texttt{Received}(x, n)$ for some n. Since x is not deactivated at t_A and unreleased at t_B, Lemma 12 implies that every message sent before t_A is received before t_B. Hence A cannot send this message to B because $t_A > t > t_B$.

In order to maintain IH 1, suppose that one of the refobs sent to B in this step is $z : B \multimap C$, where $C \in Q_t$. Then in the next configuration, $\texttt{CreatedUsing}(y, z)$ occurs in A's knowledge set. By the same argument as above, $A \in Q \setminus Q_t$ and $\Phi_A \vdash \texttt{Sent}(y, n)$ and $\Phi_C \vdash \texttt{Received}(y, n)$ for some n. Hence A cannot perform the $\textsc{SendInfo}(y, z, A, B, C)$ event before t_A, so $\Phi_A \vdash \texttt{CreatedUsing}(y, z)$ and $Q \vdash \texttt{Created}(z)$.

SendInfo(y,z,A,B,C). By the same argument as above, $A \notin Q_t$ cannot send an `info` message to $B \in Q_t$ without violating message counts, so IH 2 is preserved.

SendRelease(x, A, B). Suppose that $A \notin Q_t$ and $B \in Q_t$. By IH 1, $x : A \multimap B$ is unreleased at time t. Since Q is finalized, $\Phi_A \vdash \texttt{Active}(x)$. Hence A cannot deactivate x and IH 2 is preserved.

In(A, R). Since every potential inverse acquaintance of an actor in Q_t is also in Q, none of the actors in Q_t is a receptionist. Hence this rule does not affect the invariants.

Out(x, B, R). Suppose $(y : B \multimap C) \in R$ where $C \in Q_t$. Then y is unreleased and $Q \vdash \texttt{Unreleased}(y)$ and $B \in Q$. But this is impossible because external actors do not take snapshots. ◀

Session Subtyping and Multiparty Compatibility Using Circular Sequents

Ross Horne
Computer Science, University of Luxembourg, Esch-sur-Alzette, Luxembourg
ross.horne@uni.lu

Abstract

We present a structural proof theory for multi-party sessions, exploiting the expressive power of non-commutative logic which can capture explicitly the message sequence order in sessions. The approach in this work uses a more flexible form of subtyping than standard, for example, allowing a single thread to be substituted by multiple parallel threads which fulfil the role of the single thread. The resulting subtype system has the advantage that it can be used to capture compatibility in the multiparty setting (addressing limitations of pairwise duality). We establish standard results: that the type system is algorithmic, that multiparty compatible processes which are race free are also deadlock free, and that subtyping is sound with respect to the substitution principle. Interestingly, each of these results can be established using cut elimination. We remark that global types are optional in this approach to typing sessions; indeed we show that this theory can be presented independently of the concept of global session types, or even named participants.

2012 ACM Subject Classification Theory of computation → Type theory; Theory of computation → Proof theory; Theory of computation → Linear logic; Theory of computation → Process calculi

Keywords and phrases session types, subtyping, compatibility, linear logic, deadlock freedom

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.12

Acknowledgements This paper benefits hugely from discussions with Mariangiola Dezani-Ciancaglini and Paola Giannini who suggested restricting to a regular calculus.

1 Introduction

Session types are a class of type systems for modelling protocols that prescribe, not only the types of messages exchanged, but also the sequence in which they are communicated. The first session type systems were constrained to two parties. For such binary sessions, given a session type prescribing the behaviour of each of the participants, it is possible to determine whether the two behaviours are compatible, in the sense that they can interact together to successfully realise a protocol.

Here, in the introduction, we first make it clear there are obvious, underexploited, connections between compatibility in the binary setting and provability in non-commutative extensions of linear logic. The body of this work shows that these observations extend elegantly to the multiparty setting [32, 33], where multiparty compatibility is the problem of whether two or more participants realise a protocol when they communicate together.

On the binary setting and non-commutative logic. In the binary setting, compatibility holds when the two parties are dual to each other [30]. For example, $!\lambda_1;(?\lambda_2 \wedge ?\lambda_3)$ is dual to $?\lambda_1;(!\lambda_2 \vee !\lambda_3)$. The former types a process that is ready to output a message of type λ_1, and then receives either a message of type λ_2 or λ_3. The latter types a process that is ready to receive a message of type λ_1, and then makes a choice between two branches, sending a message of type λ_2 or λ_3. By building subtyping into the system [24, 23, 18], duality becomes a more flexible concept. For example, two processes of respective types $!\lambda_1;(?\lambda_2 \wedge ?\lambda_3)$ and

$?\lambda_1;!\lambda_2$ are also compatible. Notice a process of the type $!\lambda_1;(?\lambda_2 \wedge ?\lambda_3)$ offers two possible inputs, so is more than capable of responding correctly to $?\lambda_1;!\lambda_2$, which always chooses to send λ_2 as its second action.

For binary sessions, compatibility is proven by showing that the dual of a type is a subtype of another type, for example establishing $!\lambda_1;(?\lambda_2 \wedge ?\lambda_3) \leq !\lambda_1;?\lambda_2$. In the original paper on session types [30], it was explicit that internal and external choice were inspired by the additive operators in linear logic [27, 26, 1]. For example, interpreting $\wedge$ as additive conjunction in linear logic, subtype relation $?\lambda_2 \wedge ?\lambda_3 \leq ?\lambda_2$ is a provable implication in linear logic. While pure linear logic has no concept of sequentiality (all operators are commutative), linear logic can be extended with non-commutative operators explicitly capturing sequentiality, allowing the above subtype judgement involving prefixing to be proven. In this work, we restrict ourselves to a fragment of non-commutative logic with action prefixing only, allowing us to retain a sequent calculus presentation. Full sequential composition can be achieved [34]. However, for full sequential composition, it is necessary [50] to employ the calculus of structures [28]. The compromise adopted in this work, of restricting non-commutative logic to prefixing, allows us to formulate our subtype system using the sequent calculus, whilst still working within a fragment of a conservative extension of linear logic.

Contribution to the multiparty setting. Using non-commutative extensions of linear logic to model multiparty session types provides additional expressive power. In particular, the subtype system obtained allows more session types to be compared than possible using established subtype systems [25]. Indeed the subtype system obtained is sufficiently rich, so that subtyping can be used to evaluate compatibility in the multi-party setting. The notion of *multiparty compatibility* enforced by this methodology allows session types to be used to guarantee that multiparty sessions are deadlock free without the need for a global type choreographing all processes. An advantage of avoiding global types is that we can check compatibility for protocols for which no global type exists [48].

Problems with pairwise duality resolved. Early work on multi-party session types [8, 21] employed a notion of compatibility based on the notion of duality for binary types applied pairwise. In that early work, we take each pair of participants and *restrict them only to the inputs and outputs between the participants selected*, and then check whether each pair of projections are dual. Pair-wise duality fails to guarantee deadlock freedom, since process $?\lambda_1;!\lambda_2 \,\|\, ?\lambda_2;!\lambda_3 \,\|\, ?\lambda_3;!\lambda_1$ deadlocks, despite participants being pair-wise dual (e.g., restricting the first two participants to their mutual communications gives types $!\lambda_2$ and $?\lambda_2$, which are dual). The process above consists of three participants in parallel each waiting to receive a message, from another process before producing an output. The process is clearly deadlocked since all inputs await a message that never arrives.

The current work, and related work [20, 15, 48, 39, 19], addresses the above limitation of pair-wise duality by proposing more sophisticated notions of *multi-party compatibility*. The work on which this builds [15] (which concerned a finite fragment of Scribble [31]), handles multiparty compatibility as a special case of subtyping. In this work, as required, our example processes in the previous paragraph would **not** be multi-party compatible. The rules of the system in this paper are determined by logical principles (cut-elimination).

Related paradigms. This paper does not follow the Curry-Howard inspired proofs-as-processes school; instead, it follows a processes-as-formulae [10, 36] approach closer to intersection types [44] and algorithmic subtyping [47]. For multiparty sessions, the processes-

as-formulae [15] and proofs-as-processes paradigms [12, 11] emerged simultaneously. Papers following the Curry-Howard approach typically aim to design new (higher-order) session calculi where the process terms are proofs in an established logic. In contrast, in the processes-as-formulae approach pursued here, we typically harness the power of structural proof theory to design new logics that can directly embed established session calculi [17], while respecting their semantics. In this work, linear implication in the logical system introduced provides us with a notion of session subtyping preserving deadlock freedom.

Summary. In Section 2, we explain how the notion of multiparty subtyping is more flexible than established notions of subtyping for multiparty sessions, illustrated using an example where a participant is substituted by two participants. Section 3 formally develops a theory of session subtyping and multiparty compatibility in a coinductive sequent calculus. That section concludes with an example where we guarantee the deadlock freedom of a session for which no global type exists.

2 Motivating Example: A Generalised Substitution Principle

The problem of defining a subtype system for multiparty sessions is *in a sense* solved in the synchronous setting [14, 25]. Soundness in that work is defined according to a *substitution principle* [41], informally stated in related work [25] as: "If $\mathsf{T}\ \mathcal{R}\ \mathsf{T}'$, then a process of type T' engaged in a well-typed session may be safely replaced with a process of type T." Here $\mathcal{R}$ is a candidate subtype relation and "safely" is formalised in terms of deadlock freedom.

In the above related work, the substitution principle allows one (single threaded) participant to replace another participant. In the current paper, we take a broader interpretation of the substitution principle, permitting more parallelism to be introduced. We allow participants in a session to be replaced by any number of participants, e.g., a single thread of type T can be replaced by two parallel participants of type T_1 and T_2, where $\mathsf{T}_1 \otimes \mathsf{T}_2 \leq \mathsf{T}$. This allows parallel components to be introduced with additional communications, while preserving the ability of the multiple components to fulfil the role of the original components. An example is provided next.

An authorisation protocol. We provide an example that is out of scope of the substitution principle in related work mentioned above, but within the scope of the substitution principal in the current paper. In the example that follows, we consider an application where a *Trusted App* is replaced by an *Untrusted App* and an *OAuth Server*. This demands a rich multi-party subtype system accounting for parallelism and interactions.

Consider the protocol realised by the three participants in Fig. 1, which are modelled as threads in a typical session calculus. In this authorisation protocol, the *Trusted App* asks the *Owner* of a resource for permission before it accesses the *Resource*.

Owner: This could be you – the human user, who owns the resource. You get redirected to a login page containing the `app_id` for the *Trusted App* and a `scope` indicating the resources requested (e.g., personal contact details). If you chose to approve authorisation, you grant access to the resource by providing your `name` and `password`. You do however have the ability to chose not to approve, choosing the branch $!$`deny` in the **internal choice**, notated $\oplus$ in the process *Owner* in Fig. 1.

Resource: A token is used by the *Trusted App* to prove it has the right to access the resource. The *Resource* can be accessed many times by the *Trusted App* until the token expires or is revoked. The expiry of a token is modelled here by the *Resource* making an internal choice, deciding whether to provide data or revoke.

$$
\begin{array}{ll}
\textit{Owner:} & ?\mathtt{login\_page}(\mathtt{app\_id},\mathtt{scope});(!\mathtt{deny} \oplus !\mathtt{authorise}(\mathtt{name},\mathtt{password})) \\
\textit{Resource:} & \mathtt{rec}X.(?\mathtt{release} + ?\mathtt{request}(\mathtt{token});(!\mathtt{revoke} \oplus !\mathtt{response}(\mathtt{data});X)) \\
\textit{Trusted App:} & !\mathtt{login\_page}(\mathtt{app\_id},\mathtt{scope}); \\
& ?\mathtt{deny};!\mathtt{release} + \ ?\mathtt{authorise}(\mathtt{name},\mathtt{password}); \\
& \qquad\qquad \mathtt{rec}Y.!\mathtt{release} \oplus \ !\mathtt{request}(\mathtt{token}); \\
& \qquad\qquad\qquad\qquad ?\mathtt{revoke} + ?\mathtt{response}(\mathtt{data});Y
\end{array}
$$

Figure 1 The local behaviours of three participants in an authorisation protocol.

Trusted App: Since the App is trusted it presents directly the login page to the user. If the *Resource Owner* approves, the same App manufactures a token which is used to access the resource. Notice **external choice**, notated +, is used for inputs.

A problem with the above protocol is that user credentials are provided directly to the *Trusted App*. Furthermore, the *Trusted App* does not only know the credentials of the owner of the resource, it must also know how to manufacture tokens to access the resource; hence, in principle, has the right to freely access the resource without asking permission. Thus, there is no security offered to the *Resource Owner* or *Resource* if the app is compromised.

Substituting one participant with two participants. We can address the above limitation by making use of the OAuth 2.0 protocol [29] where credentials and the generation of tokens is handled by an *OAuth Server* that the *Owner* trusts more than the app. We can refine the above protocol by substituting *Trusted App* with two processes in parallel: an *Untrusted App* and *OAuth Server*, defined in Fig. 2.

$$
\begin{array}{l}
\textit{Untrusted App:} \\
!\mathtt{initiate}(\mathtt{app\_id},\mathtt{scope}); \\
?\mathtt{close} + \ ?\mathtt{authorisation\_code}(\mathtt{code}); \\
\qquad\quad !\mathtt{exchange}(\mathtt{app\_id},\mathtt{secret},\mathtt{code}); \\
\qquad\quad ?\mathtt{close} + \ ?\mathtt{access\_token}(\mathtt{token}); \\
\qquad\qquad\quad \mathtt{rec}Y.!\mathtt{request}(\mathtt{token});(?\mathtt{revoke} + ?\mathtt{response}(\mathtt{data});Y) \\
\textit{OAuth Server:} \\
?\mathtt{initiate}(\mathtt{app\_id},\mathtt{scope}); \\
!\mathtt{login\_page}(\mathtt{app\_id},\mathtt{scope}); \\
(?\mathtt{deny};!\mathtt{close};!\mathtt{release}) + \ ?\mathtt{authorise}(\mathtt{name},\mathtt{password}); \\
\qquad\qquad\qquad\quad (!\mathtt{close};!\mathtt{release}) \oplus \ !\mathtt{authorisation\_code}(\mathtt{code}); \\
\qquad\qquad\qquad\qquad\qquad\quad ?\mathtt{exchange}(\mathtt{app\_id},\mathtt{secret},\mathtt{code}); \\
\qquad\qquad\qquad\qquad\qquad\quad (!\mathtt{close};!\mathtt{release}) \oplus \ !\mathtt{access\_token}(\mathtt{token})
\end{array}
$$

Figure 2 Two participants that can safely replace the *Trusted App* in Fig. 1.

The OAuth protocol enables the *Untrusted App* to access the *Resource*, for which permission is required from the *Owner*, in such a way that the *Owner* never discloses their credentials to the *Untrusted App*. The *Owner* permits the *OAuth Server* to grant an access token to the *Untrusted App* that can be used to access the *Resource*. We briefly describe informally each process.

OAuth Server: As a mediator between the *Untrusted App* and *Resource Owner*, the *OAuth Server* receives an initiate request from the *Untrusted App*, resulting in the *Resource Owner* being redirected to a login page. Notice the *OAuth Server* reacts to the decision of the *Resource Owner* to either provide credentials or end the session, indicated by an *external* choice. Notice, after that point, that the server makes two internal choices: the first issuing

a `code` to the *Untrusted App* only if the correct credentials were provided by the *Owner*; the second issuing an access token only if the *Untrusted App* provides its correct credentials (and the correct `code`). If all is correct, a `token` is eventually issued to the *Untrusted App*.

Untrusted App: The *Untrusted App* initiates the protocol. It then reacts, indicated by external choices, to whether the *Resource Owner* and *OAuth Server* grant access. If an access `token` is granted, the token can be used repeatedly to access the resource requested.

What the subtype system guarantees here. The *Trusted App* can be replaced by *Untrusted App* $\|$ *OAuth Server* while preserving deadlock freedom of the protocol. We know this because the type of *App* $\|$ *OAuth* is a subtype of the type of *Trusted App*, by using the subtype system introduce in the next section. Furthermore, for protocols of the complexity of this OAuth example, it is not immediately obvious whether all roles are correctly implemented such that deadlock freedom is guaranteed. We can also use the subtype system introduced in the next section to check whether participants together are multiparty compatible.

3 A Proof System for Subtyping and Multiparty Compatibility

In this section, we introduce session types and a proof system for expressing session types called Session, which defines our subtype system for multiparty sessions. Later in this section, having introduced Session, we define multiparty compatibility and race freedom, and use these properties to establish our main deadlock freedom result.

Session types are defined according to the following syntax. Note we could have propositional data types (*nat*, *bool*, etc.), but accommodating such data types is a perpendicular issue to this work, hence we simply label messages (λ_1, λ_2, etc.).

▶ **Definition 1** (session types)**.** Session types for threads *are defined by:*

$$L \quad ::= \quad \textstyle\bigwedge_{i\in I} ?\lambda_i;L_i \;\mid\; \bigvee_{i\in I} !\lambda_i;L_i \;\mid\; \mu \boldsymbol{t}.L \;\mid\; \boldsymbol{t} \;\mid\; \mathtt{OK}$$

Session types for networks *are defined by:*

$$N \quad ::= \quad L \;\mid\; N \parr N \;\mid\; N \otimes N$$

We refer to both of the above simply as session types*, which are ranged over by* T, U, V*. We restrict ourselves to guarded recursion, avoiding the type* $\mu \boldsymbol{t}.\boldsymbol{t}$*. Index sets* I *are finite.*

The constant OK is used to type networks that, on all paths, either successfully terminate or progress forever. Intersection types (abbreviated as $\wedge$ when there are two branches) are used to type external choices between inputs; while union types (abbreviated as $\vee$) type internal choices between outputs.

Actions π are either of the form $!\lambda$ or $?\lambda$. Whenever there is only one branch in a union/intersection type, we simply write the action prefixed type $\pi;\mathsf{T}$, which is used to type a process that performs an input or output and then behaves as T. As standard, we allow OK to be omitted, by abbreviating $\pi;\mathtt{OK}$ as π.

Notably, the syntax features two commutative multiplicative operators $\parr$ and $\otimes$. When typing multiparty sessions we employ only $\mathsf{T} \otimes \mathsf{U}$, representing two parallel sessions T and U that may communicate and interleave actions. The operator $\mathsf{T} \parr \mathsf{U}$ is introduced to complete the theory, as the dual to parallel composition, and is used in subtyping proofs. Future work may also use $\parr$ as an additional modelling device that prevents one session from interfering with another session. As a consequence of including the pair of multiplicatives, every session type, has a dual type, its co-type, given by the function below.

▶ **Definition 2** (co-type). *Co-types are defined by the following mapping over types, prefixed types and actions:*

$$\overline{\bigwedge_{i\in I} \mathsf{T}_i} = \bigvee_{i\in I} \overline{\mathsf{T}_i} \qquad \overline{\bigvee_{i\in I} \mathsf{T}_i} = \bigwedge_{i\in I} \overline{\mathsf{T}_i} \qquad \overline{\pi;\mathsf{T}} = \overline{\pi};\overline{\mathsf{T}} \qquad \overline{!\lambda} = ?\lambda \qquad \overline{?\lambda} = !\lambda$$
$$\overline{\mathsf{T} \otimes \mathsf{U}} = \overline{\mathsf{T}} \parr \overline{\mathsf{U}} \qquad \overline{\mathsf{T} \parr \mathsf{U}} = \overline{\mathsf{T}} \otimes \overline{\mathsf{U}} \qquad \overline{\mu \boldsymbol{t}.\mathsf{T}} = \mu \boldsymbol{t}.\overline{\mathsf{T}} \qquad \overline{\boldsymbol{t}} = \boldsymbol{t} \qquad \overline{\mathtt{OK}} = \mathtt{OK}$$

In addition to the duality between the multiplicatives, described above, the de Morgan duality between $\vee$ and $\wedge$ is standard for session types. The co-type of a prefix action interchanges send and receive, and dualises the continuation. The unit $\mathtt{OK}$ is self-dual. Since we have only guarded recursion, we treat fixed points equi-recursively, hence the fixed point operator is self-dual. Intuitively, equi-recursive types are treated equivalently to their infinite unfoldings.

Note co-types and the use of two multiplicatives is optional in this work. Having co-types reduces the number of rules in the next section by avoiding two sided sequents.

3.1 Deriving subtype judgements using the rules of Session

The rules of Session are defined in Fig. 3, using, in proof theoretic terms, a circular (or cyclic) sequent calculus [9, 4] – which is, in type theoretical terms, a coinductive algorithmic subtype system [47]. We employ an explicit algorithmic presentation of such a circular system where we have an axiom [Leaf] which is enabled whenever there is a loop in the proof returning to a sequent visited earlier in the proof. This algorithmic approach to coinduction is standard in type theory [2], being sound and complete for infinite proofs such as these due to the restriction to guarded recursion.

We explain the notation $[\Theta]\,\Gamma \vdash$. The sequent Γ is a (comma separated) multiset of types, hence types in a sequent can commute (exchange) inside a sequent, but cannot be duplicated (contraction) or removed (weakening). A **set of sequents** Θ, where each sequent in the set is separated using $[\![$, is employed to define an algorithmic coinductive system, by remembering sequents that may be revisited. We omit Θ if it is empty.

▶ Remark 3. Note that proof systems typically formalise *provability of formulae*, written $\vdash \mathsf{T}$. For a tight match with session type conventions (without breaking the logical convention that $\wedge$ is conjunctive), we instead formulate *provability of duals of formulae*. To emphasise that we formulate probability of duals we write sequents as $\mathsf{T} \vdash$, which is equivalent to $\vdash \overline{\mathsf{T}}$.

Subtypes. Using co-types (Def. 2) and the rules in Fig. 3, subtyping can be defined as follows. Note, a type is closed when no type variables appear free.

▶ **Definition 4** (subtyping). *We say a closed type* T *is a subtype of another closed type* U*, written* $\mathsf{T} \leq \mathsf{U}$*, whenever* $\mathsf{T}\,, \overline{\mathsf{U}} \vdash$ *holds in* Session*.*

Note that in linear logic a linear implication $\mathsf{T} \multimap \mathsf{U}$ holds whenever $\overline{\mathsf{T} \otimes \overline{\mathsf{U}}}$ is provable. Translating to provability of duals, proving $\overline{\mathsf{T} \otimes \overline{\mathsf{U}}}$ is equivalent to establishing $\mathsf{T}\,, \overline{\mathsf{U}} \vdash$. Indeed subtyping as defined above is a conservative extension of linear implication in linear logic (with the mix rule). In what follows, we confirm that standard subtype judgements are covered by the above definition. In addition, some additional subtype judgements hold, which are particular to the multiparty setting.

We briefly highlight that most rules are standard rules from linear logic and coinductive proof systems. Examples appear in the next section. Rules are well-defined over closed types.

$$\text{[OK]}\quad [\Theta]\,\mathtt{ok}\,,\,\mathtt{ok}\,,\,\ldots\,\mathtt{ok}\vdash \qquad \text{[Leaf]}\quad [\Theta \,]\!]\,\Gamma]\,\Gamma\vdash \qquad \text{[Fix-}\mu\text{]}\quad \frac{[\Theta \,]\!]\,\mu\mathbf{t}.\mathsf{T}\,,\,\Gamma]\,\mathsf{T}\{^{\mu\mathbf{t}.\mathsf{T}}/_{\mathbf{t}}\}\,,\,\Gamma\vdash}{[\Theta]\,\mu\mathbf{t}.\mathsf{T}\,,\,\Gamma\vdash}$$

$$\text{[Meet]}\quad \frac{[\Theta]\,?\lambda_j;\mathsf{T}_j\,,\,\Gamma\vdash \quad \text{for } \mathbf{some}\ j\in I}{[\Theta]\,\bigwedge_{i\in I}?\lambda_j;\mathsf{T}_i\,,\,\Gamma\vdash} \qquad \text{[Join]}\quad \frac{[\Theta]\,!\lambda_j;\mathsf{T}_j\,,\,\Gamma\vdash \quad \text{for } \mathbf{all}\ j\in I}{[\Theta]\,\bigvee_{i\in I}!\lambda_j;\mathsf{T}_i\,,\,\Gamma\vdash}$$

$$\text{[Prefix]}\quad \frac{[\Theta]\,\mathsf{T}\,,\,\mathsf{U}\,,\,\Gamma\vdash}{[\Theta]\,!\lambda;\mathsf{T}\,,\,?\lambda;\mathsf{U}\,,\,\Gamma\vdash} \qquad \text{[Times]}\quad \frac{[\Theta]\,\mathsf{T}\,,\,\mathsf{U}\,,\,\Gamma\vdash}{[\Theta]\,\mathsf{T}\otimes\mathsf{U}\,,\,\Gamma\vdash} \qquad \text{[Par]}\quad \frac{[\Theta]\,\mathsf{T}\,,\,\Gamma_1\vdash \quad [\Theta]\,\mathsf{U}\,,\,\Gamma_2\vdash}{[\Theta]\,\mathsf{T}\parr\mathsf{U}\,,\,\Gamma_1\,,\,\Gamma_2\vdash}$$

Figure 3 A presentation of the algorithmic coinductive proof system Session. Note, to align with session type conventions, the system establishes provability of duals.

Rules from MALL. Most rules of Session are rules of Multiplicative Additive Linear Logic (MALL), dualised in order to formalise provability of duals. The rule [Times] breaks down types into their parallel components. The rule [Par] is required for subtyping in the presence of parallelism. The axiom [OK] indicates that a protocol with no more actions has successfully terminated (this rule is valid for MALL with mix). Rules [Join] and [Meet] are (dualised) standard rules for the additives of linear logic.

Rules for equi-recursion. Fixed points can be unfolded using the rule [Fix-μ]. Axiom [Leaf] is applied when we reach a previously visited sequent, completing a loop.

Rule [Prefix]. The exception to the above established rules for equi-recursion and MALL is the [Prefix] rule. This is used to model an interaction between two processes where one sends and the other receives. The rule enforces a causal order on interactions.

3.2 On notable admissible rules and algorithmic subtyping

For a proof system, we say a rule is *admissible*, whenever anything provable in the system with the rule present is provable in the same system but with the rule removed. We highlight the following three notable rules that are admissible in Session.

$$\text{[Cut]}\quad \frac{[\Theta]\,\Gamma_1\,,\,\mathsf{T}\vdash \quad [\Theta]\,\overline{\mathsf{T}}\,,\,\Gamma_2\vdash}{[\Theta]\,\Gamma_1\,,\,\Gamma_2\vdash} \qquad \text{[Intr]}\quad \frac{I\subseteq J \quad [\Theta]\,\mathsf{T}_k\,,\,\mathsf{U}_k\,,\,\Gamma\vdash \quad \text{for } \mathbf{all}\ k\in I}{[\Theta]\,\bigvee_{i\in I}!\lambda_i;\mathsf{T}_i\,,\,\bigwedge_{j\in J}?\lambda_j;\mathsf{U}_j\,,\,\Gamma\vdash} \qquad \text{[Mix]}\quad \frac{[\Theta]\,\Gamma_1\vdash \quad [\Theta]\,\Gamma_2\vdash}{[\Theta]\,\Gamma_1\,,\,\Gamma_2\vdash}$$

Cut elimination and algorithmic subtyping. The admissibility of [Cut], called cut elimination, is the corner stone of proof theory, since many results in logic (e.g., the consistency of classical logic) can be proven as corollaries of cut elimination. Since cut elimination justifies that rules are consistently defined, we present cut elimination in Session as a theorem.

▶ **Theorem 5** (cut elimination). *The* [Cut] *rule is admissible in Session.*

To see that the above holds, observe that, trivially, the unfolding of a proof in Session to infinite proofs (over infinitely unfolded terms) is sound, and, due to regularity, complete (i.e., an infinite proof will always eventually loop on every branch, allowing [Leaf] to be applied). Thus it is sufficient to show that cut elimination holds for the finite proof system. This follows by observing that the standard normalisation steps for MALL, plus cases for [Prefix], can be applied to unfold a cut free proof to an arbitrary depth. We show only the principal case for [Prefix], which is given by the following proof normalisation step.

$$\dfrac{\dfrac{\Gamma_1 , \mathsf{U} , \mathsf{T} \vdash}{\Gamma_1 , ?\lambda;\mathsf{U} , !\lambda;\mathsf{T} \vdash} \qquad \dfrac{\overline{\mathsf{T}} , \mathsf{V} , \Gamma_2 \vdash}{?\lambda;\overline{\mathsf{T}} , !\lambda;\mathsf{V} , \Gamma_2 \vdash}}{\Gamma_1 , ?\lambda;\mathsf{U} , !\lambda;\mathsf{V} , \Gamma_2 \vdash}\,[\textsc{Cut}] \quad \rightsquigarrow \quad \dfrac{\dfrac{\Gamma_1 , \mathsf{U} , \mathsf{T} \vdash \qquad \overline{\mathsf{T}} , \mathsf{V} , \Gamma_2 \vdash}{\Gamma_1 , \mathsf{U} , \mathsf{V} , \Gamma_2 \vdash}\,[\textsc{Cut}]}{\Gamma_1 , ?\lambda;\mathsf{U} , !\lambda;\mathsf{V} , \Gamma_2 \vdash}$$

An immediate consequence of cut elimination for session types is that subtyping relation $\leq$ is transitive. It is also reflexive by a simple induction on the structure of types.

▶ **Corollary 6.** *If* $\mathsf{T} \leq \mathsf{U}$ *and* $\mathsf{U} \leq \mathsf{V}$*, then* $\mathsf{T} \leq \mathsf{V}$*. Also, we have* $\mathsf{T} \leq \mathsf{T}$*.*

From the perspective of type theory this is a standard result that **must** hold in order to recommend an *algorithmic subtype system*. An algorithmic subtype system is expressed without a cut (or transitivity) rule, since cut violates what is known as the *sub-formula property*. The sub-formula property states that every formula appearing in the premise is a sub-formula of one of formulae appearing in the conclusion (up to unfolding of equi-recursion, which is allowed here due to regularity). The sub-formula property guarantees that proof search in Session terminates.

Admissibility of [Intr]. Established algorithmic subtype systems usually employ a rule of the form [Intr]. That rule can be simulated by using [Join], [Meet] and [Prefix], without loss of expressive power. For example, the following sequent, provable using the rule [Intr] is also provable as follows.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{ok} , \mathsf{ok} \vdash}\,[\text{OK}]}{?\lambda_1 , !\lambda_1 \vdash}\,[\textsc{Prefix}]}{(?\lambda_1 \wedge ?\lambda_2) , !\lambda_1 \vdash}\,[\textsc{Meet}] \qquad \dfrac{\dfrac{\dfrac{}{\mathsf{ok} , \mathsf{ok} \vdash}\,[\text{OK}]}{?\lambda_2 , !\lambda_2 \vdash}\,[\textsc{Prefix}]}{(?\lambda_1 \wedge ?\lambda_2) , !\lambda_2 \vdash}\,[\textsc{Meet}]}{(?\lambda_1 \wedge ?\lambda_2) , (!\lambda_1 \vee !\lambda_2) \vdash}\,[\textsc{Join}]$$

However, we cannot simulate all proofs involving the three rules discussed above, if, instead, only [Intr] is employed. The following cannot be proven using only [Intr].

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{ok} , \mathsf{ok} , \mathsf{ok} \vdash}\,[\text{OK}]}{!\lambda_3 , \mathsf{ok} , ?\lambda_3 \vdash}\,[\textsc{Prefix}]}{!\lambda_3 , \mathsf{ok} , ?\lambda_2 \wedge ?\lambda_3 \vdash}\,[\textsc{Meet}]}{!\lambda_1;!\lambda_3 , ?\lambda_1 , ?\lambda_2 \wedge ?\lambda_3 \vdash}\,[\textsc{Prefix}]}{!\lambda_1;!\lambda_3 , ?\lambda_1 \wedge ?\lambda_4 , ?\lambda_2 \wedge ?\lambda_3 \vdash}\,[\textsc{Meet}] \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathsf{ok} , \mathsf{ok} , \mathsf{ok} \vdash}\,[\text{OK}]}{!\lambda_4 , ?\lambda_4 , \mathsf{ok} \vdash}\,[\textsc{Prefix}]}{!\lambda_4 , ?\lambda_1 \wedge ?\lambda_4 , \mathsf{ok} \vdash}\,[\textsc{Meet}]}{!\lambda_2;!\lambda_4 , ?\lambda_1 \wedge ?\lambda_4 , ?\lambda_2 \vdash}\,[\textsc{Prefix}]}{!\lambda_2;!\lambda_4 , ?\lambda_1 \wedge ?\lambda_4 , ?\lambda_2 \wedge ?\lambda_3 \vdash}\,[\textsc{Meet}]}{!\lambda_1;!\lambda_3 \vee !\lambda_2;!\lambda_4 , ?\lambda_1 \wedge ?\lambda_4 , ?\lambda_2 \wedge ?\lambda_3 \vdash}\,[\textsc{Join}]$$

The following is an example of a coinductive proof that, similarly to the above proof, cannot be established using only [Intr]. In the following proof, assume $\mathsf{T} = \mu\mathbf{t}.(!\lambda_1;\mathbf{t} \vee !\lambda_2;\mathbf{t})$, $\mathsf{U} = \mu\mathbf{u}.(?\lambda_1;\mathbf{u})$, and $\mathsf{V} = \mu\mathbf{v}.(?\lambda_2;\mathbf{v})$. We also abbreviate sequents $\Gamma = \mathsf{T} , \mathsf{U} , \mathsf{V}$ and

$\Gamma' = !\lambda_1;\mathsf{T}\ ,\ \mathsf{U}\ ,\ \mathsf{V}$ and $\Gamma'' = !\lambda_2;\mathsf{T}\ ,\ \mathsf{U}\ ,\ \mathsf{V}$, but notice only Γ is used rule [Leaf].

$$\dfrac{\dfrac{\dfrac{\dfrac{}{[\Gamma' \,][\, \Gamma]\, \Gamma \vdash}\,[\textsc{Leaf}]}{[\Gamma' \,][\, \Gamma]\, !\lambda_1;\mathsf{T}\ ,\ ?\lambda_1;\mathsf{U}\ ,\ \mathsf{V} \vdash}\,[\textsc{Prefix}]}{[\Gamma]\, !\lambda_1;\mathsf{T}\ ,\ \mathsf{U}\ ,\ \mathsf{V} \vdash}\,[\textsc{Fix-}\mu] \qquad \dfrac{\dfrac{\dfrac{}{[\Gamma'' \,][\, \Gamma]\, \Gamma \vdash}\,[\textsc{Leaf}]}{[\Gamma'' \,][\, \Gamma]\, !\lambda_2;\mathsf{T}\ ,\ \mathsf{U}\ ,\ ?\lambda_2;\mathsf{V} \vdash}\,[\textsc{Prefix}]}{[\Gamma]\, !\lambda_2;\mathsf{T}\ ,\ \mathsf{U}\ ,\ \mathsf{V} \vdash}\,[\textsc{Fix-}\mu]}{[\Gamma]\, !\lambda_1;\mathsf{T} \vee !\lambda_2;\mathsf{T}\ ,\ \mathsf{U}\ ,\ \mathsf{V} \vdash}\,[\textsc{Join}]$$

$$\dfrac{[\Gamma]\, !\lambda_1;\mathsf{T} \vee !\lambda_2;\mathsf{T}\ ,\ \mathsf{U}\ ,\ \mathsf{V} \vdash}{\mathsf{T}\ ,\ \mathsf{U}\ ,\ \mathsf{V} \vdash}\,[\textsc{Fix-}\mu]$$

$$\dfrac{\mathsf{T}\ ,\ \mathsf{U}\ ,\ \mathsf{V} \vdash}{\mathsf{T}\ ,\ \mathsf{U} \otimes \mathsf{V} \vdash}\,[\textsc{Times}]$$

Notice, the above proof establishes $\mu\mathbf{u}.(?\lambda_1;\mathbf{u}) \otimes \mu\mathbf{v}.(?\lambda_2;\mathbf{v}) \leq \mu\mathbf{t}.(?\lambda_1;\mathbf{t} \wedge ?\lambda_2;\mathbf{t})$ – a subtype judgement decomposing a single threaded participant into two concurrent threads.

Admissibility of [Mix]. The fact that the [Mix] rule is admissible allows scenarios where separate parallel communications can occur. For example, the subtype judgement $!\lambda_1 \otimes ?\lambda_1 \otimes !\lambda_2 \otimes ?\lambda_2 \leq \mathtt{ok}$ (which also holds in pure linear logic **with** mix only), can be established by the following proof in Session **without** using mix.

$$\dfrac{\dfrac{\dfrac{}{\mathtt{ok}\ ,\ \mathtt{ok}\ ,\ \mathtt{ok}\ ,\ \mathtt{ok}\ ,\ \mathtt{ok} \vdash}\,[\text{OK}]}{!\lambda_1\ ,\ ?\lambda_1\ ,\ !\lambda_2\ ,\ ?\lambda_2\ ,\ \mathtt{ok} \vdash}\,[\textsc{Prefix}]\ (\textit{twice})}{!\lambda_1 \otimes ?\lambda_1 \otimes !\lambda_2 \otimes ?\lambda_2\ ,\ \mathtt{ok} \vdash}\,[\textsc{Times}]\ (\textit{twice})$$

The admissibility of [Mix] is a corollary of Theorem 5.

3.3 Typing multiparty compatible networks, by using subtyping

The syntax of processes is defined by the following grammar.

▶ **Definition 7** (Processes). Processes for threads *are defined by:*

$$\mathbb{P} ::= \quad \Sigma_{i\in I} ?\lambda_i;\mathbb{P}_i \quad | \quad \oplus_{i\in I} !\lambda_i;\mathbb{P}_i \quad | \quad \mu X.\mathbb{P} \quad | \quad X \quad | \quad 1$$

Processes for networks *are defined by grammar:* $\mathbb{N} ::= \ \mathbb{P} \ | \ \mathbb{N} \parallel \mathbb{N}$.

We simply refer to both of the above as processes*, ranged over by* P*,* Q*,* R*. . . .*

Internal choice $\oplus$ defines a process ready to perform **any** of the given outputs, and external choice $\sum$ indicates a process ready to perform **some** input. We typically abbreviate $!\lambda;P$ and $!\lambda_1;P_1 \oplus !\lambda_2;P_2$ for the unary and binary versions of the above external choice. Similarly, $?\lambda;P$ and $?\lambda_1;P_1 + ?\lambda_2;P_2$ can be used for internal choices.

$$\dfrac{\Delta \vdash P_i : \mathsf{T}_i\ (i \in I)}{\Delta \vdash \Sigma_{i\in I} ?\lambda_i;P_i : \bigwedge_{i\in I} ?\lambda_i;\mathsf{T}_i}\,[\textsc{T-ExtCh}] \qquad \dfrac{\Delta \vdash P_i : \mathsf{T}_i\ (i \in I)}{\Delta \vdash \oplus_{i\in I} !\lambda_i;P_i : \bigvee_{i\in I} !\lambda_i;\mathsf{T}_i}\,[\textsc{T-IntCh}]$$

$$\Delta, X : \mathbf{t} \vdash X : \mathbf{t}\ \ [\textsc{T-Var}] \qquad \dfrac{\Delta, X : \mathbf{t} \vdash P : \mathsf{T}}{\Delta \vdash \mu X.P : \mu\mathbf{t}.\mathsf{T}}\,[\textsc{T-Rec}]$$

$$\dfrac{\Delta \vdash P : \mathsf{T} \quad \Delta \vdash Q : \mathsf{U}}{\Delta \vdash P \parallel Q : \mathsf{T} \otimes \mathsf{U}}\,[\textsc{T-Par}] \qquad \Delta \vdash 1 : \mathtt{ok}\ \ [\textsc{T-1}] \qquad \dfrac{\Delta \vdash P : \mathsf{T} \qquad \mathsf{T} \leq \mathsf{U}}{\Delta \vdash P : \mathsf{U}}\,[\textsc{subsumption}]$$

Figure 4 Typing rules for processes, making use of the subtype relation $\leq$ in Def. 4.

Multiparty compatible processes are those with type $\mathtt{ok}$. Note, for any interesting example, this will involve applying subsumption.

▶ **Definition 8** (compatibility). *Process P is multiparty compatible whenever $\vdash P\colon \mathtt{OK}$, according to the rules of Fig. 4, where environment Δ associates process variables to type variables.*

Any application of the [SUBSUMPTION] rule can always be delayed to the final step. I.e., we calculate the minimal type for the whole network, then apply [SUBSUMPTION].

▶ **Theorem 9** (algorithmic typing). *If $\vdash P\colon \mathsf{U}$ then we can construct a T such that $\mathsf{T} \leq \mathsf{U}$ holds and $\vdash P\colon \mathsf{T}$ holds without using the [SUBSUMPTION] rule.*

The above result is another consequence of cut elimination.

An immediate consequence is that, if P is multiparty compatible, there exists T such that $\vdash P\colon \mathsf{T}$, without using the *subsumption* rule, and $\mathsf{T} \vdash$ holds. For example, proofs from the previous section can be used to established that networks such as the following are multiparty compatible: $!\lambda_1;!\lambda_3 \oplus !\lambda_2;!\lambda_4 \parallel ?\lambda_1 + ?\lambda_4 \parallel ?\lambda_2 + ?\lambda_3$ and $\mu\mathbf{t}.(!\lambda_1;\mathbf{t} \oplus !\lambda_2;\mathbf{t}) \parallel \mu\mathbf{u}.(?\lambda_1;\mathbf{u}) \parallel \mu\mathbf{v}.(?\lambda_2;\mathbf{v})$. Furthermore, the multiparty compatibility of the processes from Sec. 2 can be established in this way.

Note on open sessions. We select a flexible presentation in Fig. 4, since, as a bonus, we can also use the above type system to reason about open sessions, which may be missing participants in order for multiparty compatibility to hold. For example, by using [SUBSUMPTION] and the processes from Sec. 2, we have the following type judgement.

$$\vdash \mathit{Owner} \parallel \mathit{Untrusted\ App} \parallel \mathit{OAuth\ Server}\colon \mu\mathbf{t}.\big(!\mathit{release} \oplus\ !\texttt{request}(\texttt{token});\ (?\mathit{revoke} + ?\texttt{response}(\texttt{data});\mathbf{t})\ \big)$$

The above type judgement indicates an "interface" exposed by the open session given by network *Owner* ∥ *Untrusted App* ∥ *OAuth Server*. Hence, if composed with a process that interacts with the interface given by the dual of the above type (such as *Resource* from Sec 2) we can judge the whole system to be multiparty compatible. Composition of two open sessions can be performed by using [T-PAR] and then applying [SUBSUMPTION] to the resulting type to show that, together, they inhabit type OK, assuming that together the processes are multiparty compatible (alternatively, when composed, they may expose another interface if the composition of two open sessions is still an open session). Note this achieves the same effect as applying a rule of the following form.

$$\frac{\Delta \vdash P : \mathsf{T} \otimes \mathsf{U} \quad \Delta \vdash Q : \overline{\mathsf{U}} \otimes \mathsf{V}}{\Delta \vdash P \parallel Q : \mathsf{T} \otimes \mathsf{V}}\ [\text{T-CUT}]$$

The above rule, derivable using [T-PAR] and [SUBSUMPTION], achieves the effect of a "connecting cut", as desired in recent work on open multiparty sessions [6].

3.4 Guaranteeing deadlock freedom (via race freedom)

In order to prove deadlock freedom of multiparty compatible networks, we require a reduction system for closed networks, defined by the rules in Fig. 5. As standard [17], different behaviours are forced for internal choice and external choice. When ranging over all executions, for external choice, we consider all branches, as indicated by the transition rule for *internal choice* ($\oplus$). Notice that, in order for a communication to occur, we must have committed to a single branch of the internal choice, forcing all branches to be resolved. However, we need only select one of the inputs with the corresponding output label in an external choice ($\sum$) for a communication to occur.

Race freedom. Some multiparty compatible networks with race conditions are not deadlock free. Races can be avoided by naming participants and ensuring each branch of an external choice awaits a message from the same participant but is labelled differently compared to other branches of that external choice. For example, the following multiparty compatible networks have races, hence should be rejected. For network $!\lambda_1;!\lambda_2 \oplus !\lambda_1;!\lambda_3 \parallel ?\lambda_1;?\lambda_2 + ?\lambda_1;?\lambda_3$, when λ_1 is sent it may be received by the wrong branch of the external choice resulting in deadlock. Similarly, network $!\lambda_1;?\lambda_2 \parallel !\lambda_1 \parallel ?\lambda_1;!\lambda_2;?\lambda_1$, may deadlock if the second process engages in a communication before the first.

While explicitly naming participants, as described above, would avoid such examples, for added flexibility we show that we can also achieve race freedom without naming participants. This additional flexibility is necessary for examples such as in Sec. 2, where one participant is replaced by two or more participants (hence if participants were named we would require a mechanism such as internal delegation [13] to allow one participant act on behalf of another). An added benefit of avoiding races without naming participants is that we may guarantee race freedom without relying on participant names to guide reductions.

Race freedom can be formulated in terms of a type inference problem using the race type system in Fig. 6, where A *race type* is of the form $\langle \mathtt{o}:\alpha, \mathtt{i}:\chi \rangle$, where α and χ are sets of sets of labels. The former, α, represents a set of sets of output labels – one set of labels for each thread in a network. The latter χ represents a set of sets of inputs – one set of labels for each external choice somewhere in the network. We also require a "participant condition" ensuring all branches of a choice talk to the same process, formalised as follows.

▶ **Definition 10.** *A race type $\langle \mathtt{o}:\alpha, \mathtt{i}:\chi \rangle$ satisfies the participant condition whenever, for all $x \in \chi$ and $y, z \in \alpha$, if $x \cap y \neq \emptyset$ and $x \cap z \neq \emptyset$ then $y = z$. A process P is race free, whenever there exists a race type $\langle \mathtt{o}:\alpha, \mathtt{i}:\chi \rangle$ satisfying the participant condition such that $\vdash P : \langle \mathtt{o}:\alpha, \mathtt{i}:\chi \rangle$ using the rules of Fig. 6.*

The above *race-freedom* property we propose is satisfied whenever the unfolding of all fixed points of a process satisfies the following:

- Branches of an external choice use distinct labels for their **immediately enabled** inputs (see [R-ExtCh]).
- For any external choice, the set of immediately enabled input labels in an external choice must be disjoint from the set of all output labels of **all but one** of the parallel components (the *participant condition*). This ensures one participant is listening to at most one other participant at a time.
- For parallel processes, $P \parallel Q$, the set of **all** input labels of P and the set of **all** input labels of Q are disjoint; and, similarly, the sets of all output labels of P and Q are disjoint (see [R-Par]).

The above property is efficient to check, since it simply builds up the relevant sets of sets of labels. Note, a single thread always has a singleton set of outputs.

$$\frac{j \in I}{\oplus_{i\in I} !\lambda_i;Q_i \longrightarrow !\lambda_j;Q_j} \qquad \frac{j \in I}{!\lambda_j;P \parallel \Sigma_{i\in I} ?\lambda_i;Q_i \longrightarrow P \parallel Q_j} \qquad \frac{}{\mathtt{rec}X.P \longrightarrow P\{^{\mathtt{rec}X.P}/_X\}}$$

$$\frac{P \longrightarrow P'}{P \parallel Q \longrightarrow P' \parallel Q} \qquad \frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q} \qquad \begin{array}{l} P \parallel (Q \parallel R) \equiv (P \parallel Q) \parallel R \\ P \parallel Q \equiv Q \parallel P \quad P \parallel 1 \equiv P \end{array}$$

Figure 5 Reduction system for networks.

$$\frac{\Sigma \vdash P_i : \langle \mathtt{o}{:}\{x_i\}, \mathtt{i}{:}\chi_i \rangle \ \ (i \in I) \quad (\forall i,j \in I)\lambda_i = \lambda_j \text{ implies } i = j}{\Sigma \vdash \Sigma_{i\in I} ?\lambda_i ; P_i : \left\langle \mathtt{o}{:}\left\{\bigcup_{i\in I} x_i\right\}, \mathtt{i}{:}\bigcup_{i\in I}\chi_i \cup \{\{\lambda_i : i \in I\}\} \right\rangle} \text{[R-ExtCh]}$$

$$\frac{\Sigma \vdash P_i : \langle \mathtt{o}{:}\{x_i\}, \mathtt{i}{:}\chi_i \rangle \ \ (i \in I)}{\Sigma \vdash \oplus_{i\in I} !\lambda_i ; P_i : \left\langle \mathtt{o}{:}\left\{\bigcup_{i\in I} x_i \cup \{\lambda_i : i \in I\}\right\}, \mathtt{i}{:}\bigcup_{i\in I}\chi_i \right\rangle} \text{[R-IntCh]}$$

$$\frac{\Sigma \vdash P : \langle \mathtt{o}{:}\alpha, \mathtt{i}{:}\chi \rangle \quad \Sigma \vdash Q : \langle \mathtt{o}{:}\beta, \mathtt{i}{:}\zeta \rangle \quad \left(\bigcup \alpha\right) \cap \left(\bigcup \beta\right) = \emptyset \quad \left(\bigcup \chi\right) \cap \left(\bigcup \zeta\right) = \emptyset}{\Sigma \vdash P \parallel Q : \langle \mathtt{o}{:}\alpha \cup \beta, \mathtt{i}{:}\chi \cup \zeta \rangle} \text{[R-Par]}$$

$$\Sigma, X : \langle \mathtt{o}{:}\alpha, \mathtt{i}{:}\chi \rangle \vdash X : \langle \mathtt{o}{:}\alpha, \mathtt{i}{:}\chi \rangle \ \ \text{[R-Var]} \qquad \Sigma \vdash 1 : \langle \mathtt{o}{:}\emptyset, \mathtt{i}{:}\emptyset \rangle \ \ \text{[R-1]}$$

$$\frac{\Sigma, X : \langle \mathtt{o}{:}\alpha, \mathtt{i}{:}\chi \rangle \vdash P : \langle \mathtt{o}{:}\alpha, \mathtt{i}{:}\chi \rangle}{\Sigma \vdash \mu X.P : \langle \mathtt{o}{:}\alpha, \mathtt{i}{:}\chi \rangle} \text{[R-Rec]}$$

Figure 6 Type rules for checking race freedom.

A critical example the participant condition rejects is the following process, which is of the race type indicated below.

$$\vdash !\lambda_1 \parallel \mathtt{rec}X.\,(?\lambda_1 + ?\lambda_2 ; X) \parallel \mathtt{rec}Y.!\lambda_2 ; Y : \ \langle \mathtt{o}{:}\{\{\lambda_1\}, \emptyset, \{\lambda_2\}\}, \mathtt{i}{:}\{\{\lambda_1, \lambda_2\}\} \rangle$$

The above example processes contains a race. Two parallel outputs with different labels contact a process ready to receive a message from either process and, if actions labelled λ_1 are played, the process deadlocks. The above example is forbidden by the participant condition since we have $\{\lambda_1, \lambda_2\} \cap \{\lambda_1\} \neq \emptyset$ and $\{\lambda_1, \lambda_2\} \cap \{\lambda_2\} \neq \emptyset$ but $\{\lambda_1\} \neq \{\lambda_2\}$.

Deadlock freedom. Deadlock freedom can be defined as follows (coinductively): at any point we can either make progress or we have successfully terminated.

▶ **Definition 11** (deadlock freedom). *A network P is deadlock free whenever:*

- *either $P \equiv 1$, or there exists network Q such that $P \longrightarrow Q$;*
- *and, for all R such that $P \longrightarrow R$ we have R is deadlock free.*

The theory developed in this work guarantees deadlock freedom as in Def. 11.

▶ **Theorem 12.** *Any race-free multiparty-compatible network satisfies deadlock freedom.*

The proof of this result [see Appendix] relies on Theorem 5 and builds on novel proof normalisation techniques developed for giving computational interpretations of formulae in extensions of linear logic [35, 36].

▶ Remark 13. Note often deadlock freedom is referred to as "progress" which is an overloaded word in the literature. Deadlock freedom does not necessarily prevent starvation, as for notions such as lock freedom [37, 46]. Restricted variants of Session can be tightened to enforce stronger liveness properties – an observation deserving of attention in future work.

Soundness of the subtype system with respect to our multithreaded liberalisation of the substitution principle [25] is precisely formulated below, which is an immediate consequence of Theorem 5 and Theorem 12. Notice the flexible subtype system in this work, which permits networks consisting of parallel threads to be compared, allows a thread to be substituted by more than one thread, as motivated in Sec. 2.

▶ **Corollary 14** (substitution principle). *Assume P, Q and R are closed networks. If $\vdash P\colon \mathsf{T}$, $\vdash Q\colon \mathsf{T}'$ and $\mathsf{T} \leq \mathsf{T}'$, then if $\vdash Q \parallel R\colon \mathtt{ok}$, and $P \parallel R$ is race free, then $P \parallel R$ is deadlock free.*

Proof. Assume $\vdash P\colon \mathsf{T}$ and $\vdash Q\colon \mathsf{T}'$ without using [SUBSUMPTION], and also assume $\mathsf{T} \leq \mathsf{T}'$, $\vdash Q \parallel R\colon \mathtt{ok}$ and $P \parallel R$ is race free. By Theorem 9, there exists a type U such that $\vdash Q \parallel R\colon \mathsf{U}$ without using [SUBSUMPTION] and $\mathsf{U} \leq \mathtt{ok}$. By Lemma 15 there exists V such that $\mathsf{U} = \mathsf{T}' \otimes \mathsf{V}$ and $\vdash P \parallel R\colon \mathsf{T}' \otimes \mathsf{V}$ without using [SUBSUMPTION]. By Theorem 5, we have $\mathsf{T} \otimes \mathsf{V} \leq \mathsf{T}' \otimes \mathsf{V}$, and hence, by Theorem 5 again, $\mathsf{T} \otimes \mathsf{V} \leq \mathtt{ok}$. Thereby $\vdash P \parallel R\colon \mathtt{ok}$, and hence, by race freedom and Theorem 12, we have $P \parallel R$ is deadlock free, as required. ◀

Importance of avoiding races. The following example emphasises the importance of checking races are avoided. Consider the multiparty compatible network $1 \parallel !\lambda_1 \parallel (?\lambda_1 + ?\lambda_2)$. Observe we have $\vdash !\lambda_2 \parallel ?\lambda_2\colon \mathtt{ok}$ hence process 1 can be substituted by $!\lambda_2 \parallel ?\lambda_2$ while preserving multiparty compatibility. Now, if we remove the condition concerning races in the substitution principle, after applying the above substitution in the network at the top of the paragraph, we should have $!\lambda_2 \parallel ?\lambda_2 \parallel !\lambda_1 \parallel (?\lambda_1 + ?\lambda_2)$ is deadlock free. However, this network is in fact not deadlock free, due to the presence of a race.

Note our race-freedom property does not require output labels in an internal choice to be distinct. For example, the network $(!\lambda_1;!\lambda_2 \oplus !\lambda_1;!\lambda_3) \parallel ?\lambda_1;(?\lambda_2 + ?\lambda_3)$ is race free, multiparty compatible and deadlock free. Note this is example would not be typeable using established session type systems.

3.5 Typeable sessions for which there is no global type

Multiparty compatibility is defined independently from global types. Theories that rely on global types run into the problem that many reasonable protocols have no global type. Such problematic protocols typically feature branching under a recursion where different participants are contacted in each branch. The problem of typing protocols for which there is no established theory in which they can be assigned a global type has been explored in recent work [48].

To emphasise that Session can also be used to type multiparty sessions for which there is no global type, we adapt one of the key examples from related work (Figure 4, (2) [48]). In this recursive two-buyer protocol a buyer repeatedly asks another buyer to split the price. Assume we have the following types.

- $\mathsf{T}_\mathtt{A} = !\mathtt{query};?\mathtt{price};\mu\mathbf{t}.\mathsf{T}_1$ where $\mathsf{T}_1 = (!\mathtt{split};\mathsf{T}_2 \vee !\mathtt{cancel};!\mathtt{no})$ and $\mathsf{T}_2 = (?\mathtt{yes};!\mathtt{buy} \wedge ?\mathtt{no};\mathbf{t})$
- $\mathsf{T}_\mathtt{B} = \mu\mathbf{t}.\mathsf{T}_3$ where $\mathsf{T}_3 = (?\mathtt{split};\mathsf{T}_4 \wedge ?\mathtt{cancel})$ and $\mathsf{T}_4 = !\mathtt{yes} \vee !\mathtt{no};\mathbf{t}$
- $\mathsf{T}_\mathtt{S} = ?\mathtt{query};!\mathtt{price};\mathsf{T}_5$ where $\mathsf{T}_5 = ?\mathtt{buy} \wedge ?\mathtt{no}$.

Also assume we have sequents $\Gamma = \mu\mathbf{t}.\mathsf{T}_1$, T_5 , $\mathsf{T}_\mathtt{B}$ and $\Gamma' = \mathsf{T}_1\{^{\mu\mathbf{t}.\mathsf{T}_1}/_{\mathbf{t}}\}$, T_5 , $\mathsf{T}_\mathtt{B}$ (only the former is used in a [LEAF] axiom). The following proof can be used to establish $\mathsf{T}_\mathtt{A} \otimes \mathsf{T}_\mathtt{B} \otimes \mathsf{T}_\mathtt{S} \leq \mathtt{ok}$, which can be used in a multiparty compatibility judgement. Notice we use the admissible compound rule [INTR] to shorten the proof.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\left[\Gamma' \rangle\!\rangle\, \Gamma\right] \mathtt{ok}\,,\, \mathtt{ok}\,,\, \mathtt{ok} \vdash}\,[\text{OK}]}{\left[\Gamma' \rangle\!\rangle\, \Gamma\right] \mathtt{!buy}\,,\, \mathsf{T}_5\,,\, \mathtt{ok} \vdash}\,[\text{Intr}] \quad \dfrac{}{\left[\Gamma' \rangle\!\rangle\, \Gamma\right] \mu\mathbf{t}.\mathsf{T}_1\,,\, \mathsf{T}_5\,,\, \mathsf{T}_\mathtt{B} \vdash}\,[\text{Leaf}]}{\left[\Gamma' \rangle\!\rangle\, \Gamma\right] \mathsf{T}_2\left\{{}^{\mu\mathbf{t}.\mathsf{T}_1}/_{\mathbf{t}}\right\}\,,\, \mathsf{T}_5\,,\, \mathsf{T}_4\left\{{}^{\mu\mathbf{t}.\mathsf{T}_3}/_{\mathbf{t}}\right\} \vdash}\,[\text{Intr}] \quad \dfrac{\dfrac{}{\left[\Gamma' \rangle\!\rangle\, \Gamma\right] \mathtt{ok}\,,\, \mathtt{ok}\,,\, \mathtt{ok} \vdash}\,[\text{OK}]}{\left[\Gamma' \rangle\!\rangle\, \Gamma\right] \mathtt{!no}\,,\, \mathsf{T}_5\,,\, \mathtt{ok} \vdash}\,[\text{Intr}]}{\left[\Gamma' \rangle\!\rangle\, \Gamma\right] \mathsf{T}_1\left\{{}^{\mu\mathbf{t}.\mathsf{T}_1}/_{\mathbf{t}}\right\}\,,\, \mathsf{T}_5\,,\, \mathsf{T}_3\left\{{}^{\mu\mathbf{t}.\mathsf{T}_3}/_{\mathbf{t}}\right\} \vdash}\,[\text{Intr}]$$
$$\dfrac{\left[\Gamma\right] \mathsf{T}_1\left\{{}^{\mu\mathbf{t}.\mathsf{T}_1}/_{\mathbf{t}}\right\}\,,\, \mathsf{T}_5\,,\, \mathsf{T}_\mathtt{B} \vdash}{\mu\mathbf{t}.\mathsf{T}_1\,,\, \mathsf{T}_5\,,\, \mathsf{T}_\mathtt{B} \vdash}\,[\text{Fix-}\mu]$$
$$\dfrac{\mathtt{?price};\mu\mathbf{t}.\mathsf{T}_1\,,\, \mathtt{!price};\mathsf{T}_5\,,\, \mathsf{T}_\mathtt{B} \vdash}{\mathsf{T}_\mathtt{A}\,,\, \mathsf{T}_\mathtt{S}\,,\, \mathsf{T}_\mathtt{B} \vdash}\,[\text{Prefix}]$$

In the above example, it is possible that processes typed with $\mathsf{T}_\mathtt{A}$ and $\mathsf{T}_\mathtt{B}$ negotiate forever and a process typed with $\mathsf{T}_\mathtt{S}$, after reaching a state typed by T_5, waits forever. Such starvation is permitted by our classic notion of progress in Def. 11, i.e., deadlock freedom.

4 Related Work and Future Work

A closely related line of work studies the problem of synthesising a "coherent" global type for multi-party compatible types [43]. The approach in the current paper can be used to expose the structural proof theoretic content of a closely related system proposed for such a synthesis problem [38]. There is much work providing notions of semantic subtyping for session types [7, 5, 45], whose resulting systems can be interpreted proof theoretically using subsystems and variants of Session (at least for the first-order fragment without delegation).

It could be valuable to explore connections between Session, which follows a processes-as-formulas approach, and a variety of Curry-Howard inspired systems. There are intersection type systems, satisfying subject expansion, that completely characterise deadlock freedom for a fragment of the asynchronous π-calculus where a name can only be used as an input channel by the process that created the name [16]. Process in that work are quite different from those in our session calculus, since, in this work, we neither consider channel passing (delegation) nor asynchrony, while they do not consider choice. Challenges concerning duality of binary sessions in the presence of delegation and recursion are explored through a linear λ-calculus typed using explicit least and greatest fixedpoints rather than equi-recursion [40]. Regarding circular proofs, Derakhshan and Pfenning propose a calculus for binary sessions with delegation in a Curry-Howard style [22]. In their work, they propose a locally checkable condition that guarantees a well-typed session will always terminate either in an empty configuration or a configuration attempting to communicate along external channels.

In future work, it would be valuable to investigate variants of the rules, notably a focussed variant of Session [3, 4]. In a focussed system, rules such as Join are treated *asynchronously*, meaning that we can immediately apply the rule without backtracking; whereas rules such as Meet are *synchronous*, meaning that, in general, backtracking may be required during proof search. The important observation is that, for race-free sessions there will only be one way to apply synchronous rules, thereby eliminating the need to backtrack in the search for a proof, i.e., proof search can be conducted deterministically. The ability to search for proofs in this uniform manner is connected with goal-directed search in logic programming [42].

The system designed in this work preserves deadlock freedom for race-free processes, as established in Theorem 12; but does not guarantee stronger livelock freedom properties (sometimes referred to as lock freedom) [37, 46, 49]. Livelock freedom strengthens deadlock freedom by ensuring that no parties are starved of resources; however, there are many subtle variations on precisely how livelock freedom is defined. Hence we push the investigation of refinements of Session that can guarantee notions livelock freedom to future work.

To illustrate the above point, we observe some more unexpected properties of Session. Observe, the process $\mu X.?\lambda_1;X \parallel ?\lambda_2 \parallel \mu Y.!\lambda_1;Y$ is race-free and multiparty compatible, and hence deadlock free. However, it has a hanging input $?\lambda_2$ that never receives a message, hence it is not livelock free in any sense. Using a proof of the multiparty compatibility of the above process, we can also establish subtype judgement $\mu\mathbf{t}.?\lambda_1;\mathbf{t} \otimes ?\lambda_2 \leq \mu\mathbf{t}.?\lambda_1;\mathbf{t}$. This subtype judgements allows inactive parallel components to be typed using the subtype system, as long as they rest of the system is deadlock free. Thus the current formulation of Session guarantees no property stronger than deadlock freedom.

For a more subtle example outside the scope of established session type systems, consider the types $\mathsf{T} = \mu\mathbf{t}.(!\lambda_1;\mathbf{t} \vee !\lambda_2;!\lambda_3)$ and $\mathsf{U} = \mu\mathbf{t}.(?\lambda_1;\mathbf{t} \wedge ?\lambda_2)$. We have $\mathsf{T} \otimes \mathsf{U} \leq !\lambda_3$ thus a thread that sends λ_2 can be replaced by two threads that may choose to talk internally on λ_1 forever, although there is always the possibility of a branching taken where λ_3 is sent. This subtype judgement does preserve some notions of livelock freedom (it is always possible for everyone to eventually act [46]), but not stronger notions of livelock freedom (always everyone must act eventually [37]). An objective for future work would be to explain how Session can be refined by restricting circular proofs so that they preserve a strong form of livelock freedom. The key idea is to check that at all threads in a network act at least once in every unfolding of a recursion, thereby rejecting both subtype judgements above.

5 Conclusion

The proof calculus Session, introduced in Fig. 3, showcases tools of structural proof theory, i.e., analytic calculi satisfying cut elimination (Theorem 5), which can be used in the design of rich multiparty session type systems. Session defines an algorithmic subtype system (Definition 4), the transitivity of which follows from cut elimination (Corollary 6). The subtype system admits a more flexible substitution principle (Corollary 14) than standard. This flexibility enables subtyping to be used directly to decide multiparty compatibility (Definition 8) and also opens up fresh problems that can be tackled using subtyping, not limited to scenarios where extra parallelism is introduced, as illustrated in Sec. 2.

Race freedom may be guaranteed by naming participants; however, for extra flexibility we propose a type system for race freedom (Definition 10), which does not require participants to be named. From these definitions, we establish our main result (Theorem 12) guaranteeing deadlock freedom for networks that are both multiparty compatible and race free. In this line of work, global types are optional, allowing networks for which no global type exists to be typed.

References

1 Samson Abramsky. Computational interpretations of linear logic. *Theoretical computer science*, 111(1):3–57, 1993. doi:10.1016/0304-3975(93)90181-R.

2 Roberto M. Amadio and Luca Cardelli. Subtyping recursive types. *ACM Trans. Program. Lang. Syst.*, 15(4):575–631, September 1993. doi:10.1145/155183.155231.

3 Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992. doi:10.1093/logcom/2.3.297.

4 David Baelde, Amina Doumane, and Alexis Saurin. Infinitary Proof Theory: the Multiplicative Additive Case. In Jean-Marc Talbot and Laurent Regnier, editors, *25th EACSL Annual Conference on Computer Science Logic (CSL 2016)*, volume 62 of *LIPIcs*, pages 42:1–42:17. Schloss Dagstuhl, 2016. doi:10.4230/LIPIcs.CSL.2016.42.

5 Franco Barbanera and Ugo de'Liguoro. Sub-behaviour relations for session-based client/server systems. *Mathematical Structures in Computer Science*, 25(6):1339–1381, 2015. doi:10.1017/S096012951400005X.

6 Franco Barbanera and Mariangiola Dezani-Ciancaglini. Open multiparty sessions. In *Proceedings 12th Interaction and Concurrency Experience, ICE 2019, Copenhagen, Denmark, 20-21 June 2019.*, pages 77–96, 2019. doi:10.4204/EPTCS.304.6.

7 Giovanni Bernardi and Matthew Hennessy. Using higher-order contracts to model session types. *Logical Methods in Computer Science*, 12(2), 2016. doi:10.2168/LMCS-12(2:10)2016.

8 Eduardo Bonelli and Adriana Compagnoni. Multipoint session types for a distributed calculus. In Gilles Barthe and Cédric Fournet, editors, *Trustworthy Global Computing*, pages 240–256. Springer, 2008. doi:10.1007/978-3-540-78663-4_17.

9 James Brotherston and Alex Simpson. Sequent calculi for induction and infinite descent. *Journal of Logic and Computation*, 21(6):1177–1216, October 2010. doi:10.1093/logcom/exq052.

10 Paola Bruscoli. A purely logical account of sequentiality in proof search. In Peter J. Stuckey, editor, *Logic Programming*, pages 302–316. Springer, 2002. doi:10.1007/3-540-45619-8_21.

11 Luís Caires and Jorge A. Pérez. Multiparty session types within a canonical binary theory, and beyond. In *FORTE 2016*, pages 74–95, 2016. doi:10.1007/978-3-319-39570-8_6.

12 Marco Carbone, Fabrizio Montesi, Carsten Schürmann, and Nobuko Yoshida. Multiparty session types as coherence proofs. *Acta Informatica*, 54(3):243–269, 2017. doi:10.1007/s00236-016-0285-y.

13 Ilaria Castellani, Mariangiola Dezani-Ciancaglini, Paola Giannini, and Ross Horne. Global types with internal delegation. *Theoretical Computer Science*, 807:128–153, 2020. doi:10.1016/j.tcs.2019.09.027.

14 Tzu-chun Chen, Mariangiola Dezani-Ciancaglini, Alceste Scalas, and Nobuko Yoshida. On the Preciseness of Subtyping in Session Types. *Logical Methods in Computer Science*, Volume 13, Issue 2, 2017. doi:10.23638/LMCS-13(2:12)2017.

15 Gabriel Ciobanu and Ross Horne. Behavioural analysis of sessions using the calculus of structures. In *Perspectives of System Informatics - 10th International Andrei Ershov Informatics Conference, PSI 2015*, pages 91–106, 2015. doi:10.1007/978-3-319-41579-6_8.

16 Ugo Dal Lago, Marc de Visme, Damiano Mazza, and Akira Yoshimizu. Intersection types and runtime errors in the pi-calculus. *Proc. ACM Program. Lang.*, 3(POPL), 2019. doi:10.1145/3290320.

17 Rocco De Nicola and Matthew Hennessy. CCS without τ's. In Hartmut Ehrig, Robert Kowalski, Giorgio Levi, and Ugo Montanari, editors, *TAPSOFT '87*, pages 138–152. Springer, 1987. doi:10.1007/3-540-17660-8_53.

18 Romain Demangeon and Kohei Honda. Full abstraction in a subtyped pi-calculus with linear types. In *CONCUR*, volume 6901 of *LNCS*, pages 280–296. Springer, 2011. doi:10.1007/978-3-642-23217-6_19.

19 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In Helmut Seidl, editor, *Programming Languages and Systems*, pages 194–213. Springer, 2012. doi:10.1007/978-3-642-28869-2_10.

20 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *Automata, Languages, and Programming*, pages 174–186. Springer, 2013. doi:10.1007/978-3-642-39212-2_18.

21 Pierre-Malo Deniélou, Nobuko Yoshida, Andi Bejleri, and Raymond Hu. Parameterised multiparty session types. *Logical Methods in Computer Science*, 8(4), 2012. doi:10.2168/LMCS-8(4:6)2012.

22 Farzaneh Derakhshan and Frank Pfenning. Circular proof as session-typed processes: a local validity condition. *CoRR*, 2019. arXiv:1908.01909.

23 Simon Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Informatica*, 42(2-3):191–225, 2005. doi:10.1007/s00236-005-0177-z.

24 Simon J. Gay and Malcolm Hole. Types and subtypes for client-server interactions. In *ESOP*, volume 1576 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 1999. doi:10.1007/3-540-49099-X_6.

25 Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, Alceste Scalas, and Nobuko Yoshida. Precise subtyping for synchronous multiparty sessions. *J. Log. Algebr. Meth. Program.*, 104:127–173, 2019. doi:10.1016/j.jlamp.2018.12.002.

26 Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–112, 1987. doi:10.1016/0304-3975(87)90045-4.

27 Jean-Yves Girard and Yves Lafont. Linear logic and lazy computation. In Hartmut Ehrig, Robert Kowalski, Giorgio Levi, and Ugo Montanari, editors, *TAPSOFT '87*, pages 52–66. Springer, 1987. doi:10.1007/BFb0014972.

28 Alessio Guglielmi. A system of interaction and structure. *ACM Transactions on Compututational Logic*, 8, 2007. doi:10.1145/1182613.1182614.

29 Dick Hardt. The OAuth 2.0 authorization framework. standard rfc6749, Internet Engineering Task Force, 2012. URL: https://tools.ietf.org/html/rfc6749.

30 Kohei Honda. Types for dyadic interaction. In *CONCUR'93*, pages 509–523. Springer, 1993. doi:10.1007/3-540-57208-2_35.

31 Kohei Honda, Aybek Mukhamedov, Gary Brown, Tzu-Chun Chen, and Nobuko Yoshida. Scribbling interactions with a formal foundation. In Raja Natarajan and Adegboyega Ojo, editors, *Distributed Computing and Internet Technology*, pages 55–75. Springer, 2011. doi:10.1007/978-3-642-19056-8_4.

32 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '08, pages 273–284. ACM, 2008. doi:10.1145/1328438.1328472.

33 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *J. ACM*, 63(1):9:1–9:67, 2016. doi:10.1145/2827695.

34 Ross Horne. The consistency and complexity of multiplicative additive system virtual. *Scientific Annals of Computer Science*, 25(2):245–316, 2015. doi:10.7561/SACS.2015.2.245.

35 Ross Horne. The sub-additives: A proof theory for probabilistic choice extending linear logic. In *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019*, pages 23:1–23:16, 2019. doi:10.4230/LIPIcs.FSCD.2019.23.

36 Ross Horne and Alwen Tiu. Constructing weak simulations from linear implications for processes with private names. *Mathematical Structures in Computer Science*, 29(8):1275–1308, 2019. doi:10.1017/S0960129518000452.

37 Naoki Kobayashi. A type system for lock-free processes. *Information and Computation*, 177(2):122–159, 2002. doi:10.1016/S0890-5401(02)93171-8.

38 Julien Lange and Emilio Tuosto. Synthesising choreographies from local session types. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR 2012 – Concurrency Theory*, pages 225–239. Springer, 2012.

39 Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From communicating machines to graphical choreographies. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 221–232. ACM, 2015. doi:10.1145/2676726.2676964.

40 Sam Lindley and J. Garrett Morris. Talking bananas: Structural recursion for session types. In *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, ICFP 2016, page 434–447. ACM, 2016. doi:10.1145/2951913.2951921.

41 Barbara Liskov and Jeannette M. Wing. A behavioral notion of subtyping. *ACM Trans. Program. Lang. Syst.*, 16(6):1811–1841, 1994. doi:10.1145/197320.197383.

42 Dale Miller, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov. Uniform proofs as a foundation for logic programming. *Annals of Pure and Applied Logic*, 51(1):125–157, 1991. doi:10.1016/0168-0072(91)90068-W.

43 Dimitris Mostrous, Nobuko Yoshida, and Kohei Honda. Global principal typing in partially commutative asynchronous sessions. In *Programming Languages and Systems*, pages 316–332. Springer, 2009. doi:10.1007/978-3-642-00590-9_23.

44 Luca Padovani. Session types = intersection types + union types. In *Proceedings Fifth Workshop on Intersection Types and Related Systems, ITRS 2010, Edinburgh, U.K., 9th July 2010.*, pages 71–89, 2010. doi:10.4204/EPTCS.45.6.

45 Luca Padovani. On projecting processes into session types. *Mathematical Structures in Computer Science*, 22(2):237–289, 2012. doi:10.1017/S0960129511000405.

46 Luca Padovani. Deadlock and lock freedom in the linear pi-calculus. In *CSL-LICS*, pages 72:1–72:10. ACM Press, 2014. doi:10.1145/2603088.2603116.

47 Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996. doi:10.1017/S096012950007002X.

48 Alceste Scalas and Nobuko Yoshida. Less is more: multiparty session types revisited. *PACMPL*, 3(POPL):30:1–30:29, 2019. doi:10.1145/3290343.

49 Paula Severi and Mariangiola Dezani-Ciancaglini. Observational equivalence for multiparty sessions. *Fundam. Inform.*, 170(1-3):267–305, 2019. doi:10.3233/FI-2019-1863.

50 Alwen Tiu. A system of interaction and structure II: The need for deep inference. *Logical Methods in Computer Science*, 2(2), 2006. doi:10.2168/LMCS-2(2:4)2006.

A Proof of Theorem 12: well-typed networks are deadlock free

We require the following standard lemmas, which follow by structural induction.

▶ **Lemma 15** (inversion lemma). *In the following, we do not use the subsumption rule.*

- *If* $\vdash P \parallel Q \colon \mathsf{T}$, *there exists* U *and* V *such that* $\mathsf{T} = \mathsf{U} \otimes \mathsf{V}$ *and* $\vdash P \colon \mathsf{U}$ *and* $\vdash Q \colon \mathsf{V}$.
- *If* $\vdash \oplus_{i\in I} !\lambda_i ; P_i \colon \mathsf{T}$, *there exists* T_i *such that* $\mathsf{T} = \bigvee_{i\in I} !\lambda_i ; T_i$ *and* $\vdash P_i \colon \mathsf{T}_i$.
- *If* $\vdash \Sigma_{i\in I} ?\lambda_i ; P_i \colon \mathsf{T}$, *there exists* T_i *such that* $\mathsf{T} = \bigwedge_{i\in I} !\lambda_i ; T_i$ *and* $\vdash P_i \colon \mathsf{T}_i$.
- *If* $\vdash \mathtt{rec} X.P \colon \mathsf{T}$, *there exists* U *and* $\boldsymbol{t}$ *such that* $\mathsf{T} = \mu \boldsymbol{t}.\mathsf{U}$ *and* $X \colon \boldsymbol{t} \vdash P \colon \mathsf{U}$.
- *If* $\vdash 1 \colon \mathsf{T}$ *then* $\mathsf{T} = \mathtt{OK}$.

▶ **Lemma 16.** *If* $\vdash \mathtt{rec} X.P \colon \mu \boldsymbol{t}.\mathsf{T}$ *then* $\vdash P\{^{X.P}/_X\} \colon \mathsf{T}\{^{\mu t.\mathsf{T}}/_t\}$.

We also require that race freedom is preserved by the reduction system. This is effectively a subject reduction theorem for the race free property.

▶ **Lemma 17** (race freedom). *If* P *is race free and* $P \longrightarrow Q$, *then* Q *is race free.*

The following condition follows from inverting the type system for race freedom.

▶ **Lemma 18.** *If* $P \parallel Q$ *is race free and* $\vdash P \colon \mathsf{T}$ *and* $\vdash Q \colon \mathsf{U}$, *then if* π *appears in* T, *then* π *does not appear in* U.

Since we employ a reduction semantics, we require that the rules of the structural congruence preserve multiparty compatibility.

▶ **Lemma 19.** *If* $\vdash P \colon \mathtt{OK}$ *and* $P \equiv Q$, *then* $\vdash Q \colon \mathtt{OK}$.

We also require a subject reduction result, where proofs that $\mathsf{T} \leq \mathtt{OK}$ and race freedom play the role that a global type normally plays in such proofs. Note we avoid the term session fidelity since fidelity is typically expressed in terms of global types [32].

▶ **Lemma 20** (subject reduction). *If* $\vdash P \colon \mathtt{OK}$, *and* P *is race free, then for all* Q *such that* $P \longrightarrow Q$, *we have* $\vdash Q \colon \mathtt{OK}$.

Proof. If there exists a reduction, we can apply the structural congruence to a process to reach one of the following forms. By Lemma 19, the use of the structural congruence preserves multiparty compatibility.

Case of internal choice. Assume we have $\vdash \oplus_{i\in I}!\lambda_i;P_i \parallel Q\colon \mathtt{ok}$. By Theorem 9, for some T, we have $\vdash \oplus_{i\in I}!\lambda_i;P_i \parallel Q\colon \mathsf{T}$, without using subsumption, and $\mathsf{T} \leq \mathtt{ok}$. Consider the transition $\oplus_{i\in I}!\lambda_i;P_i \parallel Q \longrightarrow !\lambda_k;P_k \parallel Q$, where $k \in I$.

By Lemma 15, we have there exists U_i and V such that $\mathsf{T} = \bigvee_{i\in I}!\lambda_i;\mathsf{U}_i \otimes \mathsf{V}$ and $\vdash P_i\colon \mathsf{U}_i$, for all i, and $\vdash Q\colon \mathsf{V}$. Therefore $\vdash !\lambda_k;P_k \parallel Q\colon !\lambda_i;\mathsf{U}_i \otimes \mathsf{V}$.

Now, since $\bigvee_{i\in I}!\lambda_i;\mathsf{U}_i \ , \mathsf{V} \vdash$ is provable and so is $\bigwedge_{i\in I}?\lambda_i;\overline{\mathsf{U}_i} \ , !\lambda_k;\mathsf{U}_k \vdash$, by Theorem 5, $!\lambda_k;\mathsf{U}_k \ , \mathsf{V} \vdash$ holds. Hence $!\lambda_k;\mathsf{U}_k \otimes \mathsf{V} \leq \mathtt{ok}$, as required.

Case of external choice. Assume we have $\vdash \Sigma_{i\in I}?\lambda_i;P_i \parallel !\lambda_k;Q \parallel R\colon \mathtt{ok}$, where $k \in I$ and $\Sigma_{i\in I}?\lambda_i;P_i \parallel !\lambda_k;Q \parallel R$ is race free. Consider transition $\Sigma_{i\in I}?\lambda_i;P_i \parallel !\lambda_k;Q \parallel R \longrightarrow P_k \parallel Q \parallel R$.

By Theorem 9, for some T, we have that $\vdash \Sigma_{i\in I}?\lambda_i;P_i \parallel !\lambda_k;Q \parallel R\colon \mathsf{T}$ holds without using subsumption, and $\mathsf{T} \leq \mathtt{ok}$. By Lemma 15 we have there exists U_i, V and W such that we have $\mathsf{T} = \bigwedge_{i\in I}?\lambda_i;\mathsf{U}_i \otimes !\lambda_k;\mathsf{V} \otimes \mathsf{W}$ and $\vdash P_i\colon \mathsf{U}_i$, for all i, and $\vdash Q\colon \mathsf{V}$ and $\vdash R\colon \mathsf{W}$. Therefore we have $\vdash P_k \parallel Q \parallel R\colon \mathsf{U}_k \otimes \mathsf{V} \otimes \mathsf{W}$ holds.

Now, consider the proof of $\bigwedge_{i\in I}?\lambda_i;\mathsf{U}_i \ , !\lambda_k;\mathsf{V} \ , \mathsf{W} \vdash$. Since we have the type judgements $\vdash \Sigma_{i\in I}?\lambda_i;P_i \parallel !\lambda_k;Q\colon \bigwedge_{i\in I}?\lambda_i;\mathsf{U}_i \otimes !\lambda_k;\mathsf{V}$ and $\vdash R\colon \mathsf{W}$ and $\Sigma_{i\in I}?\lambda_i;P_i \parallel !\lambda_k;Q \parallel R$ is race free, by Lemma 18, neither $!\lambda_i$ nor $?\lambda_k$ appear in W. Hence there are only two possibilities, for **every branch of the proof tree**:

1. Either we eventually reach an application of rule [Prefix], possibly via an application of [Meet] as follows:

$$\dfrac{\dfrac{\dfrac{\vdots}{[\Theta] \vdash \mathsf{U}_k \ , \mathsf{V} \ , \Gamma \vdash}}{[\Theta] \, ?\lambda_k;\mathsf{U}_k \ , !\lambda_k;\mathsf{V} \ , \Gamma \vdash}\,[\textsc{Prefix}]}{\dfrac{\dfrac{\dfrac{\vdots}{[\Theta'] \, ?\lambda_k;\mathsf{U}_k \ , !\lambda_k;\mathsf{V} \ , \Gamma' \vdash}}{[\Theta'] \bigwedge_{i\in I}?\lambda_i;\mathsf{U}_i \ , !\lambda_k;\mathsf{V} \ , \Gamma' \vdash}\,[\textsc{Meet}]}{\vdots}}$$

   Note, by race freedom, if $\lambda_j = \lambda_k$ then $j = k$, hence only one branch can be selected in rule [Meet] to enable the rule [Prefix]. Hence the above application of rule [Intr] is deterministic.
2. Alternatively, on some path no [Prefix] is ever applied to type $!\lambda_k;\mathsf{V}$ and there is a [Leaf] axiom of the following form, with an corresponding ancestor [Fix-μ] rule as follows:

$$\dfrac{\dfrac{\dfrac{}{[\Theta' \, [\![\, !\lambda_k;\mathsf{V} \ , \mu\mathbf{t}.\mathsf{W}' \ , \Gamma] \, !\lambda_k;\mathsf{V} \ , \mu\mathbf{t}.\mathsf{W}' \ , \Gamma \vdash}\,[\textsc{Leaf}]}{\vdots}}{\dfrac{[\Theta \, [\![\, !\lambda_k;\mathsf{V} \ , \mu\mathbf{t}.\mathsf{W}' \ , \Gamma] \, !\lambda_k;\mathsf{V} \ , \mathsf{W}'\left\{{}^{\mu\mathbf{t}.\mathsf{W}'}/_{\mathbf{t}}\right\} \ , \Gamma \vdash}{[\Theta] \, !\lambda_k;\mathsf{V} \ , \mu\mathbf{t}.\mathsf{W}' \ , \Gamma \vdash}\,[\textsc{Fix-}\mu]}$$

   In this case, by the participant condition in the race free condition, each λ_j such that $j \in I$ can only match an output in the type $!\lambda_k;\mathsf{V}$. Hence there must also be no [Prefix]

applied to any λ_i in $\bigwedge_{i\in I} ?\lambda_i;\mathsf{U}_i$ between the [Leaf] and the corresponding [Fix-μ]. Hence either $\bigwedge_{i\in I} ?\lambda_i;\mathsf{U}_i$ appears in Γ, or there is some $j \in I$ such that $?\lambda_j;\mathsf{U}_j$ for $j \in I$ appears in Γ.

In paths in the proof satisfying the first case above, simply remove the relevant instance of the rule [Intr] below the rule in the proof, replace $\bigwedge_{i\in I} ?\lambda_i;\mathsf{U}_i$ and $!\lambda_k;\mathsf{V}$ with U_k and V.

In paths in the proof satisfying the second case above where both $\bigwedge_{i\in I} ?\lambda_i;\mathsf{U}_i$ and $!\lambda_k;\mathsf{V}$ are never touched, simply replacing these formulae with U_k and V everywhere in the given path. In cases where $?\lambda_j;\mathsf{U}_i$ appears in Γ, there must be an instance of rule [Join] below the rule [Fix-μ] that introduced Γ or the following form.

$$\frac{[\Theta''] \, !\lambda_k;\mathsf{V} \, , \, ?\lambda_j;\mathsf{U}_j \, , \, \Gamma'' \vdash}{[\Theta''] \, !\lambda_k;\mathsf{V} \, , \, \bigwedge_{i\in I} ?\lambda_i;\mathsf{U}_i \, , \, \Gamma'' \vdash}$$

Since, by the participant condition, we know that in this path we never apply [Prefix] to λ_j, we can safely remove the above rule instances from the proof and replace $?\lambda_j;\mathsf{U}_j$ with U_k along that path.

After applying the above proof transformation, we obtain a proof of $\mathsf{U}_k \, , \mathsf{V} \, , \mathsf{W} \vdash$. Hence $\mathsf{U}_k \otimes \mathsf{V} \otimes \mathsf{W} \leq \mathtt{ok}$ as required.

Case of fixed points. Assume $\vdash \mathtt{rec}X.P \parallel Q \colon \mathtt{ok}$ holds. By Theorem 9, for some T, we have $\vdash \mathtt{rec}X.P \parallel Q \colon \mathsf{T}$, without using subsumption, and $\mathsf{T} \leq \mathtt{ok}$. Consider the transition $\mathtt{rec}X.P \parallel Q \longrightarrow P\{^{\mathtt{rec}X.P}/_X\} \parallel Q$.

By Lemma 15, we have there exist types U and V and type variable $\mathbf{t}$ such that $\mathsf{T} = \mu\mathbf{t}.\mathsf{U} \otimes \mathsf{V}$ and $\vdash \mathtt{rec}X.P \colon \mu\mathbf{t}.\mathsf{U}$ and $\vdash Q \colon \mathsf{V}$. Now, by Lemma 16, $\vdash P\{^{\mathtt{rec}X.P}/_X\} \colon \mathsf{U}\{^{\mu\mathbf{t}.\mathsf{U}}/_{\mathbf{t}}\}$. Therefore, we have $\vdash P\{^{\mathtt{rec}X.P}/_X\} \parallel Q \colon \mathsf{U}\{^{\mu\mathbf{t}.\mathsf{U}}/_{\mathbf{t}}\} \otimes \mathsf{V}$.

Now, since $\vdash \mu\mathbf{t}.\mathsf{U} \, , \mathsf{V}$ is provable and $\mu\mathbf{t}.\mathsf{U}, \mathsf{U}\{^{\mu\mathbf{t}.\mathsf{U}}/_{\mathbf{t}}\} \vdash$ is provable, by Theorem 5, we have $\mathsf{U}\{^{\mu\mathbf{t}.\mathsf{U}}/_{\mathbf{t}}\} \, , \mathsf{V} \vdash$ is provable. Hence $\mathsf{U}\{^{\mu\mathbf{t}.\mathsf{U}}/_{\mathbf{t}}\} \otimes \mathsf{V} \leq \mathtt{ok}$, as required. ◀

▶ **Theorem 21** (Theorem 12)**.** *Any race-free multiparty-compatible network is deadlock free.*

Proof. Assume $\vdash P \colon \mathtt{ok}$ holds and P is race free. Consider the form of P. Either P has a fixed point or internal choice at the head of a process, hence is ready to act. Hence, there exists Q such that $P \longrightarrow Q$. Otherwise we have a process equivalent to the following form.

$$!\lambda_1;Q_1 \parallel \ldots \parallel !\lambda_m;Q_m \parallel \Sigma_{i\in I_1} ?\lambda_i^1;R_i^1 \parallel \ldots \parallel \Sigma_{i\in I_n} ?\lambda_i^n;R_i^n \parallel 1 \parallel \ldots \parallel 1$$

There are two cases to consider as follows.

In the first case, $m = n = 0$; hence we have $P = 1 \parallel \ldots \parallel 1$. Therefore, $P \equiv 1$ and hence the processes is successfully terminated.

Otherwise, observe, by Theorem 9, there exists T such that $\vdash P \colon \mathsf{T}$ without using subsumption and $\mathsf{T} \leq \mathtt{ok}$. Also, observe, by Theorem 15, there exists U_i and V_i^i such that $\mathsf{T} = !\lambda_1;\mathsf{U}_1 \otimes \ldots \otimes !\lambda_m;\mathsf{U}_m \otimes \bigwedge_{i\in I_1} ?\lambda_i^1;\mathsf{V}_i^1 \parallel \ldots \otimes \bigwedge_{i\in I_n} ?\lambda_i^n;\mathsf{V}_i^n \otimes \mathtt{ok} \otimes \ldots \otimes \mathtt{ok}$ and $\vdash Q_k \colon \mathsf{U}_k$ and $\vdash R_j^\ell \colon \mathsf{V}_j^\ell$, for all j, k and ℓ.

In the proof of $T \vdash$, there must be at least one application of the rule [Prefix]. Due to the absence of $\parr$ in T, the only other rules that may be applied before the bottommost instances of rule [Prefix] are the rules [Par] and [Meet]. In order to apply the rule [Prefix], there exists j, k and ℓ such $j \in I_\ell$ and $\lambda_k = \lambda_j^\ell$, allowing a proof tree of the following form.

$$\dfrac{\dfrac{\dfrac{[\Theta]\,\mathsf{T}_k\,,\,\mathsf{U}_i^\ell\,,\,\Gamma\vdash}{[\Theta]\,!\lambda_k;\mathsf{T}_k\,,\,?\lambda_j^\ell;\mathsf{U}_j^\ell\,,\,\Gamma\vdash}}{\vdots}}{\dfrac{[\Theta]\,!\lambda_k;\mathsf{T}_k\,,\,?\lambda_j^\ell;\mathsf{U}_j^\ell\,,\,\Gamma\vdash}{[\Theta]\,!\lambda_k;\mathsf{T}_k\,,\,\bigwedge\limits_{i\in I_\ell}?\lambda_i^\ell;\mathsf{U}_i^\ell\,,\,\Gamma\vdash}}$$

$$\dfrac{\vdots}{T\vdash}$$

Thus, simply due to the existence of such a matching pair of inputs and outputs, we have a transition of the form.

$$\begin{array}{l}!\lambda_1;Q_1 \parallel \ldots \parallel !\lambda_k;Q_k \parallel \ldots \parallel !\lambda_m;Q_m\\ \|\Sigma_{i\in I_1}?\lambda_i^1;R_i^1 \parallel \ldots \parallel \Sigma_{i\in I_\ell}?\lambda^n\ell_i;R_i^\ell \parallel \ldots \|\\ \Sigma_{i\in I_n}?\lambda_i^n;R_i^n \parallel 1 \parallel \ldots \parallel 1\end{array} \longrightarrow \begin{array}{l}!\lambda_1;Q_1 \parallel \ldots \parallel Q_k \parallel \ldots \parallel !\lambda_m;Q_m\\ \|\Sigma_{i\in I_1}?\lambda_i^1;R_i^1 \parallel \ldots \parallel R_j^\ell \parallel \ldots\\ \|\Sigma_{i\in I_n}?\lambda_i^n;R_i^n \parallel 1 \parallel \ldots \parallel 1\end{array}$$

Thus we certainly have that either $P \equiv 1$ or there exists Q such that $P \longrightarrow Q$.

Finally, by Lemma 20, since R is race free, we have that for all R such that $P \longrightarrow R$, $\vdash R\colon \mathtt{ok}$ and furthermore, by Lemma 17, R is race free, as required. Hence, deadlock freedom is established by coinduction. ◀

B The Precise Relationship to Linear Logic

For a self-contained presentation, we summarise the related non-commutative logic [15] on which this work builds, formulated in the calculus of structures [28]. We adjust the syntax to match the body of the paper. The rules of MAV [34] are presented as in Fig. 7, where the calculus of structures allows rules to be applied in any context $\mathcal{C}\{\ \cdot\ \}$ and the structural congruence $\equiv$ can be applied at any point in a proof.

$$\dfrac{}{\mathtt{ok}\vdash}\ \text{success} \qquad \dfrac{\mathcal{C}\{\ \mathtt{ok}\ \}\vdash}{\mathcal{C}\{\ !\lambda\otimes ?\lambda\ \}\vdash}\ \text{atomic interaction}$$

$$\dfrac{\mathcal{C}\{\ (\mathsf{T}\otimes\mathsf{V})\,;(\mathsf{U}\otimes\mathsf{W})\ \}\vdash}{\mathcal{C}\{\ (\mathsf{T}\,;\mathsf{U})\otimes(\mathsf{V}\,;\mathsf{W})\ \}\vdash}\ \text{seq} \qquad \dfrac{\mathcal{C}\{\ \mathsf{T}\parr(\mathsf{U}\otimes\mathsf{V})\ \}\vdash}{\mathcal{C}\{\ (\mathsf{T}\parr\mathsf{U})\otimes\mathsf{V}\ \}\vdash}\ \text{switch}$$

$$\dfrac{\mathcal{C}\{\ (\mathsf{T}\vee\mathsf{V})\,;(\mathsf{U}\vee\mathsf{W})\ \}\vdash}{\mathcal{C}\{\ (\mathsf{T}\,;\mathsf{U})\vee(\mathsf{V}\,;\mathsf{W})\ \}\vdash}\ \text{medial} \qquad \dfrac{\mathcal{C}\{\ (\mathsf{T}\otimes\mathsf{U})\vee(\mathsf{T}\otimes\mathsf{V})\ \}\vdash}{\mathcal{C}\{\ T\otimes(\mathsf{U}\vee\mathsf{V})\ \}\vdash}\ \text{external}$$

$$\dfrac{\mathcal{C}\{\ \mathsf{T}\ \}\vdash}{\mathcal{C}\{\ \mathsf{T}\wedge\mathsf{U}\ \}\vdash}\ \text{left} \qquad \dfrac{\mathcal{C}\{\ \mathsf{U}\ \}\vdash}{\mathcal{C}\{\ \mathsf{T}\wedge\mathsf{U}\ \}\vdash}\ \text{right} \qquad \dfrac{\mathcal{C}\{\ \mathtt{ok}\ \}\vdash}{\mathcal{C}\{\ \mathtt{ok}\vee\mathtt{ok}\ \}\vdash}\ \text{tidy}$$

$$\begin{array}{c}(\mathsf{T}\parr\mathsf{U})\parr\mathsf{V}\equiv\mathsf{T}\parr(\mathsf{U}\parr\mathsf{V})\\ \mathsf{T}\parr\mathtt{ok}\equiv\mathsf{T}\\ \mathsf{T}\parr\mathsf{U}\equiv\mathsf{U}\parr\mathsf{T}\end{array} \qquad \begin{array}{c}\mathtt{ok}\,;\mathsf{T}\equiv\mathsf{T}\\ \mathsf{T}\,;\mathtt{ok}\equiv\mathsf{T}\\ (\mathsf{T}\,;\mathsf{U})\,;\mathsf{V}\equiv\mathsf{T}\,;(\mathsf{U}\,;\mathsf{V})\end{array} \qquad \begin{array}{c}(\mathsf{T}\otimes\mathsf{U})\otimes\mathsf{V}\equiv\mathsf{T}\otimes(\mathsf{U}\otimes\mathsf{V})\\ \mathsf{T}\otimes\mathtt{ok}\equiv\mathsf{T}\\ \mathsf{T}\otimes\mathsf{U}\equiv\mathsf{U}\otimes\mathsf{T}\end{array}$$

Figure 7 Inference and structural rules for proof system MAV (formalising provability of duals).

We extend the notion of a co-type to local types with sequential composition.

$$\overline{(\mathsf{T} \wedge \mathsf{U})} = \overline{\mathsf{T}} \vee \overline{\mathsf{U}} \qquad \overline{(\mathsf{T} \vee \mathsf{U})} = \overline{\mathsf{T}} \wedge \overline{\mathsf{U}} \qquad \overline{\mathsf{T} \parr \mathsf{U}} = \overline{\mathsf{T}} \otimes \overline{\mathsf{U}} \qquad \overline{\mathsf{T} \otimes \mathsf{U}} = \overline{\mathsf{T}} \parr \overline{\mathsf{U}}$$

$$\overline{(\mathsf{T} \ ; \ \mathsf{U})} = \overline{\mathsf{T}} \ ; \ \overline{\mathsf{U}} \qquad \overline{\mathtt{ok}} = \mathtt{ok} \qquad \overline{!\lambda} = ?\lambda \qquad \overline{?\lambda} = !\lambda$$

Notice the only difference compared to the co-type transformation for Session (Def. 2) is that any type may appear to the left of sequential composition, not only an atomic send or receive action. The following result generalises *cut elimination* to the calculus of structures.

▶ **Theorem 22** (Horne 2015 [34])**.** *In the system in Fig. 7, if* $\mathcal{C}\{\ \overline{\mathsf{T}} \parr \mathsf{T}\ \} \vdash$ *holds then we can construct a proof of* $\mathcal{C}\{\ \mathtt{ok}\ \} \vdash$*.*

The related work [15, 34], from which the above is extracted, clarifies that, as for Session in the body of this paper, MAV defines a rich notion of multiparty subtyping and compatibility.

The following result formally relating MAV and Session is a corollary of cut elimination (each direction of the implication follows from cut elimination in one of the two systems).

▶ **Corollary 23.** *If* T *is a session type, as in Def. 1 but without fixed points, then* $\mathsf{T} \vdash$ *in Session if and only if* $\mathsf{T} \vdash$ *in System MAV.*

Finally, observe that MAV is a conservative extension of linear logic with mix and, the above corollary proves the finite fragment of Session is also a fragment of MAV.

Session Types with Arithmetic Refinements

Ankush Das
Carnegie Mellon University, Pittsburgh, PA, USA
http://www.cs.cmu.edu/~ankushd
ankushd@cs.cmu.edu

Frank Pfenning
Carnegie Mellon University, Pittsburgh, PA, USA
http://www.cs.cmu.edu/~fp
fp@cs.cmu.edu

Abstract

Session types statically prescribe bidirectional communication protocols for message-passing processes. However, simple session types cannot specify properties beyond the type of exchanged messages. In this paper we extend the type system by using index refinements from linear arithmetic capturing intrinsic attributes of data structures and algorithms. We show that, despite the decidability of Presburger arithmetic, type equality and therefore also subtyping and type checking are now undecidable, which stands in contrast to analogous dependent refinement type systems from functional languages. We also present a practical, but incomplete algorithm for type equality, which we have used in our implementation of Rast, a concurrent session-typed language with arithmetic index refinements as well as ergometric and temporal types. Moreover, if necessary, the programmer can propose additional type bisimulations that are smoothly integrated into the type equality algorithm.

2012 ACM Subject Classification Theory of computation → Process calculi; Theory of computation → Linear logic; Theory of computation → Logic and verification; Computing methodologies → Concurrent programming languages; Theory of computation → Type theory

Keywords and phrases Session Types, Refinement Types, Type Equality

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.13

Funding *Ankush Das*: funded by the National Science Foundation under SaTC Award 1801369 and CAREER Award 1845514.
Frank Pfenning: funded by the National Science Foundation under Grant No. 1718276.

1 Introduction

Session types [24, 42] provide a structured way of prescribing communication protocols in message-passing systems. This paper focuses on *binary session types* governing the interactions along channels with two endpoints. They arise either directly as part of a program notation [25], or as the result of *endpoint projection* of multi-party session types [26] and are thus of central importance in the study of message-passing concurrency. Moreover, a Curry-Howard correspondence relates propositions of linear logic to session types [8, 43, 9], further evidence for their fundamental nature.

Once recursion is introduced for session types as well as processes, we are confronted with the question as to what is the correct notion of type equality since its use in type checking is inescapable. Gay and Hole [17] convincingly answer this question and also provide a practical algorithm for subtyping (which implies an algorithm for type equality). First, since the endpoints of channels need to agree on a type (or possibly two dual types) for communication, recursive types should be a priori *structural* rather than *nominal*. Second, types should be equal if their observable communication behaviors are indistinguishable. This means that two types should be equal if there is a *bisimulation* between them. This is particularly elegant since the definition is independent of any particular programming

language in which session types are embedded, or whether they are checked statically or dynamically. The algorithm for type equality then constructs a bisimulation. It terminates because the number of pairs of types that might be related by the bisimulation is finite.

Like any type system, basic session types are limited in the kind of properties they can express, which has led to some generalizations such as polymorphic [43, 7, 21] and context-free [38] session types, each with its own questions for type equality. In this paper we propose a natural *linear arithmetic refinement* of session types, which allows us to capture a number of significant properties of message-passing communication such as size or value of data structures, number of messages exchanged or delay in those messages. In conception, this refinement is closely related to *indexed types* or *value-dependent types* familiar from functional languages [47, 46, 37], where the indices are arithmetic expressions.

To our surprise, despite an eminently decidable index domain, the type equality problem becomes undecidable. We show this via a reduction from the non-halting problem for two-counter machines [30]. Analyzing this reduction in detail shows that the problem is already undecidable for a single type constructor (pick either internal ($\oplus$) or external ($\&$) choice, in addition to arithmetic refinements). While our type system is equirecursive to aid in the simplicity of programming, even retreating to isorecursive types leaves the problem undecidable. Finally, one may be tempted to blame the quantifiers in Presburger arithmetic, but our reduction shows that even if we restrict ourselves to linear arithmetic with universal prefix quantification only, type equality remains undecidable.

A retrenchment to a *nominal* interpretation of recursive types would rule out too many programs and complicate communications, so we develop a sound but incomplete algorithm. Our experience with the Rast implementation [14] to date shows that it is effective in practice (see Section 6 for further discussion).

Most closely related is the design of LiquidPi [22], but it refines only basic data types such as int rather than equirecursively defined session types. The resulting system has a decidable subtyping problem and even type inference (under reasonable assumptions on the constraint domain), but it cannot express many of our motivating examples. Along similar lines, refinements of basic data types together with subtyping have been proposed for *runtime monitoring* of binary session-typed communication [20, 19]. Label-dependent session types [39] also support types indexed by natural numbers using a fixed schema of iteration with a particular unfolding equality, rather than arbitrary recursion and bisimulation. Zhou et al. [49, 48] refine base types with arithmetic expressions in the context of multiparty session types without recursive types. In this simpler setting, they obtain a decidable notion of type equality. Further related work can be found in Section 7.

2 Basic Session Types

We review the basic language of binary session types. We take the intuitionistic point of view [8, 9], since our experiments and motivating examples have been carried out in Rast [14]. Changes for a classical view [43] are minimal and do not affect our results or algorithms. We would add a type $\bot$ dual to $\mathbf{1}$, and replace the $\multimap$ operator with $\parr$ with only minor changes to the remainder of the development.

A, B, C	::=	$\oplus\{\ell : A_\ell\}_{\ell \in L}$	send label $k \in L$	continue at type A_k
	$\mid$	$\&\{\ell : A_\ell\}_{\ell \in L}$	receive label $k \in L$	continue at type A_k
	$\mid$	$A \otimes B$	send channel $a : A$	continue at type B
	$\mid$	$A \multimap \mathrm{B}$	receive channel $a : A$	continue at type B
	$\mid$	$\mathbf{1}$	send close message	no continuation
	$\mid$	V	defined type variable	

We provide a brief description of the operational behavior of the types from the provider's point of view. The provider of $\oplus\{\ell : A_\ell\}_{\ell \in L}$ sends a label $k \in L$ and continues to provide A_k. Dually, the provider of $\&\{\ell : A_\ell\}_{\ell \in L}$ receives one of the labels in L. The provider of $A \otimes B$ sends a channel of type A and continues to provide B, whereas the process providing $A \multimap B$ receives a channel of type A and provides B. Finally, the provider of $\mathbf{1}$ sends a close message and terminates.

We assume that labels $\ell \in L$ (for a finite, nonempty set L) and close messages can be observed, but the identity of channels can not. Instead any *communication* along channels that are sent and received can be observed in turn. Based on this notion, we adopt *type bisimulations* from Gay and Hole [17]. Rather than an explicit recursive type constructor μ we postulate a *signature* Σ with definitions of type variables V.

$$\text{Signature} \quad \Sigma \quad ::= \quad \cdot \mid \Sigma, V = A$$

In a *valid signature* all definitions $V = A$ are *contractive*, that is, A is not itself a type variable. This allows us to take an *equirecursive* view of type definitions, which means that unfolding a type definition does not require communication. We can easily adapt our definitions to an *isorecursive* view [28, 15] with explicit unfold messages (see the remark at the end of Section 4). All type variables V occurring in a valid signature may refer to each other and must be defined, and all type variables defined in a signature must be distinct.

▶ **Definition 1.** *We define* $\mathsf{unfold}_\Sigma(V) = A$ *if* $V = A \in \Sigma$ *and* $\mathsf{unfold}_\Sigma(A) = A$ *otherwise.*

▶ **Definition 2.** *A relation* $\mathcal{R}$ *on types is a* type bisimulation *if* $(A, B) \in \mathcal{R}$ *implies that for* $S = \mathsf{unfold}_\Sigma(A)$, $T = \mathsf{unfold}_\Sigma(B)$ *we have*

- *If* $S = \oplus\{\ell : A_\ell\}_{\ell \in L}$ *then* $T = \oplus\{\ell : B_\ell\}_{\ell \in L}$ *and* $(A_\ell, B_\ell) \in \mathcal{R}$ *for all* $\ell \in L$.
- *If* $S = \&\{\ell : A_\ell\}_{\ell \in L}$ *then* $T = \&\{\ell : B_\ell\}_{\ell \in L}$ *and* $(A_\ell, B_\ell) \in \mathcal{R}$ *for all* $\ell \in L$.
- *If* $S = A_1 \otimes A_2$, *then* $T = B_1 \otimes B_2$ *and* $(A_1, B_1) \in \mathcal{R}$ *and* $(A_2, B_2) \in \mathcal{R}$.
- *If* $S = A_1 \multimap A_2$, *then* $T = B_1 \multimap B_2$ *and* $(A_1, B_1) \in \mathcal{R}$ *and* $(A_2, B_2) \in \mathcal{R}$.
- *If* $S = \mathbf{1}$ *then* $T = \mathbf{1}$.

▶ **Definition 3.** *We say that* A *is* equal *to* B, *written* $A \equiv B$, *if there is a type bisimulation* $\mathcal{R}$ *such that* $(A, B) \in \mathcal{R}$.

As two simple running examples we use an interface to a queue and the representation of binary numbers as sequences of bits.

▶ **Example 4** (Queues, v1)**.** A queue provider offers a choice (indicated by &) of either receiving an **ins** label followed by a channel of type A (denoted by $\multimap$) to insert into the queue, or a **del** label to delete an element from the queue. In the latter case, the queue provider has a choice (indicated by $\oplus$) of either responding with the label **none** (if there is no element in the queue) and closes the channel (indicated by $\mathbf{1}$), or the label some followed by an element of type A (denoted by $\otimes$) and recurses to await the next round of interactions. We view queue_A as a family of types, one for each A, to avoid introducing explicit polymorphic type constructors.

$$\begin{aligned}\mathsf{queue}_A = \&\{&\mathbf{ins} : A \multimap \mathsf{queue}_A,\\ &\mathbf{del} : \oplus\{\mathbf{none} : \mathbf{1},\\ &\qquad \mathbf{some} : A \otimes \mathsf{queue}_A\}\}\end{aligned}$$

▶ **Example 5** (Binary Numbers, v1)**.** A process representing a binary number either sends a label **e** representing the number 0 and closes the channel, or one of the labels **b0** (bit 0) or **b1** (bit 1) followed by remaining bits (by recursing). We assume a "little endian" form, that is, the least significant bit is sent first.

$$\mathsf{bin} = \oplus\{\mathbf{b0} : \mathsf{bin}, \mathbf{b1} : \mathsf{bin}, \mathbf{e} : \mathbf{1}\}$$

As examples of message sequences along a fixed channel, we would have

$\mathbf{e}$; close	representing 0
$\mathbf{b0}$; $\mathbf{e}$; close	also representing 0
$\mathbf{b0}$; $\mathbf{b1}$; $\mathbf{e}$; close	representing 2
$\mathbf{b1}$; $\mathbf{b0}$; $\mathbf{b1}$; $\mathbf{b1}$; $\mathbf{e}$; close	representing 13

3 Arithmetic Refinements

Before we extend our language of types formally, we revisit the examples in order to motivate the specific constructs available. We write $V[\overline{e}]$ for a type indexed by a sequence of arithmetic expressions e. Since it has been appropriate for most of our examples, we restrict ourselves to natural numbers rather than arbitrary integers.

▶ **Example 6** (Queues, v2)**.** The provider of a queue should be constrained to answer **none** exactly if the queue contains no elements and **some** if it is nonempty. The queue type from Example 4 does not express this. This means a client may need to have some redundant branches to account for responses that should be impossible. We now define the type $\mathsf{queue}_A[n]$ to stand for a queue with exactly n elements.

$$\begin{aligned}\mathsf{queue}_A[n] = \&\{&\mathbf{ins} : A \multimap \mathsf{queue}_A[n+1],\\ &\mathbf{del} : \oplus\{\mathbf{none} : ?\{n = 0\}.\,\mathbf{1},\\ &\qquad\mathbf{some} : ?\{n > 0\}.\,A \otimes \mathsf{queue}_A[n-1]\}\}\end{aligned}$$

The first branch is easy to understand: if we add an element to a queue of length n, it subsequently contains $n + 1$ elements. In the second branch we *constrain* the arithmetic variable n to be equal to 0 if the provider sends **none** and positive if the provider sends **some**. In the latter case, we subtract one from the length after an element has been dequeued.

Conceptually, the type $?\{\phi\}.\,A$ means that the provider must *send* a proof p of ϕ, so it corresponds to $\exists p : \phi.\,A$. A characteristic of *type refinement*, in contrast to fully dependent types, is that the computation of A and thus, the execution of processes can only depend on the *existence* of a proof, but not on its form (known in type theory as *proof irrelevance*). More concretely, the process types and terms cannot refer to the proof p. This irrelevance property combined with the decidability of our index domain means that no actual proof needs to be sent (since one can be constructed from ϕ automatically, if needed), just a token *asserting* its existence. There is also a dual constructor $!\{\phi\}.\,A$ that licenses the *assumption* of ϕ, which, conceptually, corresponds to *receiving* a proof of ϕ.

▶ **Example 7** (Binary Numbers, v2)**.** The indexed type $\mathsf{bin}[n]$ should represent a binary number with value n. Because the least significant bit comes first, we expect, for example, that $\mathsf{bin}[n] = \oplus\{\mathbf{b0} : ?\{2 \mid n\}.\,\mathsf{bin}[n/2], \ldots\}$ ($a \mid b$ denotes a divides b). However, while divisibility is available in Presburger arithmetic, division itself is not; instead, we can express the constraint and the index of the recursive occurrence using quantification.

$$\begin{aligned}\mathsf{bin}[n] = \oplus\{&\mathbf{b0} : \exists k.\,?\{n = 2 * k\}.\,\mathsf{bin}[k],\\ &\mathbf{b1} : \exists k.\,?\{n = 2 * k + 1\}.\,\mathsf{bin}[k],\\ &\mathbf{e} : ?\{n = 0\}.\,\mathbf{1}\}\end{aligned}$$

As a further refinement, we could rule out leading zeros by adding the constraint $n > 0$ in the branch for **b0** (in branch **b1**, $n = 2k + 1$ implies $n > 0$ so the constraint implicitly holds).

The type $\exists n.\, A$ means that the provider must send a natural number i and proceed at type $A[i/n]$, corresponding to existential quantification in arithmetic. The dual universal quantifier $\forall n.\, A$ requires the provider to receive a number i and proceed at type $A[i/n]$.

We now extend our definitions to account for these new constructs. Below, i represents a constant, n is a natural number variable and $(i \mid e)$ means i divides e.

Types	A	$::=$	$\ldots$	
		$\mid$	$?\{\phi\}.\, A$	assert ϕ — continue at type A
		$\mid$	$!\{\phi\}.\, A$	assume ϕ — continue at type A
		$\mid$	$\exists n.\, A$	send number i — continue at type $A[i/n]$
		$\mid$	$\forall n.\, A$	receive number i — continue at type $A[i/n]$
		$\mid$	$V[\overline{e}]$	variable instantiation
Arith. Expressions	e	$::=$	$i \mid e + e \mid e - e \mid i \times e \mid (i \mid e) \mid n$	
Arith. Propositions	ϕ	$::=$	$e = e \mid e > e \mid \top \mid \bot \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \exists n.\, \phi \mid \forall n.\, \phi$	
Signature	Σ	$::=$	$\cdot \mid \Sigma, V[\overline{n} \mid \phi] = A$	

An indexed type definition $V[\overline{n} \mid \phi] = A$ containing an optional proposition ϕ requires every instantiation $\overline{e}$ (in $V[\overline{e}]$) of the sequence of variables $\overline{n}$ to satisfy $\phi[\overline{e}/\overline{n}]$. This is verified statically when a type signature is checked for validity, as defined below. We use $\mathcal{V}$ for a collection of arithmetic variables and $\mathcal{C}$ (to signify *constraints*) for an arithmetic proposition occurring among the antecedents of a judgment. We then have the following rules defining the validity of signatures ($\vdash \Sigma$ *signature*), declarations ($\vdash_\Sigma \Sigma'$ *valid*), and types ($\mathcal{V}\ ;\ \mathcal{C} \vdash_\Sigma A$ *valid*) where $\mathcal{V}$ is a collection of arithmetic variables including all free variables in constraint $\mathcal{C}$ and type A. We silently rename variables so that n does not already occur in $\mathcal{V}$ in the $\exists V$ and $\forall V$ rules. We also call upon the semantic entailment judgment $\mathcal{V}\ ;\ \mathcal{C} \vDash \phi$ which means that $\forall \mathcal{V}.\, \mathcal{C} \supset \phi$ holds in arithmetic and $\vDash \phi$ abbreviates $\cdot\ ;\ \top \vDash \phi$.

$$\frac{\vdash_\Sigma \Sigma \textit{ valid}}{\vdash \Sigma \textit{ signature}} \qquad \frac{}{\vdash_\Sigma (\cdot) \textit{ valid}} \qquad \frac{\vdash_\Sigma \Sigma' \textit{ valid} \quad \overline{n}\ ;\ \phi \vdash_\Sigma A \textit{ valid} \quad A \neq V'[\overline{e}']}{\vdash_\Sigma \Sigma', V[\overline{n} \mid \phi] = A \textit{ valid}}$$

$$\frac{\mathcal{V}\ ;\ \mathcal{C} \wedge \phi \vdash_\Sigma A \textit{ valid}}{\mathcal{V}\ ;\ \mathcal{C} \vdash_\Sigma ?\{\phi\}.\, A \textit{ valid}}\ ?V \qquad \frac{\mathcal{V}\ ;\ \mathcal{C} \wedge \phi \vdash_\Sigma A \textit{ valid}}{\mathcal{V}\ ;\ \mathcal{C} \vdash_\Sigma !\{\phi\}.\, A \textit{ valid}}\ !V$$

$$\frac{\mathcal{V}, n\ ;\ \mathcal{C} \vdash_\Sigma A \textit{ valid}}{\mathcal{V}\ ;\ \mathcal{C} \vdash_\Sigma \exists n.\, A \textit{ valid}}\ \exists V^n \qquad \frac{\mathcal{V}, n\ ;\ \mathcal{C} \vdash_\Sigma A \textit{ valid}}{\mathcal{V}\ ;\ \mathcal{C} \vdash_\Sigma \forall n.\, A \textit{ valid}}\ \forall V^n$$

$$\frac{V[\overline{n} \mid \phi] = A \in \Sigma \quad \mathcal{V}\ ;\ \mathcal{C} \vDash \phi[\overline{e}/\overline{n}]}{\mathcal{V}\ ;\ \mathcal{C} \vdash_\Sigma V[\overline{e}] \textit{ valid}}\ \mathsf{tdef}$$

We elide the compositional rules for all the other type constructors. Since we like to work over natural numbers rather than integers, it is convenient to assume that every definition $V[\overline{n}] = A$ abbreviates $V[\overline{n} \mid \overline{n} \geq 0] = A$. This means that in valid signatures every occurrence $V[\overline{e}]$ is such that $\overline{e} \geq 0$ follows from the known constraints.

▶ **Example 8.** The declaration

$$\begin{aligned}\mathsf{queue}_A[n] = \&\{&\mathbf{ins} : A \multimap \mathsf{queue}_A[n+1],\\ &\mathbf{del} : \oplus\{\mathbf{none} : ?\{n = 0\}.\, \mathbf{1},\\ &\qquad \mathbf{some} : ?\{n > 0\}.\, A \otimes \mathsf{queue}_A[n-1]\}\}\end{aligned}$$

is valid: in the **ins** branch, we verify $(n\ ;\ n \geq 0 \vDash n + 1 \geq 0)$ while checking validity of $\mathsf{queue}_A[n+1]$ with rule tdef; in the **some** branch, we add $n > 0$ to our constraint $\mathcal{C}$ (due to rule $?V$) and verify $(n\ ;\ n \geq 0 \wedge n > 0 \vDash n - 1 \geq 0)$ while checking validity of $\mathsf{queue}_A[n-1]$.

Unfolding a definition must now substitute for the arithmetic variables we abstract over.

▶ **Definition 9.** $\mathsf{unfold}_\Sigma(V[\overline{e}]) = A[\overline{e}/\overline{n}]$ *if* $V[\overline{n} \mid \phi] = A \in \Sigma$ *and* $\mathsf{unfold}_\Sigma(A) = A$ *otherwise.*

We say that a type is *closed* if it contains no free arithmetic variables n.

▶ **Definition 10.** *A relation $\mathcal{R}$ on closed valid types is a type bisimulation if $(A, B) \in \mathcal{R}$ implies that for $S = \mathsf{unfold}_\Sigma(A)$, $T = \mathsf{unfold}_\Sigma(B)$ we have the following conditions (in addition to those of Definition 2):*

- *If $S = ?\{\phi\}.\, A'$ then $T = ?\{\psi\}.\, B'$ and either (i) $\vDash \phi$, $\vDash \psi$, and $(A', B') \in \mathcal{R}$, or (ii) $\vDash \neg\phi$ and $\vDash \neg\psi$.*
- *If $S = !\{\phi\}.\, A'$ then $T = !\{\psi\}.\, B'$ and either (i) $\vDash \phi$, $\vDash \psi$, and $(A', B') \in \mathcal{R}$, or (ii) $\vDash \neg\phi$ and $\vDash \neg\psi$*
- *If $S = \exists m.\, A'$ then $T = \exists n.\, B'$ and for all $i \in \mathbb{N}$, $(A'[i/m], B'[i/n]) \in \mathcal{R}$.*
- *If $S = \forall m.\, A'$ then $T = \forall n.\, B'$ and for all $i \in \mathbb{N}$, $(A'[i/m], B'[i/n]) \in \mathcal{R}$.*

We also extend the notation $A \equiv B$ to this richer set of types.

An interesting point here is provided by the cases *(ii)* in the first two clauses. Because the type must be closed, we know that ϕ and ψ will be either true or false. If both are false, no messages can be sent along a channel of either type and therefore the continuation types A' and B' are irrelevant when considering type equality.

Fundamentally, due to the presence of arbitrary recursion and therefore non-termination, we always view a type as a restriction of what a process *might* send or receive along some channel, but it is neither *required* to send a message nor *guaranteed* to receive one. This is similar to functional programming with unrestricted recursion where an expression may not return a value. The definition based on observability of messages is then somewhat strict, as exemplified by the next example.

▶ **Example 11.** Consider the following two types

$$\begin{aligned}\mathsf{bin}[n] = \oplus\{&\mathbf{b0} : \exists k.\, ?\{n = 2 * k\}.\, \mathsf{bin}[k],\\ &\mathbf{b1} : \exists k.\, ?\{n = 2 * k + 1\}.\, \mathsf{bin}[k],\\ &\mathbf{e} : ?\{n = 0\}.\, \mathbf{1}\}\end{aligned} \qquad \begin{aligned}\mathsf{zero} = \oplus\{&\mathbf{b0} : \exists k.\, ?\{k = 0\}.\, \mathsf{zero},\\ &\mathbf{e} : ?\{0 = 0\}.\, \mathbf{1}\}\end{aligned}$$

We might expect $\mathsf{bin}[0] \equiv \mathsf{zero}$, but this is not so. A process of type $\mathsf{bin}[0]$ could send the label **b1** and an arbitrary value for k and then just loop forever (because there is no proof of $0 = 2k + 1$). The type zero cannot exhibit this behavior so the types are not equivalent.

In our implementation, missing branches for a choice in process definitions are *reconstructed* with a continuation that marks it as impossible, which is then verified by the type checker. We found this simple technique significantly limited the need for subtyping or explicit definition of types such as zero – instead, we just work with $\mathsf{bin}[0]$.

The following properties of type equality are straightforward.

▶ **Lemma 12** (Properties of Type Equality)**.** *The relation $\equiv$ is reflexive, symmetric, transitive and a congruence on closed valid types.*

4 Undecidability of Type Equality

We prove the undecidability of type equality by exhibiting a reduction from an undecidable problem about two counter machines.

The type system allows us to simulate two counter machines [30]. Intuitively, arithmetic constraints allow us to model branching zero-tests available in the machine. This, coupled with recursion in the language of types, establishes undecidability. Remarkably, a small fragment

of our language containing only type definitions, internal choice ($\oplus$) and assertions ($?\{\phi\}.\, A$) where ϕ just contains constraints $n = 0$ and $n > 0$ is sufficient to prove undecidability. Moreover, the proof still applies if we treat types isorecursively. In the remainder of this section we provide some details of the undecidability proof.

▶ **Definition 13** (Two Counter Machine). *A two counter machine $\mathcal{M}$ is defined as a sequence of instructions $\iota_1, \iota_2, \ldots, \iota_m$ and c_j ($j \in \{1, 2\}$) as the two counters. Each instruction has one of the following forms.*

- *"*$\mathsf{inc}(c_j); \mathsf{goto}\ k$*" (increment counter j by 1 and go to instruction k)*
- *"*$\mathsf{zero}(c_j)?\ \mathsf{goto}\ k : \mathsf{dec}(c_j); \mathsf{goto}\ l$*" (if the value of counter j is 0, go to instruction k, else decrement the counter by 1 and go to instruction l)*
- *"*halt*" (stop computation)*

A configuration C of the machine $\mathcal{M}$ is defined as a triple (i, c_1, c_2), where i denotes the number of the current instruction and c_j's denote the value of the counters. A configuration C' is defined as the successor configuration of C, written as $C \mapsto C'$ if C' is the result of executing the i-th instruction on C. If $\iota_i = \mathsf{halt}$, then $C = (i, c_1, c_2)$ has no successor configuration. The computation of $\mathcal{M}$ is the unique maximal sequence $\rho = \rho(0)\rho(1)\ldots$ such that $\rho(i) \mapsto \rho(i+1)$ and $\rho(0) = (1, 0, 0)$. Either ρ is infinite, or ends in (i, c_1, c_2) such that $\iota_i = \mathsf{halt}$ and $c_1, c_2 \in \mathbb{N}$.

The *halting problem* refers to determining whether the computation of a two counter machine $\mathcal{M}$ with given initial values $c_1, c_2 \in \mathbb{N}$ is finite. Both the halting problem and its dual, the *non-halting problem*, are undecidable.

▶ **Theorem 14.** *Given a valid signature Σ, two natural number variables m and n, and two types A and B such that $m, n\ ;\ \top \vdash_\Sigma A, B$ valid. Then it is undecidable whether for concrete $i, j \in \mathbb{N}$ we have $A[i/m, j/n] \equiv B[i/m, j/n]$.*

Proof. Given a two counter machine, we construct a signature Σ and two types A and B with free arithmetic variables m and n such that the computation of the machine starting with initial counter values i and j is infinite iff $A[i/m, j/n] \equiv B[i/m, j/n]$ in Σ.

We define types $T_{\text{inf}} = \oplus\{\ell : T_{\text{inf}}\}$ and $T'_{\text{inf}} = \oplus\{\ell' : T'_{\text{inf}}\}$ for *distinct* labels ℓ and ℓ'. Next, for every instruction ι_i, we define types T_i and T'_i based on the form of the instruction.

- Case ($\iota_i = \mathsf{inc}(c_1); \mathsf{goto}\ k$): We define

$$\begin{array}{lcl} T_i[c_1, c_2] & = & \oplus\{\mathsf{inc}_1 : T_k[c_1 + 1, c_2]\} \\ T'_i[c_1, c_2] & = & \oplus\{\mathsf{inc}_1 : T'_k[c_1 + 1, c_2]\} \end{array}$$

- Case ($\iota_i = \mathsf{inc}(c_2); \mathsf{goto}\ k$): We define

$$\begin{array}{lcl} T_i[c_1, c_2] & = & \oplus\{\mathsf{inc}_2 : T_k[c_1, c_2 + 1]\} \\ T'_i[c_1, c_2] & = & \oplus\{\mathsf{inc}_2 : T'_k[c_1, c_2 + 1]\} \end{array}$$

- Case ($\iota_i = \mathsf{zero}(c_1)?\ \mathsf{goto}\ k : \mathsf{dec}(c_1); \mathsf{goto}\ l$): We define

$$\begin{array}{lcl} T_i[c_1, c_2] & = & \oplus\{\mathsf{zero}_1 : ?\{c_1 = 0\}.\, T_k[c_1, c_2], \mathsf{dec}_1 : ?\{c_1 > 0\}.\, T_l[c_1 - 1, c_2]\} \\ T'_i[c_1, c_2] & = & \oplus\{\mathsf{zero}_1 : ?\{c_1 = 0\}.\, T'_k[c_1, c_2], \mathsf{dec}_1 : ?\{c_1 > 0\}.\, T'_l[c_1 - 1, c_2]\} \end{array}$$

- Case ($\iota_i = \mathsf{zero}(c_2)?\ \mathsf{goto}\ k : \mathsf{dec}(c_2); \mathsf{goto}\ l$): We define

$$\begin{array}{lcl} T_i[c_1, c_2] & = & \oplus\{\mathsf{zero}_2 : ?\{c_2 = 0\}.\, T_k[c_1, c_2], \mathsf{dec}_2 : ?\{c_2 > 0\}.\, T_l[c_1, c_2 - 1]\} \\ T'_i[c_1, c_2] & = & \oplus\{\mathsf{zero}_2 : ?\{c_2 = 0\}.\, T'_k[c_1, c_2], \mathsf{dec}_2 : ?\{c_2 > 0\}.\, T'_l[c_1, c_2 - 1]\} \end{array}$$

- Case ($\iota_i = \mathsf{halt}$): We define

$$\begin{array}{lcl} T_i[c_1, c_2] & = & T_{\text{inf}} \\ T'_i[c_1, c_2] & = & T'_{\text{inf}} \end{array}$$

We set type $A = T_1[m, n]$ and $B = T'_1[m, n]$. Now suppose, the counter machine $\mathcal{M}$ is initialized in the state $(1, i, j)$. The type equality question we ask is whether $T_1[i, j] \equiv T'_1[i, j]$. The two types only differ at the halting instruction. If $\mathcal{M}$ does not halt, the two types capture exactly the same communication behavior, since the halting instruction is never reached and they agree on all other instructions. If $\mathcal{M}$ halts, the first type proceeds with label ℓ and the second with ℓ' and they are therefore not equal. Hence, the two types are equal iff $\mathcal{M}$ does not halt. ◀

We can easily modify this reduction for an isorecursive interpretation of types, by wrapping $\oplus\{\mathbf{unfold} : \_\_\}$ around the right-hand side of each type definition to simulate the unfold message. We also see that a host of other problems are undecidable, such as determining whether two types with free arithmetic variables are equal for all instances. This is the problem that arises while type-checking parametric process definitions.

5 A Practical Algorithm for Type Equality

Despite its undecidability, we have designed a coinductive algorithm for soundly approximating type equality. Similar to Gay and Hole's algorithm, it proceeds by attempting to construct a bisimulation. Due to the undecidability of the problem, our algorithm can terminate in three different states: (1) we have succeeded in constructing a bisimulation, (2) we have found a counterexample to type equality by finding a place where the types may exhibit different behavior, or (3) we have terminated the search without a definitive answer. From the point of view of type-checking, both (2) and (3) are interpreted as a failure to type-check (but there is a recourse; see Subsection 5.2). Our algorithm is expressed as a set of inference rules where the execution of the algorithm corresponds to the bottom-up construction of a deduction. The algorithm is deterministic (no backtracking) and the implementation is quite efficient in practice (see Section 6).

One of the basic operations in Gay and Hole's algorithm is *loop detection*, that is, we have to determine that we have already added an equation $A \equiv B$ to the bisimulation we are constructing. Since we must treat *open types*, that is, types with free arithmetic variables subject to some constraints, determining if we have considered an equation already becomes a difficult operation. To that purpose we make an initial pass over the given type and introduce fresh *internal names* abstracted over their free type variables and known constraints. In the resulting signature defined type variables and type constructors alternate and we can perform loop detection entirely on type definitions (whether internal or external).

▶ **Example 15** (Queues, v3)**.** After creating internal names $\%i$ for the type of queue we obtain the following signature (here parametric in A).

$$\begin{array}{ll} \mathsf{queue}_A[n] = \&\{\mathbf{ins} : \%0[n], \mathbf{del} : \%1[n]\} & \\ \%0[n] = A \multimap \mathsf{queue}_A[n+1] & \%3 = \mathbf{1} \\ \%1[n] = \oplus\{\mathbf{none} : \%2[n], \mathbf{some} : \%4[n]\} & \%4[n] = ?\{n > 0\}.\,\%5[n] \\ \%2[n] = ?\{n = 0\}.\,\%3 & \%5[n \mid n > 0] = A \otimes \mathsf{queue}_A[n-1] \end{array}$$

Based on the invariants established by internal names, the algorithm only needs to compare two type variables or two structural types. The rules are shown in Figure 1. The judgment has the form $\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash A \equiv B$ where $\mathcal{V}$ contains the free arithmetic

$$\frac{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash A_\ell \equiv B_\ell \quad (\forall \ell \in L)}{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash \oplus\{\ell : A_\ell\}_{\ell\in L} \equiv \oplus\{\ell : B_\ell\}_{\ell\in L}}\ \oplus \qquad \frac{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash A_\ell \equiv B_\ell \quad (\forall \ell \in L)}{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash \&\{\ell : A_\ell\}_{\ell\in L} \equiv \&\{\ell : B_\ell\}_{\ell\in L}}\ \&$$

$$\frac{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash A_1 \equiv B_1 \qquad \mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash A_2 \equiv B_2}{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash A_1 \otimes A_2 \equiv B_1 \otimes B_2}\ \otimes$$

$$\frac{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash A_1 \equiv B_1 \qquad \mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash A_2 \equiv B_2}{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash A_1 \multimap A_2 \equiv B_1 \multimap B_2}\ \multimap \qquad \frac{}{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash \mathbf{1} \equiv \mathbf{1}}\ \mathbf{1}$$

$$\frac{\mathcal{V}\ ;\ \mathcal{C} \vDash \phi \leftrightarrow \psi \qquad \mathcal{V}\ ;\ \mathcal{C} \wedge \phi\ ;\ \Gamma \vdash A \equiv B}{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash ?\{\phi\}.\, A \equiv ?\{\psi\}.\, B}\ ? \qquad \frac{\mathcal{V}\ ;\ \mathcal{C} \vDash \phi \leftrightarrow \psi \qquad \mathcal{V}\ ;\ \mathcal{C} \wedge \phi\ ;\ \Gamma \vdash A \equiv B}{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash !\{\phi\}.\, A \equiv !\{\psi\}.\, B}\ !$$

$$\frac{\mathcal{V}, k\ ;\ \mathcal{C}\ ;\ \Gamma \vdash A[k/m] \equiv B[k/n]}{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash \exists m.\, A \equiv \exists n.\, B}\ \exists^k \qquad \frac{\mathcal{V}, k\ ;\ \mathcal{C}\ ;\ \Gamma \vdash A[k/m] \equiv B[k/n]}{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash \forall m.\, A \equiv \forall n.\, B}\ \forall^k$$

$$\frac{\mathcal{V}\ ;\ \mathcal{C} \vDash \bot}{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash A \equiv B}\ \bot \qquad \frac{\mathcal{V}\ ;\ \mathcal{C} \vDash e_1 = e_1' \wedge \ldots \wedge e_n = e_n'}{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash V[\overline{e}] \equiv V[\overline{e}']}\ \mathsf{refl}$$

$$\frac{\begin{array}{c} V_1[\overline{v_1} \mid \phi_1] = A \in \Sigma \qquad V_2[\overline{v_2} \mid \phi_2] = B \in \Sigma \\ \gamma = \langle \mathcal{V}\ ;\ \mathcal{C}\ ;\ V_1[\overline{e_1}] \equiv V_2[\overline{e_2}] \rangle \\ \mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma, \gamma \vdash A[\overline{e_1}/\overline{v_1}] \equiv B[\overline{e_2}/\overline{v_2}] \end{array}}{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash V_1[\overline{e_1}] \equiv V_2[\overline{e_2}]}\ \mathsf{expd}$$

$$\frac{\langle \mathcal{V}'\ ;\ \mathcal{C}'\ ;\ V_1[\overline{e_1}'] \equiv V_2[\overline{e_2}'] \rangle \in \Gamma \qquad \mathcal{V}\ ;\ \mathcal{C} \vDash \exists \mathcal{V}'.\, \mathcal{C}' \wedge \overline{e_1}' = \overline{e_1} \wedge \overline{e_2}' = \overline{e_2}}{\mathcal{V}\ ;\ \mathcal{C}\ ;\ \Gamma \vdash V_1[\overline{e_1}] \equiv V_2[\overline{e_2}]}\ \mathsf{def}$$

Figure 1 Algorithmic Rules for Type Equality.

variables in the constraints $\mathcal{C}$ and the types A and B, and Γ is a collection of *closures* $\langle \mathcal{V}'\ ;\ \mathcal{C}'\ ;\ V_1'[\overline{e_1}'] \equiv V_2'[\overline{e_2}'] \rangle$. If a derivation can be constructed, all ground instances of all closures are included in the resulting bisimulation (see the proof of Theorem 20). A ground instance $V_1'[\overline{e_1}'[\sigma']] \equiv V_2'[\overline{e_2}'[\sigma']]$ is given by a substitution σ' over variables in $\mathcal{V}'$ such that $\vDash \mathcal{C}'[\sigma']$.

The rules for type constructors simply compare the components. If the type constructors (or the label sets in the $\oplus$ and & rules) do not match, then type equality fails (having constructed a counterexample to bisimulation) unless the $\bot$ rule applies. This rules handles the case where the constraints are contradictory and no communication is possible.

The rule of reflexivity is needed explicitly here (but not in the version of Gay and Hole) because due to the incompleteness of the algorithm we may otherwise fail to recognize type variables with equal index expressions as equal.

Now we come to the key rules, expd and def. In the expd rule we expand the definitions of $V_1[\overline{e_1}]$ and $V_2[\overline{e_2}]$, and we also add the closure $\langle \mathcal{V}\ ;\ \mathcal{C}\ ;\ V_1[\overline{e_1}] \equiv V_2[\overline{e_2}] \rangle$ to Γ. Since the equality of $V_1[\overline{e_1}]$ and $V_2[\overline{e_2}]$ must hold for all its ground instances, the extension of Γ with the corresponding closure remembers exactly that. We can ignore the propositions ϕ_1 and ϕ_2 since the validity of types (rule tdef in Section 3) ensures that both $\vDash \phi_1[\overline{e_1}/\overline{v_1}]$ and $\vDash \phi_2[\overline{e_2}/\overline{v_2}]$ hold.

In the def rule we close off the derivation successfully if all instances of the equation $V_1[\overline{e_1}] \equiv V_2[\overline{e_2}]$ are already instances of a closure in Γ. This is checked by the entailment in the second premise, $\mathcal{V}\ ;\ \mathcal{C} \vDash \exists \mathcal{V}'.\, \mathcal{C}' \wedge \overline{E_1} = \overline{e_1} \wedge \overline{E_2} = \overline{e_2}$. This entailment is verified as a

closed $\forall\exists$ arithmetic formula, even if the original constraints $\mathcal{C}$ and $\mathcal{C}'$ do not contain any quantifiers. While for Presburger arithmetic we can decide such a proposition using quantifier elimination, other constraint domains may not permit such a decision procedure.

The algorithm so far is sound, but potentially nonterminating because when encountering variable/variable equations, we can use the expd rule indefinitely. To ensure termination, we restrict the expd rule to the case where *no* closure with the same type variables V_1 and V_2 is already present in Γ. This also removes the overlap between these two rules. Note that if type variables have no parameters, our algorithm specializes to Gay and Hole's (with the small optimizations of reflexivity and internal naming), which means our algorithm is sound and complete on unindexed types.

As an extension, our algorithm also allows the programmer to specify a *depth bound* k. This informs the algorithm to apply the expd rule until there are at most k closures with the same type variables V_1 and V_2 in Γ.

▶ **Example 16** (Integer Counter)**.** An integer counter with increment (**inc**), decrement (**dec**) and sign-test (**sgn**) operations provides type $\mathsf{intctr}[x, y]$, where the current value of the counter is $x - y$ for natural numbers x and y.

$$\begin{aligned}\mathsf{intctr}[x,y] = \&\{&\mathbf{inc} : \mathsf{intctr}[x+1,y],\\ &\mathbf{dec} : \mathsf{intctr}[x,y+1],\\ &\mathbf{sgn} : \oplus\{\mathbf{neg} : ?\{x<y\}.\,\mathsf{intctr}[x,y],\\ &\qquad\mathbf{zer} : ?\{x=y\}.\,\mathsf{intctr}[x,y],\\ &\qquad\mathbf{pos} : ?\{x>y\}.\,\mathsf{intctr}[x,y]\}\}\end{aligned}$$

Under this definition our algorithm verifies, for example, that an increment followed by a decrement does not change the counter value. That is,

$$x, y\ ;\ \top\ ;\ \cdot \vdash \mathsf{intctr}[x,y] \equiv \mathsf{intctr}[x+1,y+1]$$

where we have elided the assumptions $x, y \geq 0$. When applying expd, we assume $\gamma = \langle x', y'\ ;\ \top\ ;\ \mathsf{intctr}[x',y'] \equiv \mathsf{intctr}[x'+1,y'+1]\rangle$. Then, for example, in the first branch (for inc) we conclude $x, y\ ;\ \top\ ;\ \gamma \vdash \mathsf{intctr}[x+1,y] \equiv \mathsf{intcr}[x+2,y+1]$ using the def rule and the entailment $x, y\ ;\ \top \vDash \exists x'.\,\exists y'.\,x' = x+1 \wedge y' = y \wedge x'+1 = x+2 \wedge y'+1 = y+1$. The other branches are similar.

As exemplified by the above example, a distinguishing feature of our algorithm is that it goes beyond *reflexivity*. Essentially, $V[\overline{e_1}] \equiv V[\overline{e_2}]$ can hold even if $\overline{e_1} \neq \overline{e_2}$. This is in contrast with traditional refinement languages such as DML [46] that use reflexivity as the only criterion for equality on indexed type names.

5.1 Soundness of the Type Equality Algorithm

We prove that the type equality algorithm is sound with respect to the definition of type equality. The soundness is proved by constructing a type bisimulation from a derivation of the algorithmic type equality judgment. We sketch the key points of the proofs.

The first gap we have to bridge is that the type bisimulation is defined only for closed types, because observations can only arise from communication along channels which, at runtime, will be of closed type. So, if we can derive $\mathcal{V}\ ;\ \mathcal{C}\ ;\ \cdot \vdash A \equiv B$ then we should interpret this as stating that for all ground substitutions σ over $\mathcal{V}$ such that $\vDash \mathcal{C}[\sigma]$ we have $A[\sigma] \equiv B[\sigma]$.

▶ **Definition 17.** *Given a relation $\mathcal{R}$ on valid ground types and two types A and B such that $\mathcal{V}$; $\mathcal{C} \vdash A, B$ valid, we write $\forall \mathcal{V}.\mathcal{C} \Rightarrow A \equiv_{\mathcal{R}} B$ if for all ground substitutions σ over $\mathcal{V}$ such that $\vDash \mathcal{C}[\sigma]$ we have $(A[\sigma], B[\sigma]) \in \mathcal{R}$.*

Furthermore, we write $\forall \mathcal{V}.\mathcal{C} \Rightarrow A \equiv B$ if there exists a type bisimulation $\mathcal{R}$ such that $\forall \mathcal{V}.\mathcal{C} \Rightarrow A \equiv_{\mathcal{R}} B$.

Note that if $\mathcal{V}$; $\mathcal{C} \vDash \bot$, then $\forall \mathcal{V}.\mathcal{C} \Rightarrow A \equiv B$ is vacuously true, since there does not exist a ground substitution σ such that $\vDash \mathcal{C}[\sigma]$.

A key lemma is the following, which is needed to show the soundness of the def rule.

▶ **Lemma 18.** *Suppose $\forall \mathcal{V}'.\mathcal{C}' \Rightarrow V_1[\overline{e_1}'] \equiv_{\mathcal{R}} V_2[\overline{e_2}']$ holds. Further assume that $\mathcal{V}$; $\mathcal{C} \vDash \exists \mathcal{V}'.\mathcal{C}' \wedge \overline{e_1}' = \overline{e_1} \wedge \overline{e_2}' = \overline{e_2}$ for some $\mathcal{V}, \mathcal{C}, \overline{e_1}, \overline{e_2}$. Then, $\forall \mathcal{V}.\mathcal{C} \Rightarrow V_1[\overline{e_1}] \equiv_{\mathcal{R}} V_2[\overline{e_2}]$ holds.*

Proof. To prove $\forall \mathcal{V}.\mathcal{C} \Rightarrow V_1[\overline{e_1}] \equiv_{\mathcal{R}} V_2[\overline{e_2}]$, it is sufficient to show that $V_1[\overline{e_1}[\sigma]] \equiv_{\mathcal{R}} V_2[\overline{e_2}[\sigma]]$ for any substitution σ over $\mathcal{V}$ such that $\vDash \mathcal{C}[\sigma]$. Applying this substitution to $\mathcal{V}$; $\mathcal{C} \vDash \exists \mathcal{V}'.\mathcal{C}' \wedge \overline{e_1}' = \overline{e_1} \wedge \overline{e_2}' = \overline{e_2}$, we infer $\exists \mathcal{V}'.\mathcal{C}' \wedge \overline{e_1}' = \overline{e_1}[\sigma] \wedge \overline{e_2}' = \overline{e_2}[\sigma]$ since $\vDash \mathcal{C}[\sigma]$. Thus, there exists σ' over $\mathcal{V}'$ such that $\vDash \mathcal{C}'[\sigma']$ holds, and $\overline{e_1}'[\sigma'] = \overline{e_1}[\sigma]$ and $\overline{e_2}'[\sigma'] = \overline{e_2}[\sigma]$. And since $\forall \mathcal{V}'.\mathcal{C}' \Rightarrow V_1[\overline{e_1}'] \equiv_{\mathcal{R}} V_2[\overline{e_2}']$, we deduce that for any ground substitution (including the current one) σ' over $\mathcal{V}'$, $V_1[\overline{e_1}'[\sigma']] \equiv_{\mathcal{R}} V_2[\overline{e_2}'[\sigma']]$ holds. This implies that $V_1[\overline{e_1}[\sigma]] \equiv_{\mathcal{R}} V_2[\overline{e_2}[\sigma]]$ since $\overline{e_1}'[\sigma'] = \overline{e_1}[\sigma]$ and $\overline{e_2}'[\sigma'] = \overline{e_2}[\sigma]$. ◀

We construct the bisimulation from a derivation of $\mathcal{V}$; $\mathcal{C}$; $\Gamma \vdash A \equiv B$ by (i) collecting the conclusions of all the sequents, excepting only the def rule, and (ii) forming all ground instances from them.

▶ **Definition 19.** *Given a derivation $\mathcal{D}$ of $\mathcal{V}$; $\mathcal{C}$; $\Gamma \vdash A \equiv B$, we define the set $\mathcal{S}(\mathcal{D})$ of closures. For each sequent $\mathcal{V}'$; $\mathcal{C}'$; $\Gamma' \vdash A' \equiv B'$ (except the conclusion of the* def *rule) we include the closure $\langle \mathcal{V}' ; \mathcal{C}' ; A' \equiv B' \rangle$ in $\mathcal{S}(\mathcal{D})$.*

▶ **Theorem 20** (Soundness). *If $\mathcal{V}$; $\mathcal{C}$; $\cdot \vdash A \equiv B$, then $\forall \mathcal{V}.\mathcal{C} \Rightarrow A \equiv B$.*

Proof. We are given a derivation $\mathcal{D}_0$ of $\mathcal{V}_0$; $\mathcal{C}_0$; $\cdot \vdash A_0 \equiv B_0$. Construct $\mathcal{S}(\mathcal{D}_0)$ and define a relation $\mathcal{R}$ on closed valid types as follows:

$$\mathcal{R} = \{(A[\sigma], B[\sigma]) \mid \langle \mathcal{V} ; \mathcal{C} ; A \equiv B \rangle \in \mathcal{S}(\mathcal{D}_0) \text{ and } \sigma \text{ over } \mathcal{V} \text{ with } \vDash \mathcal{C}[\sigma]\}$$

We prove that $\mathcal{R}$ is a type bisimulation. Then our theorem follows since the closure $\langle \mathcal{V}_0 ; \mathcal{C}_0 ; A_0 \equiv B_0 \rangle \in \mathcal{S}(\mathcal{D}_0)$.

Consider $(A[\sigma], B[\sigma]) \in \mathcal{R}$ where $\langle \mathcal{V} ; \mathcal{C} ; A \equiv B \rangle \in \mathcal{S}(\mathcal{D}_0)$ for some σ over $\mathcal{V}$ and $\vDash \mathcal{C}[\sigma]$.

First, consider the case where $\mathcal{V}$; $\mathcal{C} \vDash \bot$. Under such a constraint, $\mathcal{V}$; $\mathcal{C}$; $\cdot \vdash A \equiv B$ holds true due to the $\bot$ rule. Furthermore, $\forall \mathcal{V}.\mathcal{C} \Rightarrow A \equiv B$ holds vacuously, and the algorithm is sound. For the remaining cases, we case analyze on the structure of $A[\sigma]$ and assume that there exists a ground substitution σ such that $\vDash \mathcal{C}[\sigma]$.

Consider the case where $A = \oplus\{\ell : A_\ell\}_{\ell \in L}$. Since A and B are both structural, $B = \oplus\{\ell : B_\ell\}_{\ell \in L}$. Since $\langle \mathcal{V} ; \mathcal{C} ; A \equiv B \rangle \in \mathcal{S}(\mathcal{D}_0)$, by definition of $\mathcal{S}(\mathcal{D}_0)$, we get $\langle \mathcal{V} ; \mathcal{C} ; A_\ell \equiv B_\ell \rangle \in \mathcal{S}(\mathcal{D}_0)$ for all $\ell \in L$. By the definition of $\mathcal{R}$, we get that $(A_\ell[\sigma], B_\ell[\sigma]) \in \mathcal{R}$. Also, $A[\sigma] = \oplus\{\ell : A_\ell[\sigma]\}_{\ell \in L}$ and similarly, $B[\sigma] = \oplus\{\ell : B_\ell[\sigma]\}_{\ell \in L}$. Hence, $\mathcal{R}$ satisfies the appropriate closure condition for a type bisimulation.

Next, consider the case where $A = ?\{\phi\}.A'$. Since A and B are both structural, $B = ?\{\psi\}.B'$. Since $\langle \mathcal{V} ; \mathcal{C} ; A \equiv B \rangle \in \mathcal{S}(\mathcal{D}_0)$, we obtain $\mathcal{V}$; $\mathcal{C} \vDash \phi \leftrightarrow \psi$ and $\langle \mathcal{V} ; \mathcal{C} \wedge \phi ; A' \equiv B' \rangle \in \mathcal{S}(\mathcal{D}_0)$. Thus, for any substitution σ such that $\vDash \mathcal{C}[\sigma] \wedge \phi[\sigma]$, we get that $(A'[\sigma], B'[\sigma]) \in \mathcal{R}$ with $A[\sigma] = ?\{\phi[\sigma]\}.A'[\sigma]$ and $B[\sigma] = ?\{\psi[\sigma]\}.B'[\sigma]$. Since $\vDash \phi[\sigma]$ and and $\mathcal{V}$; $\mathcal{C} \vDash \phi \leftrightarrow \psi$ we also obtain $\vDash \psi[\sigma]$ and the closure condition is satisfied.

Next, consider the case where $A = \exists m.\, A'$. Since A and B are both structural, $B = \exists n.\, B'$. Since $\langle \mathcal{V} \,;\, \mathcal{C} \,;\, A \equiv B \rangle \in \mathcal{S}(\mathcal{D}_0)$, we get that $\langle \mathcal{V}, k \,;\, \mathcal{C} \,;\, A'[k/m] \equiv B'[k/n] \rangle \in \mathcal{S}(\mathcal{D}_0)$. Since k was chosen fresh and does not occur in $\mathcal{C}$, we obtain that for any $i \in \mathbb{N}$ we have $\vDash \mathcal{C}[\sigma, i/k]$ and therefore $(A'[\sigma, i/k], B'[\sigma, i/k]) \in \mathcal{R}$ for all $i \in \mathbb{N}$ and the closure condition is satisfied.

The only case where a conclusion is not added to $\mathcal{S}(\mathcal{D}_0)$ is the def rule. In this case, adding $(\forall \mathcal{V}.\, \mathcal{C} \Rightarrow V_1[\overline{e_1}] \equiv V_2[\overline{e_2}])$ is redundant: Lemma 18 states that $V_1[\overline{e_1}[\sigma]] \equiv_{\mathcal{R}} V_2[\overline{e_2}[\sigma]]$ which implies $(V_1[\overline{e_1}[\sigma]], V_2[\overline{e_2}[\sigma]]) \in \mathcal{R}$. ◀

5.2 Type Equality Declarations

Even though the type equality algorithm in Section 5 is incomplete, we have yet to find a natural example where it fails after we added reflexivity as a general rule. But since we cannot see a simple reason why this should be so, we made our type equality algorithm extensible by the programmer via an additional form of declaration

$$\forall \mathcal{V}.\, \mathcal{C} \Rightarrow V_1[\overline{e_1}] \equiv V_2[\overline{e_2}]$$

in signatures. Let Γ_Σ denote the set of all such declarations. Then we check

$$\mathcal{V} \,;\, \mathcal{C} \,;\, \Gamma_\Sigma \vdash V_1[\overline{e_1}] \equiv V_2[\overline{e_2}]$$

for each such declaration, seeding the construction of a bisimulation with all the given equations. Then, when type-checking has to decide the equality of two types, it starts not with the empty context Γ but with Γ_Σ. Our soundness proof can easily accommodate this more general algorithm.

▶ **Example 21** (Queues, v4)**.** Consider the two types $\mathsf{queue}_A[n]$ and $\mathsf{queue}'_A[n]$, both representing queue data structures, but $\mathsf{queue}'_A[n]$ is rooted at 1.

$$\begin{aligned}\mathsf{queue}_A[n] = \&\{&\mathbf{ins} : A \multimap \mathsf{queue}_A[n+1],\\ &\mathbf{del} : \oplus\{\mathbf{none} : ?\{n = 0\}.\, \mathbf{1},\\ &\qquad \mathbf{some} : ?\{n > 0\}.\, A \otimes \mathsf{queue}_A[n-1]\}\}\end{aligned}$$

$$\begin{aligned}\mathsf{queue}'_A[n] = \&\{&\mathbf{ins} : A \multimap \mathsf{queue}'_A[n+1],\\ &\mathbf{del} : \oplus\{\mathbf{none} : ?\{n = 1\}.\, \mathbf{1},\\ &\qquad \mathbf{some} : ?\{n > 1\}.\, A \otimes \mathsf{queue}'_A[n-1]\}\}\end{aligned}$$

Our intuition would suggest that $\mathsf{queue}_A[0] \equiv \mathsf{queue}'_A[1]$. But this cannot be directly proved by our equality algorithm. While checking this equality, our algorithm would add $\langle \cdot \,;\, \top \,;\, \mathsf{queue}_A[0] \equiv \mathsf{queue}'_A[1] \rangle$ to Γ and would continue to check $\mathsf{queue}_A[1] \equiv \mathsf{queue}'_A[2]$ (the **ins** branch). However, our closure in Γ is not sufficient to prove this goal (the def rule fails), and our algorithm reports the types may not be equal. However, we can add a general equality declaration $\forall n.\, \mathsf{queue}_A[n] \equiv \mathsf{queue}'_A[n+1]$ to the signature. This can be verified by our algorithm since it would add $\langle n \,;\, \top \,;\, \mathsf{queue}_A[n] \equiv \mathsf{queue}'_A[n+1] \rangle$ to Γ and use it to prove $\mathsf{queue}_A[n+1] \equiv \mathsf{queue}'_A[n+2]$ in the **ins** branch. Then, we will use the same equality declaration from the signature to verify $\mathsf{queue}_A[0] \equiv \mathsf{queue}'_A[1]$ by instantiating $n = 0$.

6 Implementation and Further Examples

We have implemented the algorithm presented in Section 5 as part of the Rast programming language [14], whose name derives from "Resource-Aware Session Types". Rast is based on intuitionistic linear sessions [8, 9] extended with general equirecursive types and recursively

Table 1 Case Studies.

Module	LOC	#Defs	T (ms)
arithmetic	143	8	1.325
integers	114	8	1.074
linlam	67	6	4.003
list	441	29	3.419
primes	118	8	1.646
segments	65	9	0.195
ternary	235	16	1.967
theorems	141	16	0.894
tries	308	9	5.283
Total	**1632**	**109**	**19.806**

defined processes. We do not explicitly dualize types [43] but distinguish providers and clients that are connected by a private channel. In parallel work we have proved type safety for Rast, which includes type preservation (session fidelity) and global progress (deadlock freedom). The open-source implementation is written in Standard ML and currently comprises about 7500 lines of source code [36].

Rast supports indexed types, quantifiers, and arithmetic constraints, following the presentation in this paper with minor syntactic differences. In addition, Rast has temporal [12] and ergometric [13] types that capture parallel and sequential complexity of programs. These bounds often depend on intrinsic properties of the data structures (such as the length of a queue or the value of a binary number) which are expressed as arithmetic indices.

Rast's linear type checker is bidirectional, which means that only process definitions need to be annotated with their types. In the so-called *explicit syntax* type checking is then straightforward, breaking down the structure of the type and unfolding definitions, except for calls to type equality (which are necessary for forwarding, process invocations, and sending of channels). The implementation also supports an *implicit syntax* in which some parts of the program, specifically those concerning missing branches that can be proved to be impossible using refinements, can be omitted from the source and are reconstructed. The reconstructed code is then passed through the type checker as ultimate arbiter.

We use a straightforward implementation of Cooper's algorithm [10] to decide Presburger arithmetic with two small but significant optimizations. One takes advantage of the fact that we are working over natural numbers rather than integers, the other is to eliminate constraints of the form $x = e$ by substituting e for x in order to reduce the number of variables. We also extend our solver to handle non-linear constraints. Since non-linear arithmetic is undecidable, in general, we use a normalizer which collects coefficients of each term in the multinomial expression. To check $e_1 = e_2$, we normalize $e_1 - e_2$ and check that each coefficient of the normal form is 0. To check $e_1 \geq e_2$, we normalize $e_1 - e_2$ and check that each coefficient is non-negative.

We have a variety of 21 examples implemented, totaling about 3700 lines of code, for which complete code can be found in our open source repository [36]. Table 1 describes the results for nine representative case studies: LOC describes the lines of code, #Defs shows the number of process definitions, and T (ms) shows the type-checking time in milliseconds respectively. The experiments were run on an Intel Core i5 2.7 GHz processor with 16 GB 1867 MHz DDR3 memory. We briefly describe each case study.

1. **arithmetic**: natural numbers in unary and binary representation indexed by their value and processes implementing standard arithmetic operations.
2. **integers**: an integer counter represented using two indices x and y with value $x - y$.
3. **linlam**: expressions in the linear λ-calculus indexed by their size with an *eval* process to evaluate them (see below for an excerpt).
4. **list**: lists indexed by their size with standard operations (e.g., *append, reverse, map*).
5. **primes**: implementation of the sieve of Eratosthenes.
6. **segments**: type $\mathsf{seg}[n] = \forall k.\mathsf{list}[k] \multimap \mathsf{list}[n+k]$ representing partial lists with constant-work append operation.
7. **ternary**: natural numbers represented in balanced ternary form with digits $0, 1, -1$, indexed by their value, and some standard operations on them.
8. **theorems**: processes representing (circular [15]) proofs of simple arithmetic theorems.
9. **tries**: a trie data structure to store multisets of binary numbers, with constant amortized work insertion and deletion, verified with ergometric types.

Linear λ-calculus. We briefly sketch the types in an implementation of the (untyped) linear λ-calculus in which the index objects track the size of the expression, because it uses multiple feature of the type system.

$$\begin{aligned}\mathsf{exp}[n] = \oplus\{&\mathbf{lam} : ?\{n > 0\}.\, \forall n_1.\, \mathsf{exp}[n_1] \multimap \mathsf{exp}[n_1 + n - 1],\\ &\mathbf{app} : \exists n_1.\, \exists n_2.\, ?\{n = n_1 + n_2 + 1\}.\, \mathsf{exp}[n_1] \otimes \mathsf{exp}[n_2]\}\end{aligned}$$

An expression is either a λ-abstraction (sending label **lam**) or an application (sending label **app**). In case of **lam**, the continuation receives a number n_1 and an argument of size n_1 and then behaves like the body of the λ-abstraction of size $n_1 + n - 1$. In case of **app**, it will send n_1 and n_2 such that $n = n_1 + n_2 + 1$ followed an expression of size n_1 and then behave as an expression of size n_2.

A value can only be a λ-abstraction

$$\mathsf{val}[n] = \oplus\{\mathbf{lam} : ?\{n > 0\}.\, \forall n_1.\, \mathsf{exp}[n_1] \multimap \mathsf{exp}[n_1 + n - 1]\}$$

so the **app** label is not permitted. Type checking verifies that that the result of evaluating a linear λ-term is no larger than the original term. The declaration below expresses that $\mathit{eval}\,[n]$ is client to a process sending a λ-expression of size n along channel e and provides a value of size k, where $k \leq n$.

$$(e : \mathsf{exp}[n]) \vdash \mathit{eval}\,[n] :: (v : \exists k.\, ?\{k \leq n\}.\, \mathsf{val}[k])$$

7 Further Related Work

Traditional languages with dependent type refinements such as Zenger's [47] or DML [46] only use the rule of reflexivity as a criterion for equality of indexed types. This is justified in the context of these functional languages because data types are generative and therefore *nominal* in nature. This is also true for more recent languages with linearity and value-dependent types such as Granule [32].

Session type systems that allow dependencies are label-dependent session types [39] and richer linear type theories [40, 33, 41]. Toninho et al. [40, 33] allow sufficient dependencies that, in general, proofs must be sent in some circumstances. They do not provide a type equality algorithm or implementation. In a more recent paper, Toninho et al. [41] propose a dependent type theory with rich notions of value and process equality based on $\beta\eta$-congruences

and certain process equalities, but they do not discuss decidability or algorithms for type checking or type equality. Wu and Xi [44] propose a dependent session type system based on ATS [45] formalizing type equality in terms of subtyping and regular constraint relations. They mention recursive session types as a possible extension, but do not develop them nor investigate properties of the required type equality.

Linearly refined session types [2, 16] extend the π-calculus with capabilities from a fragment of multiplicative linear logic. These capabilities encode an authorization logic enabling fine-grained specifications and are thus not directly comparable to arithmetic refinements. Session types with limited arithmetic refinements (only base types could be refined) have been proposed for the purpose of runtime monitoring [20, 19], which is complementary to our uses for static verification. They have also been proposed to capture work [13, 11] and parallel time [12], but parameterization over index objects was left to an informal meta-level and not part of the object language. Consequently, these languages contain neither constraints nor quantifiers, and the metatheory of type equality, type checking, and reconstruction in the presence of index variables was not developed.

Several other generalizations of session types for specification and verification have been proposed. Generalizing the idea of "Design by Contract" [29] to distributed domains, session types have been elaborated with logical predicates to obtain *global assertions* [4]. Actris [23] combines concurrent separation logics with session types for reasoning about message passing in the presence of other concurrency paradigms. Actris is able to prove functional correctness of a distributed merge sort, a distributed load-balancing mapper, and a variant of the map-reduce model. Context-free session types [38] are another generalization of basic session types in a different direction, essentially allowing the concatenation of sessions. This generalization has decidable type checking and type equality problems that have been shown to be efficient in practice [1].

Asynchronous session types [18] have a notion of subtyping under different assumptions regarding communication behavior [31]. The resulting subtyping relation also turns out to be undecidable [6, 27] with the development of recent practical incomplete algorithms [5]. The expressive power of asynchronous session subtyping seems incomparable to our arithmetically refined session types.

8 Conclusion

This paper explored the metatheory of session types with arithmetic refinements, showing the undecidability of type equality. Nevertheless, we have shown a sound, but incomplete algorithm that has performed well over a range of examples in our Rast implementation.

Natural extensions include nonlinear arithmetic and other constraint domains, balancing practicality of type checking with expressive power. We would also like to generalize from type equality to subtyping, replacing the notion of bisimulation with a simulation. Clearly, this will be undecidable as well, but the pioneering work by Gay and Hole and the characteristics of our algorithms suggest that it should extend cleanly and remain practical.

Finally, we would also like to generalize our approach to a mixed linear/nonlinear language [3] or all the way to adjoint session types [34, 35]. Since the main issues of type equality are orthogonal to the presence or absence of structural properties, we conjecture that the algorithm proposed here will extend to this more general setting.

References

1 Bernardo Almeida, Andreia Mordido, and Vasco T. Vasconcelos. Deciding the bisimilarity of context-free session types. In A. Biere and D. Parker, editors, *16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2020)*, pages 39–56, Dublin, Ireland, April 2020. Springer LNCS 12079.

2 Pedro Baltazar, Dimitris Mostrous, and Vasco T. Vasconcelos. Linearly refined session types. In S. Alves and I. Mackie, editors, *International Workshop on Linearity (LINEARITY 2012)*, pages 38–49, Tallinn, Estonia, April 2012. EPTCS 101.

3 Nick Benton. A mixed linear and non-linear logic: Proofs, terms and models. In L. Pacholski and J. Tiuryn, editors, *Selected Papers from the 8th International Workshop on Computer Science Logic (CSL 1994)*, pages 121–135, Kazimierz, Poland, September 1994. Springer LNCS 933. An extended version appears as Technical Report UCAM-CL-TR-352, University of Cambridge.

4 Laura Bocchi, Kohei Honda, Emilio Tuosto, and Nobuko Yoshida. A theory of design-by-contract for distributed multiparty interactions. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory*, pages 162–176, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

5 Mario Bravetti, Marco Carbone, Julien Lange, Nobuko Yoshida, and Gianluigi Zavattaro. A sound algorithm for asynchronous session subtyping. In W. Fokkink and R. van Glabbeek, editors, *30th International Conference on Concurrency Theory (CONCUR 2019)*, pages 38:1–38:16, Amsterdam, The Netherlands, August 2019. LIPIcs 140.

6 Mario Bravetti, Marco Carbone, and Gianluigi Zavattaro. Undecidability of asynchronous session subtyping. *Information & Computation*, 256:300–320, 2017.

7 Luís Caires, Jorge A. Pérez, Frank Pfenning, and Bernardo Toninho. Behavioral polymorphism and parametricity in session-based communication. In M.Felleisen and P.Gardner, editors, *Proceedings of the European Symposium on Programming (ESOP'13)*, pages 330–349, Rome, Italy, March 2013. Springer LNCS 7792.

8 Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In P.Gastin and F.Laroussinie, editors, *Proceedings of the 21st International Conference on Concurrency Theory (CONCUR 2010)*, pages 222–236, Paris, France, August 2010. Springer LNCS 6269.

9 Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Mathematical Structures in Computer Science*, 26(3):367–423, 2016. Special Issue on Behavioural Types.

10 David C. Cooper. Theorem proving in arithmetic without multiplication. *Machine Intelligence*, 7:91–99, 1972.

11 Ankush Das, Stephanie Balzer, Jan Hoffmann, Frank Pfenning, and Ishani Santurkar. Resource-aware session types for digital contracts, 2019. `arXiv:1902.06056`.

12 Ankush Das, Jan Hoffmann, and Frank Pfenning. Parallel complexity analysis with temporal session types. In M. Flatt, editor, *Proceedings of International Conference on Functional Programming (ICFP 2018)*, pages 91:1–91:30, St. Louis, Missouri, USA, September 2018. ACM.

13 Ankush Das, Jan Hoffmann, and Frank Pfenning. Work analysis with resource-aware session types. In A. Dawar and E. Grädel, editors, *Proceedings of 33rd Symposium on Logic in Computer Science (LICS'18)*, pages 305–314, Oxford, UK, July 2018.

14 Ankush Das and Frank Pfenning. Rast: Resource-aware session types with arithmetic refinements. In Z. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, pages 4:1–4:17. LIPIcs 167, June 2020. System description. To appear.

15 Farzaneh Derakhshan and Frank Pfenning. Circular proofs as session-typed processes: A local validity condition, August 2019. `arXiv:1908.01909`.

16 Juliana Franco and Vasco T. Vasconcelos. A concurrent programming language with refined session types. In S. Counsell and M. Núñez, editors, *Software Engineering and Formal Methods (SEFM 2013)*, pages 15–28, Madrid, Spain, September 2013. Springer LNCS 8368.

17 Simon J. Gay and Malcolm Hole. Subtyping for session types in the π-calculus. *Acta Informatica*, 42(2–3):191–225, 2005.

18 Simon J. Gay and Vasco T. Vasconcelos. Linear type theory for asynchronous session types. *Journal of Functional Programming*, 20(1):19–50, January 2010.

19 Hannah Gommerstadt. *Session-Typed Concurrent Contracts*. PhD thesis, Carnegie Mellon University, September 2019. Available as Technical Report CMU-CS-19-119.

20 Hannah Gommerstadt, Limin Jia, and Frank Pfenning. Session-typed concurrent contracts. In A. Ahmed, editor, *European Symposium on Programming (ESOP'18)*, pages 771–798, Thessaloniki, Greece, April 2018. Springer LNCS 10801.

21 Dennis Griffith. *Polarized Substructural Session Types*. PhD thesis, University of Illinois at Urbana-Champaign, April 2016.

22 Dennis Griffith and Elsa L. Gunter. LiquidPi: Inferrable dependent session types. In *Proceedings of the NASA Formal Methods Symposium*, pages 186–197. Springer LNCS 7871, 2013.

23 Jonas Kastberg Hinrichsen, Jesper Bengtson, and Robbert Krebbers. Actris: Session-type based reasoning in separation logic. *Proc. ACM Program. Lang.*, 4(POPL), December 2019. doi:10.1145/3371074.

24 Kohei Honda. Types for dyadic interaction. In E. Best, editor, *4th International Conference on Concurrency Theory (CONCUR 1993)*, pages 509–523. Springer LNCS 715, 1993.

25 Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In C. Hankin, editor, *7th European Symposium on Programming Languages and Systems (ESOP 1998)*, pages 122–138. Springer LNCS 1381, 1998.

26 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In G. Necula and P. Wadler, editors, *Proceedings of the 35th Symposium on Principles of Programming Language (POPL 2008)*, pages 273–284, San Francisco, California, USA, January 2008. ACM.

27 Julien Lange and Nobuko Yoshida. On the undecidability of asynchronous session subtyping. In *Proceedings of the 20th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2017)*, pages 441–457. Springer LNCS 10203, 2017.

28 Sam Lindley and J. Garrett Morris. Talking bananas: Structural recursion for session types. In J. Garrigue, G. Keller, and E. Sumii, editors, *Proceedings of the 21st International Conference on Functional Programming*, pages 434–447, Nara, Japan, September 2016. ACM.

29 Bertrand Meyer. Applying "design by contract". *Computer*, 25(10):40–51, October 1992. doi:10.1109/2.161279.

30 Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Inc., USA, 1967.

31 Dimitris Mostrous and Nobuko Yoshida. Session typing and asynchronous subtyping for the higher-order π-calculus. *Information & Computation*, 241:227–263, 2015.

32 Dominic Orchard, Vilem-Benjamin Liepelt, and Harley Eades, III. Quantitative program reasoning with graded modal types. In *International Conference on Functional Programming (ICFP 2019)*, pages 110:1–110:30, Berlin, Germany, August 2019. ACM.

33 Frank Pfenning, Luís Caires, and Bernardo Toninho. Proof-carrying code in a session-typed process calculus. In *1st International Conference on Certified Programs and Proofs (CPP 2011)*, pages 21–36, Kenting, Taiwan, December 2011. Springer LNCS 7086.

34 Frank Pfenning and Dennis Griffith. Polarized substructural session types. In A. Pitts, editor, *Proceedings of the 18th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2015)*, pages 3–22, London, England, April 2015. Springer LNCS 9034. Invited talk.

35 Klaas Pruiksma and Frank Pfenning. A message-passing interpretation of adjoint logic. In F. Martins and D. Orchard, editors, *Workshop on Programming Language Approaches to Concurrency and Communication-Centric Software (PLACES 2019)*, pages 60–79, Prague, Czech Republic, April 2019. EPTCS 291.

36 Rast language, February 2020. Version 1.01. URL: `https://bitbucket.org/fpfenning/rast/src/master/rast/`.

37 Patrick Maxim Rondon, Ming Kawaguchi, and Ranjit Jhala. Liquid types. In R. Gupta and S. Amarasinghe, editors, *Conference on Programming Language Design and Implementation (PLDI 2008)*, pages 159–169, Tuscon, Arizona, June 2008. ACM.

38 Peter Thiemann and Vasco T. Vasconcelos. Context-free session types. In *Proceedings of the 21st International Conference on Functional Programming (ICFP 2016)*, pages 462–475, Nara, Japan, September 2016. ACM.

39 Peter Thiemann and Vasco T. Vasconcelos. Label-dependent session types. In L. Birkedal, editor, *Proceedings of the Symposium on Programming Languages (POPL 2020)*, pages 67:1–67:29, New Orleans, Louisiana, USA, January 2020. ACM Proceedings on Programming Languages 4.

40 Bernardo Toninho, Luís Caires, and Frank Pfenning. Dependent session types via intuitionistic linear type theory. In P.Schneider-Kamp and M.Hanus, editors, *Proceedings of the 13th International Conference on Principles and Practice of Declarative Programming (PPDP 2011)*, pages 161–172, Odense, Denmark, July 2011. ACM.

41 Bernardo Toninho and Nobuko Yoshida. Depending on session-types processes. In C. Baier and U. Dal Lago, editors, *21st International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2018)*, pages 128–145, Thessaloniki, Greece, April 2018. Springer LNCS 10803.

42 Vasco T. Vasconcelos. Fundamentals of session types. *Information & Computation*, 217:52–70, 2012.

43 Philip Wadler. Propositions as sessions. In *Proceedings of the 17th International Conference on Functional Programming (ICFP 2012)*, pages 273–286, Copenhagen, Denmark, September 2012. ACM Press.

44 Hanwen Wu and Hongwei Xi. Dependent session types, 2017. `arXiv:1704.07004`.

45 Hongwei Xi. Applied type system: Extended abstract. In S. Berardi, M. Coppo, and F. Damiani, editors, *International Workshop on Types for Proofs and Programming (TYPES 2003)*, pages 394–408, Torino, Italy, April 2003. Springer LNCS 3085.

46 Hongwei Xi and Frank Pfenning. Dependent types in practical programming. In A. Aiken, editor, *Conference Record of the 26th Symposium on Principles of Programming Languages (POPL 1999)*, pages 214–227, San Antonio, Texas, USA, January 1999. ACM Press.

47 Christoph Zenger. Indexed types. *Theoretical Computer Science*, 187:147–165, 1997.

48 Fangyi Zhou. Refinement session types. Master's thesis, Imperial College London, 2019.

49 Fangyi Zhou, Francisco Ferreira, Rumyana Neykova, and Nobuko Yoshida. Fluid Types: Statically Verified Distributed Protocols with Refinements. In *11th Workshop on Programming Language Approaches to Concurrency and Communication-Centric Software*, 2019.

Probabilistic Analysis of Binary Sessions

Omar Inverso
Gran Sasso Science Institute, L'Aquila, Italy

Hernán Melgratti
ICC, Universidad de Buenos Aires, Conicet, Argentina

Luca Padovani
Università di Torino, Italy

Catia Trubiani
Gran Sasso Science Institute, L'Aquila, Italy

Emilio Tuosto
Gran Sasso Science Institute, L'Aquila, Italy

Abstract

We study a probabilistic variant of binary session types that relate to a class of Finite-State Markov Chains. The probability annotations in session types enable the reasoning on the probability that a session terminates successfully, for some user-definable notion of successful termination. We develop a type system for a simple session calculus featuring probabilistic choices and show that the success probability of well-typed processes agrees with that of the sessions they use. To this aim, the type system needs to track the propagation of probabilistic choices across different sessions.

2012 ACM Subject Classification Theory of computation → Type structures

Keywords and phrases Probabilistic choices, session types, static analysis, deadlock freedom

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.14

Related Version A longer version of the paper with additional example details and proofs is available at `http://dx.doi.org/10.5281/zenodo.3951070`.

Funding Hernán Melgratti, Luca Padovani and Emilio Tuosto have been partially supported by EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement No 778233.
Omar Inverso: Omar Inverso has been partially supported by MIUR project PRIN 2017FTXR7S *IT MATTERS* (Methods and Tools for Trustworthy Smart Systems).
Hernán Melgratti: Hernán Melgratti has been partially supported by UBACyT projects 20020170100544BA and 20020170100086BA and PIP project 11220130100148CO.
Catia Trubiani: Catia Trubiani has been partially supported by MIUR project PRIN 2017TWRCNB *SEDUCE* (Designing Spatially Distributed Cyber-Physical Systems under Uncertainty).

Acknowledgements The authors are grateful to the anonymous reviewers for their detailed feedback.

1 Introduction

Session types [29, 30] have consolidated as a formalism for the modular analysis of complex systems of communicating processes. A *session* is a private channel connecting two (sometimes more) processes, each owning one *endpoint* of the session and using the endpoint according to a specification – the *session type* – that constrains the sequence of messages that can be sent and received through that endpoint. As an example, the session type

$$!\mathsf{int}.(\circ \,\&\, ?\mathsf{int}.(\circ \oplus T)) \tag{1.1}$$

could describe (part of) an auction protocol as seen from the viewpoint of a buyer process, which sends a bid (!int) and waits for a decision from the auctioneer. The protocol proceeds in two different ways, as specified by the two sides of the branching operator &. The auctioneer

may declare that the item is sold, in which case the session terminates immediately ($\circ$), or it may inform the buyer of a different (higher) bid (?int). At that point the buyer may choose ($\oplus$) to quit the auction or to restart the same protocol, here denoted by T, with another bid.

Most session type theories are aimed at enforcing qualitative properties of a system, such as type safety, protocol compliance, deadlock and livelock freedom, and so on [30]. In these theories, branches ($\&$) and choices ($\oplus$) are given a non-deterministic interpretation since all that matters is understanding whether the system "behaves well" no matter how it evolves. In this work, we propose a session type system for *a particular quantitative analysis* of session-based networks of communicating processes. More specifically, we shift from a non-deterministic to a *probabilistic* interpretation of branches and choices in session types and study a type system aimed at determining the probability with which a particular session terminates *successfully*. Since there is no universal interpretation of "successful termination", we differentiate successful from unsuccessful termination of a session by means of a dedicated type constructor. For example, in our type system we can refine (1.1) as

$$!\mathsf{int}.(\bullet\ {}_{p}\&\ ?\mathsf{int}.(\circ\ {}_{q}\oplus T)) \tag{1.2}$$

where the session type $\bullet$ indicates successful termination and branches and choices are annotated with probabilities p and q. In particular, the auctioneer declares the item sold with probability p and answers with a counteroffer with probability $1-p$, whereas the buyer decides to quit the auction with probability q and to bid again with probability $1-q$.

From an abstract description such as (1.2), we can easily compute the probability that the interaction ends up in a particular state (*e.g.*, the probability with which the buyer wins the auction). However, (1.2) is "just" the type of one endpoint of a single session in a system, while the system itself could be much more complex: there could be many different processes involved, each making probabilistic choices affecting the behavior of faraway processes that directly or indirectly receive information about such choices through messages exchanged in sessions. Also, new processes and sessions could be created and the network topology could evolve dynamically as the system runs. How do we know that (1.2) is a faithful abstraction of our system? How do we know that the probability annotations we see in (1.2) correspond to the actual probabilities that the system evolves in a certain way? Here is where our type system comes into play: by certifying that a system of processes is well typed with respect to a given set of session types with probability annotations, we support the computation of the probability that the system evolves in certain way statically – *i.e.*, before the system runs – and solely looking at the session types we are interested in as opposed to the system itself.

Summary of contributions and structure of the paper. We define a session calculus in which processes may perform probabilistic choices (Section 2). We study a variant of session types based on a probabilistic interpretation of branches and choices so that session types correspond to a particular class of Discrete-Time Markov Chains (Section 3). We provide syntax-directed typing rules for relating processes and session types (Section 4). Well-typed processes are shown to behave probabilistically as specified by the corresponding session types. We are able to trace this correspondence not just for finite processes (Theorem 4.8) but also for processes engaged in potentially infinite interactions (Corollary 4.9). We discuss related work in Section 5 and ideas for further developments in Section 6. Some example details are provided in the appendix.

Table 1 Syntax of processes.

Domains	$p, q, r \in [0,1]$	probability	$\mid \texttt{case}\, x\, [P, Q]$	branch
	$x, y, z \in \mathcal{N}$	name	$\mid \texttt{inl}\, x.P$	left selection
			$\mid \texttt{inr}\, x.P$	right selection
Processes	$P, Q ::= \texttt{idle}$	inaction	$\mid P \mid Q$	parallel composition
	$\mid \texttt{done}\, x$	success	$\mid (x)P$	session restriction
	$\mid x?(y).P$	message input	$\mid P \,{}_p\!\boxplus\, Q$	probabilistic choice
	$\mid x!y.P$	message output	$\mid A\langle \overline{x} \rangle$	process invocation

2 A Probabilistic Session Calculus

We let p, q and r range over *probabilities*, namely real numbers in the range $[0, 1]$. We let x, y and z range over an infinite set $\mathcal{N}$ of *channel names*. We write $\overline{x}$ for finite sequences of names and other entities. Processes, ranged over by P, Q and R, are defined by the grammar in Table 1. We have two distinct terms, $\texttt{idle}$ and $\texttt{done}\, x$, for modeling inactive processes. We use $\texttt{idle}$ to denote plain termination and $\texttt{done}\, x$ to denote successful termination of session x. This way, we are able to relate the success rate resulting from processes to that inferrable from session types (Theorem 4.8). The terms $x?(y).P$ and $x!y.P$ denote a process that respectively performs an input and an output of a message y on session x and then continues as P. For simplicity, in the model we only consider messages that are themselves (session) channels, while in some examples we will also use more elaborate message types. The term $\texttt{case}\, x\, [P, Q]$ represents a process that waits for a selection (either "left" or "right") on session x and continues as either P or Q accordingly. The terms $\texttt{inl}\, x.P$ and $\texttt{inr}\, x.P$ represent processes that perform a selection (respectively "left" and "right") on session x and continue as P. Parallel composition $P \mid Q$, channel restriction $(x)P$ and process invocation $A\langle \overline{x} \rangle$ are standard. We assume that for every process variable A there is an equation $A(\overline{x}) := P$ defining it. Finally, the term $P \,{}_p\!\boxplus\, Q$ represents a process that has performed a probabilistic choice and that behaves as P with probability p and as Q with probability $1 - p$.

The notions of free and bound names are standard. In the following, we write $\mathsf{fn}(P)$ and $\mathsf{bn}(P)$ for the set of free and bound names of P, respectively. For the sake of readability, we occasionally omit $\texttt{idle}$ terms and we assume that input/output prefixes and selections bind more tightly than choices and parallel compositions. So for example, $\texttt{inl}\, x.\texttt{done}\, y \,{}_p\!\boxplus\, \texttt{inr}\, x$ is to be read $(\texttt{inl}\, x.\texttt{done}\, y) \,{}_p\!\boxplus\, (\texttt{inr}\, x.\texttt{idle})$.

The operational semantics of processes is given by a structural precongruence relation $\preccurlyeq$ and a reduction relation $\rightarrow$, which are defined by the axioms and rules in Table 2 where we abbreviate with $P \equiv Q$ the two relations $P \preccurlyeq Q$ and $Q \preccurlyeq P$. We use a pre-congruence instead of a symmetric relation because careless rewriting of processes may compromise their well typing. Nonetheless, the use of a pre-congruence does not affect the ability of processes to reduce (*cf.* Theorem 4.5) and most relations are symmetric anyway. We now describe the structural pre-congruence and reduction, focusing on the former relation since it is the only one that deals with probabilistic choices.

The relations described by S-PAR-COMM, S-NEW-COMM and S-PAR-NEW are standard and need no commentary. Axiom S-CHOICE-COMM allows us to commute a probabilistic choice. The probability needs to be suitably adjusted so as to preserve the semantics of the process. Axiom S-NO-CHOICE turns a probabilistic choice into a deterministic one when the probability is trivial. This axiom is the main motivation for adopting a pre-congruence rather than a symmetric relation. Indeed, while the symmetric relation $P \preccurlyeq P \,{}_1\!\boxplus\, Q$ makes sense operationally, it violates typing in general for the process Q can be arbitrary. On the contrary, knowing

Table 2 Structural pre-congruence and reduction of processes.

Structural pre-congruence $\boxed{P \preccurlyeq Q}$

S-NO-CHOICE
$$P \;{}_1\!\boxplus\, Q \preccurlyeq P$$

S-CHOICE-IDEM
$$P \;{}_p\!\boxplus\, P \equiv P$$

S-CHOICE-COMM
$$P \;{}_p\!\boxplus\, Q \equiv Q \;{}_{1-p}\!\boxplus\, P$$

S-NEW-COMM
$$(x)(y)P \equiv (y)(x)P$$

S-PAR-COMM
$$P \mid Q \equiv Q \mid P$$

S-PAR-CHOICE
$$(P \;{}_p\!\boxplus\, Q) \mid R \preccurlyeq (P \mid R) \;{}_p\!\boxplus\, (Q \mid R)$$

S-PAR-NEW
$$\frac{x \notin \mathsf{fn}(Q)}{(x)P \mid Q \equiv (x)(P \mid Q)}$$

S-CHOICE-ASSOC
$$\frac{pq < 1}{(P \;{}_q\!\boxplus\, Q) \;{}_p\!\boxplus\, R \equiv P \;{}_{pq}\!\boxplus\, (Q \;{}_{\frac{p-pq}{1-pq}}\!\boxplus\, R)}$$

S-PAR-ASSOC
$$\frac{\mathsf{fn}(Q) \cap \mathsf{fn}(R) \neq \emptyset}{(P \mid Q) \mid R \preccurlyeq P \mid (Q \mid R)}$$

Reduction $\boxed{P \to Q}$

R-COM
$$x!y.P \mid x?(y).Q \to P \mid Q$$

R-LEFT
$$\mathtt{inl}\; x.P \mid \mathtt{case}\; x\; [Q, R] \to P \mid Q$$

R-VAR
$$\frac{A(\overline{x}) := P}{A\langle \overline{x} \rangle \to P}$$

R-PAR
$$\frac{P \to Q}{P \mid R \to Q \mid R}$$

R-NEW
$$\frac{P \to Q}{(x)P \to (x)Q}$$

R-CHOICE
$$\frac{P \to Q}{P \;{}_p\!\boxplus\, R \to Q \;{}_p\!\boxplus\, R}$$

R-STRUCT
$$\frac{P \preccurlyeq R \to R' \preccurlyeq Q}{P \to Q}$$

that $P \;{}_1\!\boxplus\, Q$ is well typed allows us to easily derive that P alone is also well typed. Axiom S-CHOICE-IDEM states that the probabilistic choice is idempotent, namely that a probabilistic choice between equal behaviors is not really a choice. Rule S-CHOICE-ASSOC expresses the standard associativity property for probabilistic choices, which requires a normalization of the involved probabilities. Note that this rule is applicable only when $pq < 1$, or else the rightmost probability in the conclusion would be undefined. When $pq = 1$, the process can be simplified using S-NO-CHOICE. Rule S-PAR-ASSOC expresses the associativity property for the parallel composition. The side condition, requiring the middle and rightmost processes to be connected by one shared name, is needed by the type system (*cf.* Section 4). The reader might be worried by the side conditions imposed on the associativity rule, since they are limiting the ability to rewrite processes to an extent which could prevent processes to be placed next to each other and reduce according to the reduction relation. It is possible to prove a *proximity property* [31] ensuring that this is not the case, namely that it is always possible to rearrange (well-typed) processes in such a way that processes connected by a session can communicate. The symmetric relation $P \mid (Q \mid R) \preccurlyeq (P \mid Q) \mid R$ when $\mathsf{fn}(P) \cap \mathsf{fn}(Q) \neq \emptyset$ is derivable using S-PAR-ASSOC and S-PAR-COMM. Rule S-PAR-CHOICE distributes parallel compositions over probabilistic choices. This rule is pivotal in our model, for two different reasons. First, being able to distribute a process over a probabilistic choice is essential to make sure that processes connected by a session can be placed next to each other so that they can reduce according to $\to$. Second, the relation is quite challenging to handle at the typing level: when R is composed in parallel with P and Q, it might be necessary to type R differently depending on whether or not the session that connects R with P and Q is affected by the probabilistic choice. This is doable provided that R uses the session *safely*, namely if it does not delegate the session before it becomes aware of the probabilistic choice (*cf.* Section 4).

The reduction relation is standard. The base cases consist of the usual rules for communication (R-COM), branch selection (R-LEFT and R-RIGHT, the latter omitted) along with the expansion of process variables (R-VAR). Reduction is closed under parallel compositions (R-PAR), restrictions (R-NEW), probabilistic choices (R-CHOICE) and structural precongruence (R-STRUCT). Note that a probabilistic choice $P \ {}_p\boxplus\ Q$ is *persistent*, in the sense that neither P nor Q is discarded by reduction even though they morally represent two mutually-exclusive evolutions of the same process. This is one of the standard approaches for describing the semantics of probabilistic processes [28, 54, 38]. As a consequence, a process like $A := \mathtt{idle} \ {}_{0.001}\boxplus\ A$ diverges but terminates with probability 1. We will be able to state interesting properties of such processes through a soundness result that is relativized to the probability of termination.

We write $\Rightarrow$ for the reflexive, transitive closure of $\rightarrow$, we write $P \rightarrow$ if there exists Q such that $P \rightarrow Q$ and $P \not\rightarrow$ if not $P \rightarrow$. In the above example, $A \Rightarrow P$ implies $P \rightarrow$.

▶ **Example 2.1** (Auction). We end this section showing how to represent in our calculus the auction example informally described in Section 1. We define two processes, a *Buyer* and a *Seller* connected by a session x:

$$\begin{aligned} \mathit{Buyer}(x) &:= x!\mathit{bid}.\mathtt{case}\ x\ [\mathtt{done}\ x, x?(y).(\mathtt{inl}\ x \ {}_q\boxplus\ \mathtt{inr}\ x.\mathit{Buyer}\langle x\rangle)] \\ \mathit{Seller}(x) &:= x?(z).(\mathtt{inl}\ x.\mathtt{done}\ x \ {}_p\boxplus\ \mathtt{inr}\ x.x!\mathit{counteroffer}.\mathtt{case}\ x\ [\mathtt{idle}, \mathit{Seller}\langle x\rangle]) \end{aligned}$$

The buyer sends the current *bid* on x and waits for a reaction from the seller. The seller accepts the bid with probability p and rejects it with probability $1 - p$. If the seller accepts (by selecting the left branch of the session), the buyer terminates successfully. Otherwise, the seller proposes a counteroffer, which the buyer rejects with probability q and accepts with probability $1 - q$. In the first case, the session terminates without satisfaction of the buyer. In the second case, the buyer starts a new negotiation. ⌟

3 Probabilistic Session Types

Session types. Probabilistic session types describe communication protocols taking place through session endpoints and their (finite) syntax is given by the following grammar:

$$\textbf{Session type} \qquad T, S \ ::= \ \circ \ \mid \ \bullet \ \mid \ ?t.T \ \mid \ !t.T \ \mid \ T \ {}_p\&\ S \ \mid \ T \ {}_p\oplus\ S \tag{3.1}$$

The session types $\circ$ and $\bullet$ describe a session endpoint on which no further input/output operations are possible. We use $\bullet$ to mark those termination points of a protocol that represent success and that we target in our probabilistic analysis. The precise meaning of "successful termination" is domain specific but also irrelevant in the technical development that follows. The session types $?t.T$ and $!t.T$ describe session endpoints used for receiving (respectively, sending) a message of type t and then according to T. Types will be discussed shortly. The session types $T \ {}_p\&\ S$ and $T \ {}_p\oplus\ S$ describe a session endpoint used for receiving (respectively, sending) a binary choice which is "left" with probability p and "right" with probability $1 - p$. The endpoint is then used according to T or S, respectively. Note that ${}_p\oplus$ is an internal choice – the process behaving according to this type internally chooses either "left" or "right" – whereas ${}_p\&$ is an external choice – the process behaving according to this type externally offers behaviors corresponding to both choices. Therefore, the probability annotation in ${}_p\&$ is completely determined by the one in the corresponding internal choice and it could be argued that it is somewhat superfluous. Nonetheless, as we will see when discussing the typing rule for branch processes, having direct access to this annotation makes it easy to propagate the probability of choices across different sessions.

We do not use any special syntax for specifying infinite session types. Rather, we interpret the productions for T coinductively and we call session types the possibly infinite trees generated by the productions in (3.1) that satisfy the following conditions:

Regularity. We require every tree to consist of finitely many *distinct* subtrees. This condition ensures that session types are finitely representable either using the so-called "μ notation" [48] or as solutions of finite sets of equations [16].

Reachability. We require every subtree T of a session type to contain a *reachable leaf* labelled by $\circ$ or $\bullet$. This condition ensures that it is always possible to terminate a session regardless of how long it has been running.

To formalize these conditions, we define a relation $T \rightsquigarrow_p S$ modeling the fact that (the behavior described by) T may evolve into S with probability p in a single step:

$$\begin{array}{llll} \circ \rightsquigarrow_1 \circ & ?t.T \rightsquigarrow_1 T & T \;{}_p\&\; S \rightsquigarrow_p T & T \;{}_p\&\; S \rightsquigarrow_{1-p} S \\ \bullet \rightsquigarrow_1 \bullet & !t.T \rightsquigarrow_1 T & T \;{}_p\oplus\; S \rightsquigarrow_p T & T \;{}_p\oplus\; S \rightsquigarrow_{1-p} S \end{array}$$

We also consider the relation $\rightsquigarrow_p^*$, which accounts for multiple steps in the expected way:

$$T \rightsquigarrow_1^* T \qquad \frac{T \rightsquigarrow_p S}{T \rightsquigarrow_p^* S} \qquad \frac{T \rightsquigarrow_p^* T' \qquad T' \rightsquigarrow_q^* S}{T \rightsquigarrow_{pq}^* S}$$

Roughly speaking, $\rightsquigarrow_p^*$ is the reflexive, transitive closure of $\rightsquigarrow_p$ except that the probability annotation p accounts for the cumulative transition probability between two session types.

▶ **Definition 3.1** (well-formed session type)**.** *Let* $\mathcal{T}(T) \stackrel{\text{def}}{=} \{S \mid \exists p, S : T \rightsquigarrow_p^* S\}$*. A (possibly infinite) tree* T *generated by the productions in* (3.1) *is a* well-formed session type *if* $\mathcal{T}(T)$ *is finite and, for every* $S \in \mathcal{T}(T)$*, there exists* $p > 0$ *such that either* $S \rightsquigarrow_p^* \circ$ *or* $S \rightsquigarrow_p^* \bullet$*.*

▶ **Example 3.2** (auction protocol, buyer side)**.** Even though we have not presented the typing rules for the calculus of Section 2, we can speculate on the session type of the endpoint used *e.g.*, by the buyer process in Example 2.1, which satisfies the equation

$$T = !\mathsf{int}.(\bullet \;{}_p\&\; ?\mathsf{int}.(\circ \;{}_q\oplus\; T))$$

In this case we have $\mathcal{T}(T) = \{\circ, \bullet, \bullet \;{}_p\&\; (?\mathsf{int}.(\circ \;{}_q\oplus\; T)), ?\mathsf{int}.(\circ \;{}_q\oplus\; T), \circ \;{}_q\oplus\; T, T\}$ and it is easy to see that T is well formed provided that at least one among p and q is positive. ⌟

From now on we assume that all the session types we work with are well formed.

Success probability. We now define the probability that a protocol described by a session type T terminates successfully. Intuitively, this probability is computed by accounting for all paths in the structure of T that lead to a leaf labelled by $\bullet$. Formally:

▶ **Definition 3.3** (success probability)**.** *The* success probability *of a session type* T*, denoted by* $[\![T]\!]$*, is determined by the following equations:*

$$\begin{array}{lll} [\![\circ]\!] = 0 & [\![?t.T]\!] = [\![T]\!] & [\![T \;{}_p\&\; S]\!] = p[\![T]\!] + (1-p)[\![S]\!] \\ [\![\bullet]\!] = 1 & [\![!t.T]\!] = [\![T]\!] & [\![T \;{}_p\oplus\; S]\!] = p[\![T]\!] + (1-p)[\![S]\!] \end{array}$$

For a *finite* session type T, Definition 3.3 gives a straightforward recursive algorithm for computing $[\![T]\!]$. When T is infinite, however, it is less obvious that Definition 3.3 provides a way for determining $[\![T]\!]$. To address the problem in the general case we observe that, by interpreting $[\![T]\!]$ as a *probability variable*, Definition 3.3 allows us to derive a *finite system*

of equations relating such variables. Indeed, the right hand side of each equation for $[\![T]\!]$ in Definition 3.3 is expressed in terms of probability variables corresponding to the children nodes in the tree of T. Since T has finitely many subtrees, we end up with finitely many equations. Then, we observe that every session type T corresponds to a Discrete-Time Markov Chain (DTMC) [34, 49] whose state space is $\mathcal{T}(T) = \{S_1, \ldots, S_n\}$ and such that the probability p_{ij} of performing a transition from state S_i to state S_j is given by

$$p_{ij} \stackrel{\text{def}}{=} \begin{cases} p & \text{if } S_i \rightsquigarrow_p S_j \\ 0 & \text{otherwise} \end{cases}$$

Regularity and reachability imply that the DTMC we obtain from any session type T is finite state and absorbing. That is, it is always possible to reach an *absorbing state* (either $\circ$ or $\bullet$) from any *transient state* (any other session type). In any finite-state, absorbing DTMC, the probability of reaching a specific absorbing state from any transient state can be computed by solving a particular system of equations which is guaranteed to have a unique solution [34]. Moreover, the system that we obtain for $[\![T]\!]$ using Definition 3.3 is precisely the one whose solution is the probability of reaching $\bullet$ from T (see Appendix A).

▶ **Example 3.4.** We compute the success probability of T from Example 3.2 where, for the sake of illustration, we take $p = \frac{1}{4}$ and $q = \frac{2}{3}$. Let $T_1 = \bullet \;{}_{\frac{1}{4}}\&\; T_2$ and $T_2 = ?\mathsf{int}.T_3$ and $T_3 = \circ \;{}_{\frac{2}{3}}\oplus\; T$ be convenient names for some of its subtrees. Using Definition 3.3 we obtain the system of equations

$$\begin{array}{lll} [\![T]\!] = [\![T_1]\!] & [\![\bullet]\!] = 1 & [\![T_3]\!] = \frac{2}{3}[\![\circ]\!] + \frac{1}{3}[\![T]\!] \\ [\![T_1]\!] = \frac{1}{4}[\![\bullet]\!] + \frac{3}{4}[\![T_2]\!] & [\![T_2]\!] = [\![T_3]\!] & [\![\circ]\!] = 0 \end{array}$$

from which we compute $[\![T]\!] = \frac{1}{3}$ (Appendix A details the corresponding DTMC). ⌟

Duality. We write $\overline{T}$ for the *dual* of T, that is the session type obtained from T by swapping input actions with output actions and leaving the remaining forms unchanged. Formally, $\overline{T}$ is the session type obtained from T that satisfies the following equations:

$$\begin{array}{lll} \overline{\circ} = \circ & \overline{?t.T} = !t.\overline{T} & \overline{T \;{}_p\&\; S} = \overline{T} \;{}_p\oplus\; \overline{S} \\ \overline{\bullet} = \bullet & \overline{!t.T} = ?t.\overline{T} & \overline{T \;{}_p\oplus\; S} = \overline{T} \;{}_p\&\; \overline{S} \end{array}$$

It is easy to see that duality is an involution (that is, $\overline{\overline{T}} = T$) and that the success probability is unaffected by duality, that is $[\![T]\!] = [\![\overline{T}]\!]$. This means that we can compute the success probability of a session from either of its two endpoints.

Types. Types describe resources used by processes and exchanged as messages. We distinguish between session endpoints, whose type is a session type T, from sessions with success probability p, whose type has the form $\langle p \rangle$:

$$\textbf{Type} \qquad t, s \;::=\; T \;\mid\; \langle p \rangle \tag{3.2}$$

We will see in Section 4 that a type $\langle p \rangle$ results from "joining" the two peer endpoints of a session having dual sessions types T and $\overline{T}$ such that $p = [\![T]\!] = [\![\overline{T}]\!]$. For brevity we omit message types such as unit and int from the formal development as their handling is folklore and does not affect the presented results. We occasionally use them in the examples though.

A key aspect of the type system is that processes may use session endpoints differently, depending on the outcome of probabilistic choices. Nonetheless, we need to capture the overall effect of such different uses in a single type. For this reason, we introduce a *probabilistic type combinator* that allows us to combine types by weighing the different ways in which a resource is used according to a given probability.

▶ **Definition 3.5** (probabilistic type combination)**.** *We write* $t \ {}_p\boxplus\ s$ *for the* combination of t and s weighed by p*, which is defined by cases on the form of* t *and* s *as follows:*

$$t \ {}_p\boxplus\ s \stackrel{\text{def}}{=} \begin{cases} t & \text{if } t = s \\ T \ {}_{pq+(1-p)r}\oplus\ S & \text{if } t = T \ {}_q\oplus\ S \text{ and } s = T \ {}_r\oplus\ S \\ \langle pq + (1-p)r \rangle & \text{if } t = \langle q \rangle \text{ and } s = \langle r \rangle \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

Intuitively, $t \ {}_p\boxplus\ s$ describes a resource that is used according to t with probability p and according to s with probability $1 - p$. The combination of t and s is only defined when t and s have "compatible shapes", the trivial case being when they are the same type. The interesting cases are when t and s describe a choice (a point of the protocol where one process performs a selection) and when t and s describe a session as a whole. In both cases, the success probability of the choice (respectively, of the session) is weighed by p. As an example, consider a session endpoint that is used according to $T \ {}_1\oplus\ S$ with probability p and according to $T \ {}_0\oplus\ S$ with probability $1 - p$. In the first case, we are certain that the session endpoint is used for selecting "left" and then according to T. In the second case, we are certain that the session endpoint is used for selecting "right" and then according to S. Overall, the session endpoint is used according to the type $T \ {}_p\oplus\ S$.

The combination $\langle q \rangle \ {}_p\boxplus\ \langle r \rangle = \langle pq + (1-p)r \rangle$ captures the fact that the success probability of a whole session that is carried out in two different ways having success probabilities respectively q and r is the convex sum of q and r weighed by p. The success probability with which we annotate this type allows us to state the soundness properties of the type system, by relating the success probabilities in session types with those of a process that behaves according to those session types. Speaking of success probability, a fundamental property that is used extensively in the soundness proofs is the following one. Any conceivable generalization of Definition 3.5 must guarantee this property for the type system to be sound.

▶ **Proposition 3.6.** $[\![T_1 \ {}_p\boxplus\ T_2]\!] = p[\![T_1]\!] + (1-p)[\![T_2]\!]$.

Definition 3.5 is quite conservative in that, except for top-level choices, any other session type can only be combined with itself. It is conceivable to generalize ${}_p\boxplus$ to permit the combination of "deep choices" found after a common prefix. For example, we could have $!\mathsf{int}.(T \ {}_1\oplus\ S) \ {}_p\boxplus\ !\mathsf{int}.(T \ {}_0\oplus\ S) = !\mathsf{int}.(T \ {}_p\oplus\ S)$. This generalization is not for free, though. As we will see in Section 4, session endpoints that are affected by a probabilistic choice must be "handled with care" and Definition 3.5 as it stands helps ensuring that this is actually the case. We leave the combination of "deep choices" to future work.

4 Typing Rules

We use contexts for tracking the type of free variables occurring in processes. A *context* is a finite map from variables to types written $x_1 : t_1, \ldots, x_n : t_n$. We let Γ and Δ range over contexts, we write $\emptyset$ for the empty context, $\mathsf{dom}(\Gamma)$ for the domain of Γ and Γ, Δ for the union of Γ and Δ when $\mathsf{dom}(\Gamma) \cap \mathsf{dom}(\Delta) = \emptyset$. We also extend ${}_p\boxplus$ pointwise to contexts in the obvious way.

Table 3 Typing rules.

$$\textsc{T-IDLE}\ \frac{\mathsf{un}(\Gamma)}{\Gamma \vdash \mathtt{idle}} \qquad \textsc{T-DONE}\ \frac{\mathsf{un}(\Gamma)}{\Gamma, x : \bullet \vdash \mathtt{done}\, x} \qquad \textsc{T-VAR}\ \frac{\mathsf{un}(\Gamma) \qquad A : \tilde{t} \qquad \mathsf{safe}(\tilde{t})}{\Gamma, \overline{x : t} \vdash A\langle \tilde{x} \rangle}$$

$$\textsc{T-IN}\ \frac{\Gamma, x : T, y : t \vdash P}{\Gamma, x : ?t.T \vdash x?(y).P} \qquad \textsc{T-BRANCH}\ \frac{\Gamma, x : T \vdash P \qquad \Delta, x : S \vdash Q}{\Gamma\ {}_p\boxplus\ \Delta, x : T\ {}_p\&\ S \vdash \mathtt{case}\, x\, [P, Q]}$$

$$\textsc{T-OUT}\ \frac{\Gamma, x : T \vdash P \qquad \mathsf{safe}(t)}{\Gamma, x : !t.T, y : t \vdash x!y.P} \qquad \textsc{T-LEFT}\ \frac{\Gamma, x : T \vdash P}{\Gamma, x : T\ {}_1\oplus\ S \vdash \mathtt{inl}\, x.P} \qquad \textsc{T-RIGHT}\ \frac{\Gamma, x : S \vdash P}{\Gamma, x : T\ {}_0\oplus\ S \vdash \mathtt{inr}\, x.P}$$

$$\textsc{T-PAR}\ \frac{\Gamma, x : T \vdash P \qquad \Delta, x : \overline{T} \vdash Q}{\Gamma, \Delta, x : \langle [\![T]\!] \rangle \vdash P \mid Q} \qquad \textsc{T-CHOICE}\ \frac{\Gamma \vdash P \qquad \Delta \vdash Q}{\Gamma\ {}_p\boxplus\ \Delta \vdash P\ {}_p\boxplus\ Q} \qquad \textsc{T-NEW}\ \frac{\Gamma, x : \langle p \rangle \vdash P}{\Gamma \vdash (x)P}$$

Before we discuss the typing rules, we have to introduce two predicates to single out types that have particular properties. The class of *unrestricted types*, defined next, is aimed at describing resources that can be discarded and duplicated at will.

▶ **Definition 4.1** (unrestricted type and context). *We say that t is* unrestricted *and we write* $\mathsf{un}(t)$ *if* $t = \circ$. *We write* $\mathsf{un}(\Gamma)$ *if* $\mathsf{un}(\Gamma(x))$ *for all* $x \in \mathsf{dom}(\Gamma)$.

In our case, the only unrestricted type is $\circ$, but if the type system is extended with basic types such as unit and int, these would be unrestricted as well. Next we introduce the class of *safe types*, those describing resources that can be safely sent in messages and used in process invocations because they cannot be passively affected by a probabilistic choice.

▶ **Definition 4.2** (safe type). *We write* $\mathsf{safe}(t)$ *if* t *is* not *of the form* $T\ {}_p\&\ S$.

The ultimate motivation for the safety predicate has its roots in the soundness proof of the type system. In a nutshell, an unsafe session type is one whose dual admits a non-trivial probabilistic combination (Definition 3.5) and therefore that may change unpredictably, from the standpoint of a process using a resource with that (unsafe) type. In this case, the process must wait to be notified of the (probabilistic) choice that has occurred before using the resource in a message. Should the need arise to send an unsafe endpoint in a message, it is possible to patch the endpoint's session type so as to make it safe, for example by prefixing the session type with a dummy input/output action. We will see an instance where this patch is necessary in Example 4.12.

Judgments have the form $\Gamma \vdash P$, meaning that P is well typed in Γ, and are derived by the rules in Table 3. We assume a global map from process variables to sequences of types written $\{A_i : \tilde{t}_i\}_{i \in I}$ whose domain includes all the process variables for which there is a definition $A_i(\tilde{x}) := P_i$ and that $\overline{x : t} \vdash P_i$ is derivable for every $i \in I$. This ensures that all process definitions are typed consistently. The typing rules are syntax directed, so that each process form corresponds to a typing rule. We now discuss each rule in detail. Rules T-IDLE and T-DONE deal with terminated processes. In T-IDLE the whole context must be unrestricted, since the idle process does not use any resource. Rule T-DONE is similar, except that the session x flagged by the process must have type $\bullet$. This way, we enforce the correspondence between successful termination in processes and successful termination in

session types. Rule T-VAR establishes that a process invocation is well typed provided that the type of the parameters passed to the process match the expected ones and that any unused resource has an unrestricted type. The premise $A : \overline{t}$ indicates that A is associated with the sequence of types $\overline{t}$ in the global map, ensuring that A is invoked with parameters of the right type. Observe that the type of such parameters must be safe. This way, we prevent to use as parameters resources whose type can be (passively) affected by a probabilistic choice. Rules T-IN and T-OUT deal with the exchange of a message y on session x. The rules update the type of x from the conclusion to the premise of the rule to account for the communication. As usual, a linear resource y being sent in a message is no longer available in the continuation of the process. As anticipated earlier, T-OUT requires the type of y to be safe, again to ensure that the type of y does not suddenly change under the effect of a probabilistic choice.

The typing rules described so far are fairly standard for any session calculus. We now move on to the part of the type system that handles probabilities. Rules T-LEFT and T-RIGHT deal with selections. In these cases, the type of x must be of the form $T \,{}_p\!\oplus\, S$ and the process continuation uses x according to either T or S respectively. The key aspect is the probability p with which the process selects "left", which is 1 in the case of $\texttt{inl}\, x$ and 0 in the case of $\texttt{inr}\, x$. These processes behave deterministically, hence the probability annotation in the session type is trivial. Rule T-BRANCH illustrates the typing of a branch, whereby a process receives a choice from a session x and continues accordingly. The type of x must be of the form $T \,{}_p\!\&\, S$, where p is the probability with which the process will receive a "left" choice. The key part of the rule concerns *all the other resources* used by the process, which will be used according to Γ if the process receives a "left" choice and according to Δ otherwise. That is, the process is becoming aware of a probabilistic choice that has been performed elsewhere and whose outcome is communicated on x. Depending on this information, the process uses its resources (not just x) accordingly. The behavior of the process as a whole is described by the combination $\Gamma \,{}_p\!\boxplus\, \Delta$ of the contexts in the two branches. Recall that the ${}_p\!\boxplus$ operator, when used on session types, is idempotent in all cases but for selections (Definition 3.5). Hence, $\Gamma \,{}_p\!\boxplus\, \Delta$ is *nearly the same* as Γ and Δ, except that the probabilities with which some future selections will be performed on endpoints in Γ and Δ may have been adjusted as a side effect of the information received from x. This mechanism enables the propagation of probabilistic choices through the system as messages are exchanged on sessions.

Rule T-PAR deals with parallel compositions $P \mid Q$, where P and Q must use x according to dual session types. Writing Γ, Δ in the conclusion of the rule ensures that P and Q do not share any name other than x, thus preventing the creation of network topologies that may lead to deadlocks [12]. In the conclusion of the rule the type of x becomes of the form $\langle p \rangle$ to record the fact that both endpoints of x have been used. The success probability p coincides with that of one of the endpoints and is well defined since $[\![T]\!] = [\![\overline{T}]\!]$. Rule T-CHOICE deals with probabilistic choices performed by a process and partially overlaps with T-BRANCH in that the contexts of the two alternative evolutions of the process after the choice are combined by ${}_p\!\boxplus$. Finally, rule T-NEW removes a session x from the context when x is restricted.

Let us now discuss the main properties enjoyed by well-typed processes. First and foremost, typing is preserved by reductions.

▶ **Theorem 4.3** (subject reduction). *If* $\Gamma \vdash P$ *and* $P \to Q$*, then* $\Gamma \vdash Q$*.*

Although this result is considered standard, one detail makes it special in our setting. Specifically, we observe that the reduct Q is well typed in the *very same environment* used for typing P, despite the fact that a communication may have taken place on a session x in P, determining a change in the session types associated with the endpoints of x. A communication can occur only if P contains *both* endpoints for x, and more precisely if there

are two subprocesses of P that use x according to dual session types and that are composed in parallel using T-PAR. Then, x in Γ must be associated with a type of the form $\langle p \rangle$, where p is the success probability of P. Then, Theorem 4.3 guarantees that not only the typing, but also the *success probability of sessions is preserved by reductions*. This is counterintuitive at first, given that a session may evolve through different branches each having different success probabilities. However, recall that probabilistic choices are *persistent* in our calculus, meaning that the reduct Q accounts for *all possible evolutions* of P. This is what entails such strong formulation of Theorem 4.3.

Next we turn our attention to termination. To this aim, we provide two characterizations of process termination respectively concerning the present and the future states of a process.

▶ **Definition 4.4** (immediate and eventual termination). *We say that P is* terminated *if $P\downarrow$ is derivable using the following axioms and rules:*

$$\texttt{idle}\downarrow \qquad \texttt{done}\, x\downarrow \qquad \frac{P\downarrow \qquad Q\downarrow}{P \mid Q\downarrow} \qquad \frac{P\downarrow \qquad Q\downarrow}{P \;{}_p\!\boxplus\, Q\downarrow} \qquad \frac{P\downarrow}{(x)P\downarrow}$$

We say that P terminates with probability p*, notation $P \Downarrow_p$, if there exist (P_n), (Q_n) and (p_n) for $n \in \mathbb{N}$ such that $P \Rightarrow P_n \;{}_{p_n}\!\boxplus\, Q_n$ and $P_n\downarrow$ for every $n \in \mathbb{N}$ and $\lim_{n\to\infty} p_n = p$.*

In words, $P\downarrow$ means that P does not contain any pending communications, whereas $P \Downarrow_p$ means that P evolves with probability p to states in which there are no pending communications. Our type system is not strong enough to guarantee (probable) termination. For example, the process Ω defined by $\Omega := \Omega$ is well-typed and diverges. In general, however, well-typed processes are guaranteed to be deadlock free, as stated formally below.

▶ **Theorem 4.5** (deadlock freedom). *If $\emptyset \vdash P$ and $P \Rightarrow Q$, then either $Q \to$ or $Q\downarrow$.*

Note that deadlock freedom is not simply a bonus feature of our type system. It is actually a requirement for proving the properties of the type system that specifically pertain probabilities, which we will discuss shortly. Before doing so, we need an operational characterization of successful termination relative to a particular session.

▶ **Definition 4.6** (successful termination of a session). *We say that P* successfully terminates session x with probability p *if $P\uparrow^x_p$ is derivable using the following axioms and rules:*

$$\textit{P-DONE}\ \frac{}{\texttt{done}\, x\uparrow^x_1} \qquad \textit{P-PAR-1}\ \frac{P\uparrow^x_p}{P \mid Q\uparrow^x_p} \qquad \textit{P-PAR-2}\ \frac{Q\uparrow^x_p}{P \mid Q\uparrow^x_p} \qquad \textit{P-RES}\ \frac{P\uparrow^x_p \qquad x \neq y}{(y)P\uparrow^x_p}$$

$$\textit{P-CHOICE}\ \frac{P\uparrow^x_q \qquad Q\uparrow^x_r}{P \;{}_p\!\boxplus\, Q\uparrow^x_{pq+(1-p)r}} \qquad \textit{P-ANY}\ \frac{}{P\uparrow^x_0}$$

Axiom P-DONE states that a process of the form $\texttt{done}\, x$ has successfully terminated session x with probability 1. The rules P-PAR-i state that the successful termination of a parallel composition $P \mid Q$ with respect to a session x can be reduced to the successful termination of either P or Q. In particular, we do not require that *both* P and Q have successfully terminated x, for two reasons: first, it could be the case that P and Q are connected by a session different from x, hence only one among P and Q could own x; second, if a process has successfully terminated a session through one of its endpoints, then duality ensures that the peer owning the other endpoint cannot have pending operations on it, so the session as a whole can be considered successfully terminated even if only one peer has become $\texttt{done}\, x$.

Rule P-RES accounts for session restrictions in the expected way and P-CHOICE states that the successful termination of x in a process distribution is obtained by weighing the probabilities of successful termination of the processes in the distribution. Note that P-CHOICE

can be applied only if it is possible to derive the successful termination of x for *all* of the processes in the distribution, whereas in general only *some* of such processes will have successfully terminated x. To account for this possibility, we can use P-ANY to *approximate* the probability of successful termination of x for any process to 0.

The type system gives us an upper bound to the success probability of any session:

▶ **Proposition 4.7.** *If $x : \langle p \rangle \vdash P$ and $P \uparrow^x_q$, then $q \leq p$.*

In particular, a session with type $\langle 0 \rangle$ cannot be successfully completed, which could indicate a flaw in the system. The upper bound is matched exactly by terminated processes:

▶ **Theorem 4.8.** *If $x : \langle p \rangle \vdash P$ and $P \not\to$, then $P \uparrow^x_p$.*

Note that Theorem 4.8 does not hold unless processes are deadlock free, whence the key role of Theorem 4.5. As stated, Theorem 4.8 appears of limited use since it only concerns processes that cannot reduce any further, whereas in general we are interested in computing the probability of successful termination also for processes engaged in arbitrarily long interactions, for which the predicate $P \not\to$ might never hold. It turns out that Theorem 4.8 can be relativized to the probability that a process terminates, thus:

▶ **Corollary 4.9** (relative success). *Let $P \Uparrow^x_p$ if there exist (P_n) and (p_n) such that $P \Rightarrow P_n$ and $P_n \uparrow^x_{p_n}$ for all $n \in \mathbb{N}$ and $\lim_{n\to\infty} p_n = p$. Then (1) $x : \langle 1 \rangle \vdash P$ and $P \Downarrow_p$ imply $P \Uparrow^x_p$ and (2) $x : \langle p \rangle \vdash P$ and $P \Downarrow_1$ imply $P \Uparrow^x_p$.*

Property (1) states that a well-typed process using a session with type $x : \langle 1 \rangle$ successfully completes the session with the same probability with which it terminates. Property (2) extends Theorem 4.8 to processes that are known to terminate with probability 1.

▶ **Example 4.10.** Below is the type derivation for the process *Buyer* from Example 2.1 using T from Example 3.2 and assuming the type assignment $Buyer : T$.

$$\dfrac{\dfrac{}{x : \bullet \vdash \mathtt{done}\, x}\,\text{T-DONE} \quad \dfrac{\dfrac{\dfrac{}{x : \circ \vdash \mathtt{idle}}\,\text{T-IDLE}}{x : \circ \,{}_1\!\oplus T \vdash \mathtt{inl}\, x}\,\text{T-LEFT} \quad \dfrac{\dfrac{}{x : T, y : \mathsf{int} \vdash \mathit{Buyer}\langle x \rangle}\,\text{T-VAR}}{x : \circ \,{}_0\!\oplus T, y : \mathsf{int} \vdash \mathtt{inr}\, x.\mathit{Buyer}\langle x \rangle}\,\text{T-RIGHT}}{x : \circ \,{}_q\!\oplus T, y : \mathsf{int} \vdash \mathtt{inl}\, x \,{}_q\!\boxplus\, \mathtt{inr}\, x.\mathit{Buyer}\langle x \rangle}\,\text{T-CHOICE}}{x : ?\mathsf{int}.(\circ \,{}_q\!\oplus T) \vdash x?(y).(\mathtt{inl}\, x \,{}_q\!\boxplus\, \mathtt{inr}\, x.\mathit{Buyer}\langle x \rangle)}\,\text{T-IN}}{x : \bullet \,{}_p\!\&\, ?\mathsf{int}.(\circ \,{}_q\!\oplus T) \vdash \mathtt{case}\, x\, [\mathtt{done}\, x, x?(y).(\mathtt{inl}\, x \,{}_q\!\boxplus\, \mathtt{inr}\, x.\mathit{Buyer}\langle x \rangle)]}\,\text{T-BRANCH}}{x : T \vdash x!\mathit{bid}.\mathtt{case}\, x\, [\mathtt{done}\, x, x?(y).(\mathtt{inl}\, x \,{}_p\!\boxplus\, \mathtt{inr}\, x.\mathit{Buyer}\langle x \rangle)]}\,\text{T-OUT}$$

Observe the application of T-CHOICE, which turns the probabilistic choice $\circ \,{}_q\!\oplus T$ in the conclusion of the rule into a deterministic one in the two premises. There exists an analogous derivation for $x : \overline{T} \vdash Q$ where Q is the body of $\mathit{Seller}\langle x \rangle$ in Example 2.1. By taking p and q as in Example 3.4, we derive $x : \langle \frac{1}{3} \rangle \vdash \mathit{Buyer}\langle x \rangle \mid \mathit{Seller}\langle x \rangle$ with one application of T-PAR. It is easy to establish that this process terminates with probability 1, hence by Corollary 4.9(2) the buyer wins the auction with probability $\frac{1}{3}$. ⌟

▶ **Example 4.11.** The separation of probabilistic choices from the communication of information ("left" and "right" selections) that depends on such choices implies that there is no 1-to-1 correspondence between choices as seen in session types and choices performed by processes. Below are a few instances in which the type system performs a non-trivial reconciliation between the probability annotations in types and those in processes. The type derivations are detailed in [31].

1. The process $\mathtt{case}\, x\, [\mathtt{inr}\, y.\mathtt{done}\, x, \mathtt{inl}\, y.\mathtt{done}\, y]$ inverts a choice from session x to y, so that it successfully completes x if and only if it does not successfully complete y. It is well typed in the context $x : \bullet\ {}_{p}\&\ \circ, y : \bullet\ {}_{1-p}\oplus\ \circ$, which reflects the effect of the inversion.
2. The process $\mathtt{case}\, x\, [\mathtt{case}\, y\, [\mathtt{inl}\, z.\mathtt{done}\, z, \mathtt{inr}\, z], \mathtt{case}\, y\, [\mathtt{inr}\, z, \mathtt{inr}\, z]]$ coalesces two choices received from x and y into a choice sent on z. The process is well typed in the context $x : \circ\ {}_{p}\&\ \circ, y : \circ\ {}_{q}\&\ \circ, z : \bullet\ {}_{pq}\oplus\ \circ$, indicating that the success probability for z is the product of the probabilities of receiving "left" from both x and y.
3. The process $\mathtt{inl}\, x.\mathtt{inl}\, x.\mathtt{done}\, x\ {}_{\frac{1}{2}}\boxplus\ \mathtt{inr}\, x.\mathtt{inr}\, x$ sends the same probabilistic choice twice on session x. It is well typed in the context $x : (\bullet\ {}_{1}\oplus\ \circ)\ {}_{\frac{1}{2}}\oplus\ (\circ\ {}_{0}\oplus\ \circ)$ but *not* in the context $x : (\bullet\ {}_{\frac{1}{2}}\oplus\ \circ)\ {}_{\frac{1}{2}}\oplus\ (\circ\ {}_{\frac{1}{2}}\oplus\ \circ)$. Once the choice is communicated, subsequent "left" or "right" selections that depend on that choice become deterministic. ⌟

▶ **Example 4.12** (Work sharing). Consider a system $C\langle x\rangle \mid x?(z).B\langle x, y, z\rangle \mid A\langle y\rangle$ modeling (from left to right) a master process C connected with two slave processes which can be "busy" handling jobs or "idle" waiting for jobs. The processes are defined as follows:

$$\begin{aligned} C(x) &:= x!job.\mathtt{case}\, x\, [\mathtt{done}\, x, \mathtt{idle}] \\ B(x, y, job) &:= y!\langle\rangle.\left(\mathtt{inl}\, x.\mathtt{inl}\, y.\mathtt{done}\, x\ {}_{p}\boxplus\ \left(\mathtt{inr}\, x.\mathtt{inl}\, y\ {}_{q}\boxplus\ \mathtt{inr}\, y.y!x.y!job.A\langle y\rangle\right)\right) \\ A(y) &:= y?().\mathtt{case}\, y\, [\mathtt{idle}, y?(x).y?(z).B\langle x, y, z\rangle] \end{aligned}$$

The master sends a job to the first slave and waits for a notification indicating whether the job has been handled or not. Obviously, the master succeeds only in the first case. A busy slave decides whether to handle the job (with probability p) or not (with probability $1-p$). In the first case, it notifies the master and the idle slave that the job has been handled and terminates. In the second case, it decides whether to discard the job (with probability q) or to hand it over to the other slave (with probability $1-q$). Note that the busy slave sends on y a dummy value to the idle one before taking any decision so that the type of y is *safe* when y is used in $A\langle y\rangle$. This way, by the time the busy slave makes a probabilistic choice that may affect (and will be communicated to) the idle slave, the idle slave is blocked on a $\mathtt{case}$ waiting for such choice, and therefore its typing can be suitably adjusted when it is moved (by S-PAR-CHOICE) into the scope of the choice.

Now, take $T = !\mathsf{unit}.(\circ\ {}_{p-pq+q}\oplus\ !S.!\mathsf{int}.\overline{T})$ and $S = \bullet\ {}_{r}\oplus\ \circ$ where $\max\{p, q\} > 0$ and $r = \frac{p}{p-pq+q}$. It is possible to show that the above composition is well typed under the global type assignments $C : !\mathsf{int}.\overline{S}$, $B : S, T, \mathsf{int}$ and $A : \overline{T}$, where we assume that *job* has type int. From the fact that the system terminates with probability 1, we conclude that the master succeeds with probability r. Details can be found in [31]. ⌟

5 Related Work

Type systems for probabilistic, concurrent programs. Despite their close relationship with process algebras, many of which have been extensively studied in a probabilistic setting, there are few results concerning probabilistic variants of session types. A notable exception is [2], which considers a probabilistic variant of multiparty session types (MST) where global types are decorated by ranges of probabilities representing the degree of likelihood for interactions to happen. Besides using MST while we use binary session types, a key difference is that [2] does not consider interleaved sessions. The effect of probabilistic choices across different sessions and the type system presented therein ensures that the aggregate probability of all execution paths is 1, which in our case is guaranteed by the semantics of the probabilistic choice operator in processes. The type system in [2] essentially checks that each choice

in a process is made according to the probability range written in its type, i.e., a process chooses a branch with a probability value that lies within the range specified by its session type. Differently, a probability value in our types does not necessarily translate into the same probability value in a process; moreover, the same probabilistic choice in a process may be reflected as different probabilities in different sessions, as illustrated in Example 4.11.

Some type systems for probabilistic programs have been developed to characterize precisely the space of the possible execution traces [39] or to ensure that well-typed programs do not leak secret information [17]. The work [54] considers a sub-structural type system for a probabilistic variant of the linear π-calculus. Although the type system is not concerned with probabilities directly, there are interesting analogies with our typing discipline: it is only by relying on the properties of well-typed processes – most notably, race and deadlock freedom – that we are able to relate the probabilities in processes with those in types.

Probabilistic models of concurrent processes. The design of computational models that combine concurrency and probabilities has a long tradition [55, 50] and gave birth to a variety of operational approaches [51] and concrete probabilistic extensions of well-known concurrency models, such as CCS [27], CSP [41, 25], Petri nets [9], Klaim [19], and name-passing process calculi [28, 54, 43, 26]. Our language for processes can be seen as the session-based counterpart of (a synchronous version of) the *simple probabilistic π-calculus* [43], which features both probabilistic and non-deterministic choices. While non-deterministic choices in [43] correspond to the standard choice operator ($+$) of the π-calculus, we adopted a session discipline, and hence a choice is realised by communicating a label over a session.

The development of a denotational semantics for languages that combine non-determinism, concurrency and probabilities has revealed challenging. On the one hand, probabilistic choices do not distribute over non-deterministic ones, *i.e.*, it matters whether the environment chooses before or after a probabilistic choice is made, as highlighted in [53]. This observation appears to be reflected in our type system by the typing rules that require a term to be of a safe type, *e.g.*, when a session is delegated. Establishing a precise connection between these two notions may pave the way for generalisations of our probabilistic type combinator. On the other hand, probabilistic choices in a system need to be (probabilistically) independent. This problem is connected with the well-known *confusion phenomenon*, in which concurrent (and hence, independent) choices may influence each other (*e.g.*, one choice may enable/disable some branch in another choice). As shown in [1, 33, 11], confusion can be avoided by establishing an order in which choices are executed; essentially, by reducing concurrency. We remark that the session discipline imposed by our language – and rule T-PAR in particular – makes all probabilistic choices independent (in a probabilistic sense).

Probabilistic languages and analyses. Probabilistic models are frequently used to prove properties that can be expressed as reachability probabilities; they are then verified by model-checking [32]. Our types are also reachability properties related to the probability of successful completion of a session. Besides, our type system guarantees deadlock-freedom. Many approaches have been recently proposed for reasoning on probabilistic programs, *e.g.*, deductive-style approaches based on separation logic [6, 52, 5], probabilistic strategy logic [3], proof of termination [23, 37], static analysis [56], and probabilistic symbolic execution [10]. Typing has been used in the sequential setting to ensure almost-sure termination in a probabilistic lambda calculus [35]. Our type system does not ensure termination, but it could form the basis for a probabilistic termination analysis.

Deadlock-free sessions. The technique we use for preventing deadlocks, which only addresses tree-like network topologies, is directly inspired to logic-based session type systems [12]. However, our probabilitic analysis is independent of the exact mechanism that enforces deadlock freedom and applies to other type systems relying on richer type structures [46, 18].

6 Concluding Remarks

In this work we start the study of a type-based static analysis technique for reasoning on probabilistic reachability problems in session-based systems. We relate a probabilistic variant of a session-based calculus (Section 2) with a probabilistic variant of binary session types (Section 3) and establish a correspondence between probability annotations in processes and those in types (Section 4). By breaking down a complex system of communicating processes into sessions, we are able to modularly infer properties concerning the (probable) evolution of the system from the much simpler specifications described by session types.

There are many developments that stem from this work addressing both technical and practical problems. Here we discuss those looking more promising or intriguing.

To make our approach practical, the type system must be supported by suitable type checking and inference algorithms. Indeed, even though the typing rules are syntax directed, the probabilistic type combinator (Definition 3.5) is difficult to deal with because it is not injective (the same type can result from combining types with different probability annotations). We are also considering extensions of the very same operator so that it is applicable to "deep choices" that do not necessarily occur at the top level of a session type. This extension requires a careful balancing with the notion of type safety (Definition 4.2).

Subtyping relations for session types [24] are important for addressing realistic programming scenarios. Given the already established connections between session subtyping and (fair) testing relations [36, 13, 45, 8, 47] and the extensive literature on probabilistic testing relations [14, 44, 21, 20] and behavioral equivalences [40], the investigation of probabilistic variants of session subtyping has solid grounds to build upon. A related problem is that process models that feature both non-deterministic and probabilistic choices are known to be difficult to model and analyze [21]. It could be the case that session-based systems with both non-deterministic and probabilistic choices are easier to address thanks to their simpler structure, as already observed in [54].

Our analysis based on probabilistic session types can be extended in several ways. For example, it would be interesting to quantify the probability of (partial) execution traces rather than (or in addition to) the reachability of "successful states". As remarked in Section 3, reachability ensures uniqueness of solutions of the systems of equations induced by Definition 3.3, but it could be interesting to analyse the spectra of solutions obtained when reachability is dropped. One could also study variants where probabilities are allowed to vary during the execution. For instance, one would like to analyse recursive protocols where probabilities may decrease (or increase) at each iteration. In our setting this may spoil regularity (subtrees may be decorated with infinitely many probabilities), allowing one to give non finitary specifications. A possible way of tackling this problem is to allow imprecise probabilities in the types; this may retain regularity at the cost of a coarser static analysis. Probability ranges could also be useful in those cases where probability annotations in processes are uncertain, possibly because they have been estimated from execution traces [22]. Besides probabilities, there might be other methods suitable to model the uncertainty behind the behavior of processes. Further approaches include the "possibilistic" one, where uncertainty is described using linguistic categories with fuzzy boundaries [57],

information gap decision theory, where the impact of uncertain parameters is estimated by the deviation of errors [7], and interval analysis, where uncertain parameters are modelled as intervals and worst-case analysis is usually performed [42]. We think that probability annotations in session types may also support forms of static analysis aimed at quantifying the termination probability of session-based programs. Known type systems that ensure progress, deadlock and livelock freedom are often quite constraining on the structure of well-typed programs [46, 15, 4]. It could be the case that switching to a probabilistic setting broadens substantially the range of addressable programs.

References

1 Samy Abbes and Albert Benveniste. True-concurrency probabilistic models: Branching cells and distributed probabilities for event structures. *Information and Computation*, 204(2):231–274, 2006. doi:10.1016/j.ic.2005.10.001.

2 Bogdan Aman and Gabriel Ciobanu. Probabilities in session types. In Mircea Marin and Adrian Craciun, editors, *Proceedings Third Symposium on Working Formal Methods, FROM 2019, Timişoara, Romania, 3-5 September 2019*, volume 303 of *EPTCS*, pages 92–106, 2019. doi:10.4204/EPTCS.303.7.

3 Benjamin Aminof, Marta Kwiatkowska, Bastien Maubert, Aniello Murano, and Sasha Rubin. Probabilistic strategy logic. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 32–38, 2019. doi:10.24963/ijcai.2019/5.

4 Stephanie Balzer, Bernardo Toninho, and Frank Pfenning. Manifest deadlock-freedom for shared session types. In *Proceedings of the European Symposium on Programming Languages (ESOP)*, volume 11423, pages 611–639. Springer, 2019. doi:10.1007/978-3-030-17184-1_22.

5 Gilles Barthe, Justin Hsu, and Kevin Liao. A probabilistic separation logic. *Proc. ACM Program. Lang.*, 4(POPL):55:1–55:30, 2020. doi:10.1145/3371123.

6 Kevin Batz, Benjamin Lucien Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Thomas Noll. Quantitative separation logic: a logic for reasoning about probabilistic pointer programs. *Proc. ACM Program. Lang.*, 3(POPL):34:1–34:29, 2019. doi:10.1145/3290347.

7 Yakov Ben-Haim. *Info-gap decision theory: decisions under severe uncertainty*. Academic Press, 2006.

8 Giovanni Bernardi and Matthew Hennessy. Using higher-order contracts to model session types. *Logical Methods in Computer Science*, 12(2), 2016. doi:10.2168/LMCS-12(2:10)2016.

9 Rémi Bonnet, Stefan Kiefer, and Anthony Widjaja Lin. Analysis of probabilistic basic parallel processes. In *Proceedings of the International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 8412, pages 43–57. Springer, 2014. doi:10.1007/978-3-642-54830-7_3.

10 Mateus Borges, Antonio Filieri, Marcelo d'Amorim, and Corina S. Pasareanu. Iterative distribution-aware sampling for probabilistic symbolic execution. In *Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*, pages 866–877, 2015. doi:10.1145/2786805.2786832.

11 Roberto Bruni, Hernán C. Melgratti, and Ugo Montanari. Concurrency and probability: Removing confusion, compositionally. *Log. Methods Comput. Sci.*, 15(4), 2019. doi:10.23638/LMCS-15(4:17)2019.

12 Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Math. Struct. Comput. Sci.*, 26(3):367–423, 2016. doi:10.1017/S0960129514000218.

13 Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, Elena Giachino, and Luca Padovani. Foundations of session types. In António Porto and Francisco Javier López-Fraguas, editors, *Proceedings of the International Conference on Principles and Practice of Declarative Programming (PPDP)*, pages 219–230. ACM, 2009. doi:10.1145/1599410.1599437.

14 Rance Cleaveland, Zeynep Dayar, Scott A. Smolka, and Shoji Yuen. Testing preorders for probabilistic processes. *Inf. Comput.*, 154(2):93–148, 1999. doi:10.1006/inco.1999.2808.

15 Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. Global progress for dynamically interleaved multiparty sessions. *Math. Struct. Comput. Sci.*, 26(2):238–302, 2016. doi:10.1017/S0960129514000188.

16 Bruno Courcelle. Fundamental properties of infinite trees. *Theor. Comput. Sci.*, 25:95–169, 1983. doi:10.1016/0304-3975(83)90059-2.

17 David Darais, Ian Sweet, Chang Liu, and Michael Hicks. A language for probabilistically oblivious computation. *Proc. ACM Program. Lang.*, 4(Proceedings of the ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)):50:1–50:31, 2020. doi:10.1145/3371118.

18 Ornela Dardha and Simon J. Gay. A new linear logic for deadlock-free session-typed processes. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10803 of *Lecture Notes in Computer Science*, pages 91–109. Springer, 2018. doi:10.1007/978-3-319-89366-2_5.

19 Rocco De Nicola, Diego Latella, and Mieke Massink. Formal modeling and quantitative analysis of klaim-based mobile systems. In *Proceedings of the ACM symposium on Applied computing (SAC)*, pages 428–435, 2005. doi:10.1145/1066677.1066777.

20 Yuxin Deng, Rob Van Glabbeek, Matthew Hennessy, and Carroll Morgan. Testing finitary probabilistic processes. In *Proceedings of the International Conference on Concurrency Theory (CONCUR)*, pages 274–288. Springer, 2009. doi:10.1007/978-3-642-04081-8_19.

21 Yuxin Deng, Rob J. van Glabbeek, Matthew Hennessy, Carroll Morgan, and Chenyi Zhang. Remarks on testing probabilistic processes. *Electron. Notes Theor. Comput. Sci.*, 172:359–397, 2007. doi:10.1016/j.entcs.2007.02.013.

22 Seyedeh Sepideh Emam and James Miller. Inferring extended probabilistic finite-state automaton models from software executions. *ACM Trans. Softw. Eng. Methodol.*, 27(1):4:1–4:39, 2018. doi:10.1145/3196883.

23 Luis María Ferrer Fioriti and Holger Hermanns. Probabilistic termination: Soundness, completeness, and compositionality. In *Proceedings of the ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pages 489–501. ACM, 2015. doi:10.1145/2775051.2677001.

24 Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Inf.*, 42(2-3):191–225, 2005. doi:10.1007/s00236-005-0177-z.

25 Sonja Georgievska and Suzana Andova. Probabilistic CSP: preserving the laws via restricted schedulers. In *Proceedings of the International GI/ITG Conference on Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance (MMB/DFT)*, volume 7201, pages 136–150. Springer, 2012. doi:10.1007/978-3-642-28540-0_10.

26 Jean Goubault-Larrecq, Catuscia Palamidessi, and Angelo Troina. A probabilistic applied pi-calculus. In Zhong Shao, editor, *Programming Languages and Systems, 5th Asian Symposium, APLAS 2007, Singapore, November 29-December 1, 2007, Proceedings*, volume 4807 of *Lecture Notes in Computer Science*, pages 175–190. Springer, 2007. doi:10.1007/978-3-540-76637-7_12.

27 Hans A. Hansson. Time and probabilities in specification and verification of real-time systems. In *Proceedings of the Euromicro workshop on Real-Time Systems (RTS)*, pages 92–97, 1992. doi:10.1109/EMWRT.1992.637477.

28 Oltea Mihaela Herescu and Catuscia Palamidessi. Probabilistic asynchronous π-calculus. In *Proceedings of the International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 1784, pages 146–160. Springer, 2000. doi:10.1007/3-540-46432-8_10.

29 Kohei Honda. Types for dyadic interaction. In Eike Best, editor, *Proceedings of the International Conference on Concurrency Theory (CONCUR)*, volume 715, pages 509–523. Springer, 1993. doi:10.1007/3-540-57208-2_35.

30 Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira, and Gianluigi Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016. doi:10.1145/2873052.

31 Omar Inverso, Hernán Melgratti, Luca Padovani, Catia Trubiani, and Emilio Tuosto. Probabilistic Analysis of Binary Sessions. Technical report, Università di Torino, 2020. doi:10.5281/zenodo.3951070.

32 Joost-Pieter Katoen. The probabilistic model checking landscape. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 31–45. ACM, 2016. doi:10.1145/2933575.2934574.

33 Joost-Pieter Katoen and Doron A. Peled. Taming confusion for modeling and implementing probabilistic concurrent systems. In Matthias Felleisen and Philippa Gardner, editors, *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7792 of *Lecture Notes in Computer Science*, pages 411–430. Springer, 2013. doi:10.1007/978-3-642-37036-6_23.

34 John G. Kemeny and J. Laurie Snell. *Finite Markov Chains*. Springer-Verlag, 1976.

35 Ugo Dal Lago and Charles Grellois. Probabilistic termination by monadic affine sized typing. *ACM Trans. Program. Lang. Syst.*, 41(2):10:1–10:65, 2019. doi:10.1145/3293605.

36 Cosimo Laneve and Luca Padovani. The pairing of contracts and session types. In *Concurrency, Graphs and Models, Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*, volume 5065, pages 681–700. Springer, 2008. doi:10.1007/978-3-540-68679-8_42.

37 Ondrej Lengál, Anthony Widjaja Lin, Rupak Majumdar, and Philipp Rümmer. Fair termination for parameterized probabilistic concurrent systems. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 10205, pages 499–517, 2017. doi:10.1007/978-3-662-54577-5_29.

38 Thomas Leventis. A deterministic rewrite system for the probabilistic λ-calculus. *Math. Struct. Comput. Sci.*, 29(10):1479–1512, 2019. doi:10.1017/S0960129519000045.

39 Alexander K. Lew, Marco F. Cusumano-Towner, Benjamin Sherman, Michael Carbin, and Vikash K. Mansinghka. Trace types and denotational semantics for sound programmable inference in probabilistic languages. *Proc. ACM Program. Lang.*, 4(POPL):19:1–19:32, 2020. doi:10.1145/3371087.

40 Natalia López and Manuel Núñez. An overview of probabilistic process algebras and their equivalences. In *Validation of Stochastic Systems - A Guide to Current Research*, volume 2925, pages 89–123. Springer, 2004. doi:10.1007/978-3-540-24611-4_3.

41 Gavin Lowe. Probabilistic and prioritized models of timed CSP. *Theor. Comput. Sci.*, 138(2):315–352, 1995. doi:10.1016/0304-3975(94)00171-E.

42 Ramon E. Moore, R. Baker Kearfott, and Michael J. Cloud. *Introduction to Interval Analysis*. SIAM, 2009. doi:10.1137/1.9780898717716.

43 Gethin Norman, Catuscia Palamidessi, David Parker, and Peng Wu. Model checking the probabilistic pi-calculus. In *Fourth International Conference on the Quantitative Evaluaiton of Systems (QEST 2007), 17-19 September 2007, Edinburgh, Scotland, UK*, pages 169–178. IEEE Computer Society, 2007. doi:10.1109/QEST.2007.31.

44 Manuel Núñez and David Rupérez. Fair testing through probabilistic testing. In *Proceedings of the Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification (PSTV)*, volume 156, pages 135–150. Kluwer, 1999. doi:10.1007/978-0-387-35578-8_8.

45 Luca Padovani. Fair subtyping for open session types. In *Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 7966, pages 373–384. Springer, 2013. doi:10.1007/978-3-642-39212-2_34.

46 Luca Padovani. Deadlock and lock freedom in the linear π-calculus. In *Proceedings of the Joint Meeting of the EACSL Annual Conference on Computer Science Logic and the Annual ACM/IEEE Symposium on Logic in Computer Science (CSL-LICS)*, pages 72:1–72:10. ACM, 2014. doi:10.1145/2603088.2603116.

47 Luca Padovani. Fair subtyping for multi-party session types. *Math. Struct. Comput. Sci.*, 26(3):424–464, 2016. doi:10.1017/S096012951400022X.

48 Benjamin C. Pierce. *Types and programming languages*. MIT Press, 2002.

49 Jan J. M. M. Rutten, Marta Z. Kwiatkowska, Gethin Norman, David Parker, and Prakash Panangaden. *Mathematical techniques for analyzing concurrent and probabilistic systems*, volume 23 of *CRM monograph series*. American Mathematical Society, 2004.

50 Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

51 Ana Sokolova and Erik P. de Vink. Probabilistic automata: System types, parallel composition and comparison. In Christel Baier, Boudewijn R. Haverkort, Holger Hermanns, Joost-Pieter Katoen, and Markus Siegle, editors, *Validation of Stochastic Systems - A Guide to Current Research*, volume 2925 of *Lecture Notes in Computer Science*, pages 1–43. Springer, 2004. doi:10.1007/978-3-540-24611-4_1.

52 Joseph Tassarotti and Robert Harper. A separation logic for concurrent randomized programs. *Proc. ACM Program. Lang.*, 3(POPL):64:1–64:30, 2019. doi:10.1145/3290377.

53 Daniele Varacca and Glynn Winskel. Distributing probability over non-determinism. *Math. Struct. Comput. Sci.*, 16(1):87–113, 2006. doi:10.1017/S0960129505005074.

54 Daniele Varacca and Nobuko Yoshida. Probabilistic π-calculus and event structures. *Electronic Notes in Theoretical Computer Science*, 190(3):147–166, 2007. doi:10.1016/j.entcs.2007.07.009.

55 Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 327–338. IEEE Computer Society, 1985. doi:10.1109/SFCS.1985.12.

56 Di Wang, Jan Hoffmann, and Thomas W. Reps. PMAF: an algebraic framework for static analysis of probabilistic programs. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pages 513–528, 2018. doi:10.1145/3296979.3192408.

57 Lotfi A. Zadeh. Fuzzy sets. *Inf. Control.*, 8(3):338–353, 1965. doi:10.1016/S0019-9958(65)90241-X.

A Supplement to Section 3

▶ **Example A.1.** Consider the type T in Example 3.4. The transition matrix $P = [p_{ij}]$ of its associated DTMC is shown below:

$$P = \left[\begin{array}{cc|cccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 \\ \frac{1}{4} & 0 & 0 & 0 & \frac{3}{4} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & \frac{2}{3} & \frac{1}{3} & 0 & 0 & 0 \end{array}\right] \quad \textit{where} \quad \begin{array}{l} S_0 = \bullet \\ S_1 = \circ \\ S_2 = T \\ S_3 = \bullet \ {}_{\frac{1}{4}}\& \ ?\mathsf{int}.(\circ \ {}_{\frac{2}{3}}\oplus T) \\ S_4 = ?\mathsf{int}.(\circ \ {}_{\frac{2}{3}}\oplus T) \\ S_5 = \circ \ {}_{\frac{2}{3}}\oplus T \end{array}$$

Note that we have given P in its *canonical form* [34], in which we have partitioned P in four submatrices with the names and meaning described below in clockwise order, starting from the top-left corner of P:

- S is the 2-by-2 identity matrix giving the probability transitions among the absorbing states. By definition of absorbing state, this is an identity matrix.

- O is the 2-by-4 matrix giving the probability transitions from the absorbing states to the transient states. By definition, these probabilities are all zeros.
- Q is the 4-by-4 matrix giving the probability transitions among the transient states.
- R is the 4-by-2 matrix giving the probability transitions from the transient states to the absorbing states.

Now, the probability of S_2 being absorbed by S_1, *i.e.*, $[\![T]\!]$, can be obtained from the matrix $B = [b_{ij}]$ which is computed as follows:

$$\begin{aligned} B &= (I-Q)^{-1}R \\ &= \begin{bmatrix} 1 & -1 & 0 & 0 \\ 0 & 1 & -\frac{3}{4} & 0 \\ 0 & 0 & 1 & -1 \\ -\frac{1}{3} & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 0 \\ \frac{1}{4} & 0 \\ 0 & 0 \\ 0 & \frac{2}{3} \end{bmatrix} = \begin{bmatrix} \frac{4}{3} & \frac{4}{3} & 1 & 1 \\ \frac{1}{3} & \frac{4}{3} & 1 & 1 \\ \frac{4}{9} & \frac{4}{9} & \frac{4}{3} & \frac{4}{3} \\ \frac{4}{9} & \frac{4}{9} & \frac{1}{3} & \frac{4}{3} \end{bmatrix} \begin{bmatrix} 0 & 0 \\ \frac{1}{4} & 0 \\ 0 & 0 \\ 0 & \frac{2}{3} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} \\ \frac{1}{3} & \frac{2}{3} \\ \frac{1}{9} & \frac{8}{9} \\ \frac{1}{9} & \frac{8}{9} \end{bmatrix} \end{aligned}$$

Then, the probability of absorption for $S_2 = T$ is b_{00}. Hence, $[\![T]\!] = \frac{1}{3}$. ⌟

▶ **Theorem A.2** ([34]). *Let P be the transition matrix of an absorbing DTMC and B^* be the matrix of the absorption probabilities. Then, $PB^* = B^*$.*

Note that the column l of B^*, *i.e.*, $[b_{il}]$ contains the probabilities of s_i being absorbed by s_l. Consequently, $b_{ll} = 1$ and $b_{il} = 0$ for all absorbing states $s_i \neq s_l$. Also, the probability b_{il} for non-absorbing states s_i can be obtained by solving the system of linear equations corresponding to l-column of B^* in the equality $B^* = PB^*$, *i.e.*,

$$\begin{aligned} b_{ll} &= 1 \\ b_{ii} &= 0 && \text{for all absorbing states } s_i \neq s_l \\ b_{il} &= \textstyle\sum_h p_{ih} \times b_{hl} && \text{for all non-absorbing states } s_i \end{aligned}$$

When considering the DTMCs associated with session types there are exactly two absorbing states, namely $\bullet$ and $\circ$. Moreover, we are interested in computing the column in B^* associated with $\bullet$. If we write $[\![S_i]\!]$ in place of $b_i l$ when $S_l = \bullet$, then the set of linear equations is

$$\begin{aligned} [\![\bullet]\!] &= 1 \\ [\![\circ]\!] &= 0 \\ [\![S_i]\!] &= \textstyle\sum_h p_{ih} \times [\![S_h]\!] && \text{for all } S_i \notin \{\circ, \bullet\} \end{aligned}$$

▶ **Example A.3.** The system of equations for the DTMC in Example A.1 is

$$\begin{aligned} [\![\bullet]\!] &= 1 \\ [\![\circ]\!] &= 0 \\ [\![T]\!] &= [\![S_3]\!] \\ [\![S_3]\!] &= \tfrac{1}{4}[\![\bullet]\!] + \tfrac{3}{4}[\![S_4]\!] \\ [\![S_4]\!] &= [\![S_5]\!] \\ [\![S_5]\!] &= \tfrac{2}{3}[\![\circ]\!] + \tfrac{1}{3}[\![T]\!] \end{aligned}$$

Note in particular that the system of equations corresponds exactly to the one derived from Definition 3.3 and its solution is $[\![T]\!] = \frac{1}{3}$, $[\![S_3]\!] = \frac{1}{3}$, $[\![S_5]\!] = \frac{1}{9}$, $[\![S_5]\!] = \frac{1}{9}$. ⌟

We conclude this section with the proof of Proposition 3.6.

▶ **Proposition 3.6.** $[\![T_1 \ {}_p\boxplus \ T_2]\!] = p[\![T_1]\!] + (1-p)[\![T_2]\!]$.

Proof. The only interesting case is when $T_1 = T \,{}_q\!\oplus S$ and $T_2 = T \,{}_r\!\oplus S$. We have

$$
\begin{aligned}
&[\![T_1 \,{}_p\!\boxplus T_2]\!] \\
&= [\![(T \,{}_q\!\oplus S) \,{}_p\!\boxplus (T \,{}_r\!\oplus S)]\!] && \text{by definition of } T_1 \text{ and } T_2 \\
&= [\![T \,{}_{pq+(1-p)r}\!\oplus S]\!] && \text{by definition of } {}_p\!\boxplus \\
&= (pq + (1-p)r)[\![T]\!] + (1 - pq - (1-p)r)[\![S]\!] && \text{by definition of } [\![\cdot]\!] \\
&= pq[\![T]\!] + r[\![T]\!] - pr[\![T]\!] + [\![S]\!] - pq[\![S]\!] - r[\![S]\!] + pr[\![S]\!]
\end{aligned}
$$

$$
\begin{aligned}
&p[\![T_1]\!] + (1-p)[\![T_2]\!] \\
&= p[\![T \,{}_q\!\oplus S]\!] + (1-p)[\![T \,{}_r\!\oplus S]\!] && \text{by definition of } T_1 \text{ and } T_2 \\
&= p(q[\![T]\!] + (1-q)[\![S]\!]) + (1-p)(r[\![T]\!] + (1-r)[\![S]\!]) && \text{by definition of } [\![\cdot]\!] \\
&= \; pq[\![T]\!] + p[\![S]\!] - pq[\![S]\!] + r[\![T]\!] + [\![S]\!] - r[\![S]\!] - pr[\![T]\!] - p[\![S]\!] + pr[\![S]\!] \\
&= \; pq[\![T]\!] + r[\![T]\!] - pr[\![T]\!] + [\![S]\!] - pq[\![S]\!] - r[\![S]\!] + pr[\![S]\!]
\end{aligned}
$$

which confirms the statement. ◀

On Ranking Function Synthesis and Termination for Polynomial Programs

Eike Neumann
Department of Computer Science, Oxford University, UK
eike.neumann@cs.ox.ac.uk

Joël Ouaknine
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany
Department of Computer Science, Oxford University, UK
joel@mpi-sws.org

James Worrell
Department of Computer Science, Oxford University, UK
jbw@cs.ox.ac.uk

Abstract

We consider the problem of synthesising polynomial ranking functions for single-path loops over the reals with continuous semi-algebraic update function and compact semi-algebraic guard set. We show that a loop of this form has a polynomial ranking function if and only if it terminates. We further show that termination is decidable for such loops in the special case where the update function is affine.

2012 ACM Subject Classification Theory of computation → Semantics and reasoning

Keywords and phrases Semi-algebraic sets, Polynomial ranking functions, Polynomial programs

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.15

Funding *Joël Ouaknine*: Supported by ERC grant AVS-ISS (648701) and DFG grant 389792660 as part of TRR 248 (see `https://perspicuous-computing.science`).
James Worrell: Supported by EPSRC Fellowship EP/N008197/1.

1 Introduction

The method of proving program termination through ranking functions is one of the oldest and most fundamental ideas in computer science. More recently, the idea of automatically synthesising ranking functions has emerged as an important topic in automated verification and program analysis. Particular attention has focussed on linear ranking functions. Indeed for simple programs, such as linear constraint loops, there are complete methods for synthesising linear ranking functions: such methods find a ranking function whenever one exists, typically by reduction to linear and integer programming [16]. We refer to survey [2] for a thorough discussion of the extensive literature on this topic. More expressive generalisations of linear ranking functions include lexicographic linear ranking functions [3, 9], multiphase linear ranking functions [4], and piecewise linear ranking functions [18].

The advent of powerful techniques for solving non-linear constraints has led to another direction generalising linear ranking functions, namely polynomial ranking functions. Semi-definite programming was used in [10] to synthesise polynomial ranking functions on polynomial loops, while [8] uses cylindrical algebraic decomposition. More recently, polynomial ranking functions [7] have been used to prove termination of probabilistic programs.

In this paper we consider the problem of synthesising polynomial ranking functions for *semi-algebraic loops*: single-path loops over the reals in which the update computed by the loop body is a continuous semi-algebraic function and the loop guard is a semi-algebraic set.

Since we allow polynomials of arbitrary degree as ranking functions, the search for a ranking function cannot immediately be reduced to a constraint satisfaction problem. Our main result shows that if the guard set is compact then a semi-algebraic loop admits a polynomial ranking function if and only if it terminates. The assumption of compactness is essential here: it is straightforward to give examples of terminating loops with non-compact guard that admit no polynomial ranking function. Nevertheless, the class of programs we consider is highly non-trivial. Indeed, to the best of our knowledge, the termination problem for semi-algebraic loops with compact guard set is open (see the discussion below), hence the equivalent problem of deciding the existence of polynomial ranking functions is likewise open. Our main result illustrates the utility and generality of polynomial ranking functions, and offers a potential approach to resolve the decidability of termination for the class of loops in question.

As a second contribution, we show that in the case of semi-algebraic loops in which the guard is compact and the update map is affine, non-termination is equivalent to the existence of a polynomial invariant for the update that is contained in the guard set, and hence decidable. For comparison, recall that Tiwari [17] has given a method to decide termination over the reals of loops with convex polyhedral guard sets and linear update functions (i.e., the same loop dynamics and an incomparable class of loop guards). The case of termination of linear loops with general polyhedral guards (i.e., potentially neither compact nor convex) remains open to the best of our knowledge.

Related Work. It is shown in [1, Lemma 12] and [14, last paragraph in the proof of Theorem 8] that a linear constraint loop with compact polyhedral guard terminates if and only if it has a linear ranking function. This is similar in form to our main result, but involves an incomparable class of transition relations in which non-determinism is allowed but all constraints must be linear.

A partial decidability result for termination of single-path polynomial loops with compact connected guard sets is given in [13]. Under certain semantic assumptions on the loop, this procedure reduces the problem of deciding termination to that of finding fixed points of polynomial maps. Another partial decidability result is [11], which considers a syntactic subclass of polynomial loop programs called *triangular weakly non-linear* and reduces the termination problem to the decision problem for the existential theory of real-closed fields.

For loops with linear updates and convex linear guard conditions, termination is known to be decidable when program variables respectively range over $\mathbb{R}$, $\mathbb{Q}$, and $\mathbb{Z}$ – see [6, 12, 17]. In [19] decidability is shown for loops whose guard sets can be expressed as conjunctions of polynomial inequalities and whose updates are linear functions which satisfy an additional technical condition called the "non-zero minimum property".

2 The main theorem

Recall that $K \subseteq \mathbb{R}^n$ is said to be *semi-algebraic* if it is definable by a Boolean combination of inequalities $f(x_1, \ldots, x_n) < 0$ and $f(x_1, \ldots, x_n) \leq 0$ with $f \in \mathbb{Z}[x_1, \ldots, x_n]$. A function $g : \mathbb{R}^m \to \mathbb{R}^n$ is moreover said to be *semi-algebraic* if its graph is a semi-algebraic set. Our main result concerns the existence of ranking functions for single-path loops with compact semi-algebraic guard set and semi-algebraic update map:

▶ **Theorem 1.** *Let $g \colon \mathbb{R}^n \to \mathbb{R}^n$ be a continuous semi-algebraic map. Let $K \subseteq \mathbb{R}^n$ be a compact semi-algebraic set. If for all $x \in K$ there exists an $m \in \mathbb{N}$ such that $g^m(x) \notin K$ then there exists a polynomial ranking function for g on K, i.e., a polynomial $f \in \mathbb{R}[x_1, \ldots, x_n]$ satisfying $f(x) - f(g(x)) \geq 1$ and $f(x) \geq 0$ for all $x \in K$.*

Since the existence of a ranking function clearly implies termination, Theorem 1 shows that such a semi-algebraic loop terminates precisely when it has a polynomial ranking function. The theorem clearly fails if we remove the requirement that K be closed: for loop guard $K = (0, 1)$ and transition function $g(x) = 2x$ there cannot be a polynomial ranking function since the time to escape K under the action of g is not bounded from above. Likewise the theorem fails in case K is closed but not bounded. For example, consider $K = \{(x, y) \in \mathbb{R}^2 : x \geq 0, y \geq 1\}$ and $g(x, y) = \left(x - \frac{1}{y}, y + 1\right)$. The loop is terminating since $\sum_{y=1}^{\infty} \frac{1}{y}$ diverges, but from $\sum_{y=1}^{n} \frac{1}{y} < \ln n + 1$ one sees that the time to termination from $(m, 1)$ is at least e^{m-1}, which precludes the existence of a polynomial ranking function.

The proof of Theorem 1 relies on several lemmas. In the rest of this section we give an informal overview of the proof structure, formulate the key lemmas, and finally show how they imply the main result.

According to the Stone-Weierstraß theorem, a continuous function on K can be uniformly approximated arbitrarily closely by polynomials, and hence it suffices to find a continuous function $f\colon K \cup g(K) \to \mathbb{R}$ satisfying $f(x) - f(g(x)) \geq 1$ for all $x \in K$. We show the stronger assertion that there exists a continuous function f satisfying $f(x) = f(g(x)) + 1$ for all $x \in K$.

The first step associates with the loop update function g an injective "covering map" $\widetilde{g}$. Specifically, we construct a space T and continuous surjective map $p : T \to K \cup g(K)$ such that there is $S \subseteq T$ and an injective semi-algebraic map $\widetilde{g}\colon S \to T$ making the following diagram commute:

$$\begin{array}{ccc} S & \xrightarrow{\widetilde{g}} & T \\ \downarrow{\scriptstyle p\restriction_S} & & \downarrow{\scriptstyle p} \\ K & \xrightarrow{g} & K \cup g(K) \end{array}$$

We then construct a continuous function $\widetilde{f}\colon T \to \mathbb{R}$ that satisfies the functional equation $\widetilde{f}(x) = \widetilde{f}(\widetilde{g}(x)) + 1$ for all $x \in S$ and is constant on each fibre $p^{-1}(x)$, $x \in K \cup g(K)$. To construct $\widetilde{f}$ we find a partition of T into finitely many semi-algebraic pieces $S_1, \ldots, S_m$ such that (i) the boundary of a piece is contained in a union of pieces of lower dimension, (ii) S is a union of pieces, (iii) each piece $S_i \subseteq S$ is mapped by $\widetilde{g}$ onto another piece S_j. We order the pieces of a given dimension by the transitive closure of the relation $S_i = \widetilde{g}(S_j)$. This is a well-founded partial ordering thanks to the termination assumption – that for all $x \in K$ there exists m such that $g^m(x) \notin K$. Ignoring for now the technical requirement that $\widetilde{f}$ be constant on the fibres of p, we can essentially construct $\widetilde{f}$ as follows: On the zero-dimensional pieces S_i which are minimal with respect to this ordering let $\widetilde{f}(S_i) = 0$. Extend $\widetilde{f}$ to the remaining zero-dimensional pieces using the functional equation $\widetilde{f}(x) = \widetilde{f}(\widetilde{g}(x)) + 1$. Now $\widetilde{f}$ can be defined on the minimal one-dimensional pieces by interpolating the boundary values. It can be extended to all one-dimensional pieces using the functional equation. Continuing with the higher-dimensional pieces in the same manner, we eventually obtain a continuous solution to the functional equation $\widetilde{f}(x) = \widetilde{f}(\widetilde{g}(x)) + 1$ on all of T. We will also be able to ensure that it is constant on the fibres of the projection p, so that we obtain a continuous solution to the equation $f(x) = f(g(x)) + 1$ on $K \cup g(K)$ by letting $f(x) = \widetilde{f}(\widetilde{x})$ where $\widetilde{x}$ is any point in T satisfying $p(\widetilde{x}) = x$.

As a preliminary result we observe that if the iteration of a continuous function escapes a compact set then the number of iterations required to escape is bounded independently of the starting point.

▶ **Proposition 2.** *Let $g\colon \mathbb{R}^n \to \mathbb{R}^n$ be a continuous map. Let $K \subseteq \mathbb{R}^n$ be a compact set. Assume that for every $x \in K$ there exists a positive integer m such that $g^m(x) \notin K$. Then there exists a positive integer M such that for all $x \in K$ there exists $i \leq M$ such that $g^i(x) \notin K$.*

Proof. For every positive integer $m \in \mathbb{N}$ let $B_m = \{x \in \mathbb{R}^n \mid g^m(x) \notin K\}$. Since K is closed and g is continuous, each of the sets B_m is open. The assumption says that every $x \in K$ is contained in B_m for some m. In other words, the family $(B_m)_{m\in\mathbb{N}}$ is an open cover for K. Since K is compact this cover has a finite subcover $B_{m_1}, \ldots, B_{m_s}$. Now take $M = \max\{m_1, \ldots, m_s\}$. ◀

Let us now state the required lemmas more formally. The first lemma concerns the existence of a suitable covering space, which allows us to replace g with an injective map.

▶ **Lemma 3.** *Let $g\colon \mathbb{R}^n \to \mathbb{R}^n$ be a continuous semi-algebraic map. Let $K \subseteq \mathbb{R}^n$ be a compact semi-algebraic set. If for all $x \in K$ there exists $m \in \mathbb{N}$ such that $g^m(x) \notin K$ then there exist a compact semi-algebraic set $T \subseteq \mathbb{R}^N$, a continuous semi-algebraic surjection $p\colon T \to K \cup g(K)$, and an injective continuous semi-algebraic map $\widetilde{g}\colon p^{-1}(K) \to T$ such that the following diagram commutes:*

$$\begin{array}{ccc} p^{-1}(K) & \xrightarrow{\widetilde{g}} & T \\ \downarrow{\scriptstyle p} & & \downarrow{\scriptstyle p} \\ K & \xrightarrow{g} & K \cup g(K) \end{array}$$

Proof. By Proposition 2 there exists a number $M \geq 0$ such that for all $x \in K$ there exists $i \leq M$ such that $g^i(x) \notin K$. Let

$$H = \left\{(i, x, g(x), g^2(x), \ldots, g^i(x), 0, \ldots, 0) \in \mathbb{R}^{n(M+2)+1} \mid i \in \mathbb{N}, i \leq M+1, x \in K\right\}.$$

Then H is clearly a compact semi-algebraic set.

Let

$$p\colon H \to \mathbb{R}^n,\ p(i, x, g(x), \ldots, g^i(x), 0, \ldots, 0) = g^i(x).$$

Let $T = p^{-1}(K \cup g(K)) \subseteq H$ and $S = p^{-1}(K)$. Then S and T are compact semi-algebraic subsets of H and p maps S onto K and T onto $K \cup g(K)$ (as we allow $i = 0$ in the definition of H).

Let

$$\widetilde{g}\colon S \to T,\ \widetilde{g}(i, x, g(x), g^2(x), \ldots, g^i(x), 0, \ldots, 0) = (i{+}1, x, g(x), g^2(x), \ldots, g^{i+1}(x), 0, \ldots, 0).$$

Then $\widetilde{g}$ is a continuous, injective semi-algebraic map. It satisfies the equation $p{\circ}\widetilde{g}(x) = g{\circ}p(x)$ for all $x \in S$. ◀

We will construct a ranking function for $\widetilde{g}$ that is constant on fibres of the projection. This will be achieved through the construction and refinement of certain semi-algebraic cellular decompositions of the codomain T of $\widetilde{g}$. For the definition of semi-algebraic cellular decomposition compare, e.g., [5, Definition 9.1.11].

▶ **Definition 4.** *A subset $C \subseteq \mathbb{R}^n$ is a* cell of dimension $d \in \mathbb{N}$ *if there is a semi-algebraic homeomorphism $\Phi\colon D \to C$ where D is the open disk in $\mathbb{R}^d$. Given a closed semi-algebraic set $S \subseteq \mathbb{R}^n$, a* semi-algebraic cellular decomposition *of S is a finite partition of S into cells such that for each cell C its closure in $\mathbb{R}^n$ is the union of C and a collection of other cells of strictly smaller dimension than C.*

Based on the above, we introduce the notion of an *invariant semi-algebraic stratification*:

▶ **Definition 5.** *Let $g\colon S \to T$ be a semi-algebraic map between semi-algebraic sets S and T, with $S \subseteq T$ and T closed. A* g-invariant semi-algebraic stratification *of T consists of a partition of T into finitely many semi-algebraic sets $S_1, \ldots, S_m$, called* strata*, together with a semi-algebraic cellular decomposition of T, such that*

1. *Each stratum is a finite union of cells, homeomorpic to an open disk.*
2. *The set S is a finite union of strata.*
3. *For all strata $S_i \subseteq S$, the set $g(S_i)$ is a stratum.*

The *boundary* of a stratum S is defined to be $\mathrm{cl}\,(S) \setminus S$. Note that this is different from the topological boundary of S regarded as a subset of $\mathbb{R}^n$. Since every stratum S is a finite union of cells, and the closure of every cell is also a union of cells, the boundary of S is a finite union of cells.

The next two lemmas constitute the core of the proof of Theorem 1. They will be proved in Sections 3 and 4 respectively.

▶ **Lemma 6.** *Let $g\colon S \to T$ be an injective continuous semi-algebraic map between compact semi-algebraic sets S and T with $S \subseteq T$. Assume that for all $x \in S$ there exists an integer $i \geq 0$ such that $g^i(x) \notin S$. Then there exists a g-invariant semi-algebraic stratification of T.*

▶ **Lemma 7.** *Let $g\colon \mathbb{R}^n \to \mathbb{R}^n$ be a continuous semi-algebraic map. Let $K \subseteq \mathbb{R}^n$ be a compact semi-algebraic set such that for all $x \in K$ there exists $m \in \mathbb{N}$ such that $g^m(x) \notin K$. Let $T \subseteq \mathbb{R}^N$ be a compact semi-algebraic set such that there exists a continuous semi-algebraic surjection $p\colon T \to K \cup g(K)$ and an injective continuous semi-algebraic map $\widetilde{g}\colon p^{-1}(K) \to T$ such that the following diagram commutes:*

$$\begin{array}{ccc} p^{-1}(K) & \xrightarrow{\widetilde{g}} & T \\ \downarrow{\scriptstyle p} & & \downarrow{\scriptstyle p} \\ K & \xrightarrow{g} & K \cup g(K) \end{array}$$

Assume that there exists a $\widetilde{g}$-invariant stratification of T. Then there exists a continuous function $\widetilde{f}\colon T \to \mathbb{R}$ satisfying the functional equation $\widetilde{f}(x) = \widetilde{f}(\widetilde{g}(x)) + 1$ for all $x \in p^{-1}(K)$ which is constant on the fibres of p.

Assuming the lemmas above we can now prove Theorem 1:

Proof of Theorem 1. We will prove that there exists a continuous function $f\colon K \cup g(K) \to \mathbb{R}$ which satisfies the functional equation $f(x) = f(g(x)) + 1$ for all $x \in K$. This suffices to prove the claim, for by the Stone-Weierstraß theorem there exists a polynomial $p \in \mathbb{R}[x_1, \ldots, x_n]$ satisfying $|p(x) - f(x)| < \frac{1}{4}$ for all $x \in K$. Let $c = \min\{p(x) \mid x \in K\}$. Then the polynomial $h(x) = 2(p(x) - c)$ satisfies $h(x) - h(g(x)) \geq 1$ and $h(x) \geq 0$ for all $x \in K$.

By Lemma 3 there exist compact semi-algebraic sets $p^{-1}(K) = S \subseteq T \subseteq \mathbb{R}^N$, a continuous semi-algebraic surjection $p\colon T \to K \cup g(K)$, which restricts to a surjection $p\colon S \to K$, and an injective continuous semi-algebraic map $\widetilde{g}\colon S \to T$ such that $p \circ \widetilde{g} = g \circ p$. By Lemma 6 there exists a $\widetilde{g}$-invariant stratification of T. By Lemma 7 there exists a continuous function $\widetilde{f}\colon T \to \mathbb{R}$ which satisfies $\widetilde{f}(x) = \widetilde{f}(\widetilde{g}(x)) + 1$ for all $x \in S$ and is constant on fibres of p.

Define on $K \cup g(K)$ the function $f(x) = \widetilde{f}(\widetilde{x})$ where $\widetilde{x}$ is any point in the fibre of x under p. This function is well-defined since $\widetilde{f}$ is constant on fibres. We have for all $x \in K$:

$$f(x) = \widetilde{f}(\widetilde{x}) = \widetilde{f}(\widetilde{g}(\widetilde{x})) + 1 = f(g(x)) + 1.$$

The last equality uses the fact that $p \circ \widetilde{g} = g \circ p$, so that if $p(\widetilde{x}) = x$ then $p(\widetilde{g}(\widetilde{x})) = g(p(\widetilde{x})) = g(x)$.

Finally, f is continuous. Observe that we have $f^{-1}(A) = p(\widetilde{f}^{-1}(A))$ for all sets $A \subseteq \mathbb{R}$. If A is closed, then $\widetilde{f}^{-1}(A)$ is compact, so that $p(\widetilde{f}^{-1}(A))$ is closed. Hence the preimage of any closed set under f is closed, so that f is continuous. ◀

3 Proof of Lemma 6

In this section we prove Lemma 6. Throughout this section S and T are compact semi-algebraic subsets of $\mathbb{R}^n$ with $S \subseteq T$, and $g\colon S \to T$ is a continuous injective semi-algebraic map such that for all $x \in S$ there exists an integer $i \geq 0$ such that $g^i(x) \notin S$.

We show that we can find a g-invariant stratification of T, in the sense of Definition 5. The construction relies on the following standard result in real-algebraic geometry on the existence of cellular decompositions (see, e.g., [5, Proposition 9.1.12]):

▶ **Theorem 8.** *Let $S \subseteq \mathbb{R}^n$ be a compact semi-algebraic set. Let $S_1, \ldots, S_m$ be semi-algebraic subsets of S. Then S admits a semi-algebraic cellular decomposition such that each S_i is a finite union of cells.*

We have a semi-algebraic function

$$\operatorname{rank}\colon T \to \mathbb{N},\ \operatorname{rank}(x) = \min\left\{i \in \mathbb{N} \mid g^i(x) \notin S\right\}.$$

We can decompose T into finitely many semi-algebraic subsets

$$E_i = \left\{x \in T \mid g^i(x) \notin S \text{ and } g^j(x) \in S \text{ for all } j < i\right\}.$$

On these sets the function rank is constant. By Theorem 8 the set T admits a semi-algebraic cellular decomposition such that each E_i is a finite union of cells. Then rank is constant on every cell of this cellular decomposition, so that we can assign to every cell e the number $\operatorname{rank}(e)$.

Let every cell of this decomposition be a stratum. We will keep refining this stratification until g maps strata onto strata.

A stratum A is called an *injury* if it is not mapped by g onto another stratum, *i.e.*, there does not exist another stratum A' such that $g(A) = A'$. If A is an injury which is mapped into another stratum, *i.e.*, there exists another stratum A' such that $g(A) \subseteq A'$, then A is called en injury of the second kind. Otherwise it is called an injury of the first kind.

The *signature* of an injury A is the tuple $(\operatorname{kind} A, \dim A, \operatorname{rank} A)$. The *injury signature* of a stratification of T is the multiset of all signatures of all injuries. Thus, if n different injuries have the same signature, then the injury signature of the stratification contains n copies of this signature.

▶ **Lemma 9.** *If the stratification of T contains an injury of the first kind, then the decomposition can be refined such that the new injury signature is the old injury signature with one signature of the form $(1, m, k)$ removed and finitely many signatures added of the form $(2, l, k)$ with $l \leq m$. Additionally, the subdivision may result in any number of injuries of the second kind of rank $k + 1$ and dimension $\leq m$ becoming injuries of the first kind.*

Proof. Let A be an injury of the first kind with dimension m and rank k.

The sets of the form $g^{-1}(C) \cap A$ where C is a stratum are semi-algebraic subsets of $\operatorname{cl}(A)$. By Theorem 8 there exists a semi-algebraic cellular decomposition of $\operatorname{cl}(A)$ such that each set of the form $g^{-1}(C) \cap A$ and every cell of the old cellular decomposition of $\operatorname{cl}(A)$ is a finite union of cells. Replace the cellular decomposition on $\operatorname{cl}(A)$ with this new decomposition.

This defines a new cellular decomposition of T that refines the old one. Remove A from the stratification, and add all cells of the refined decomposition which are contained in A as new strata.

The newly added strata get mapped into strata by construction, but they may still be injuries of the second kind. Since they are contained in A they have the same rank as A and their dimension is at most $\dim A$. Strata that were mapped into but not onto A may no longer be mapped into strata in the new stratification, so that they become injuries of the first kind. Their rank is equal to $\operatorname{rank} A + 1$. Since g is injective their dimension is at most $\dim A$.

It may be the case that there is a stratum C that was mapped onto A. Since g is injective there is at most one such stratum. Its closure $\operatorname{cl}(C)$ is mapped by g onto $\operatorname{cl}(A)$. Its dimension is the same as the dimension of A and its rank is $\operatorname{rank} A + 1$. This stratum would become an injury of the first kind in the new stratification. To remove this injury we proceed as follows: Apply Theorem 8 to obtain a new cellular decomposition of $\operatorname{cl}(C)$ such that each old cell which is contained in $\operatorname{cl}(C)$ and each set of the form $g^{-1}(c)$, where c is a cell of the old cellular decomposition of $\operatorname{cl}(A)$, is a finite union of cells. Remove C from the stratification. For each of the strata C_i that have replaced A in the new stratification add the set $g^{-1}(C_i)$ to the stratification. Then every new stratum contained in the old stratum C gets mapped onto some C_i by construction. The new strata are homeomorphic to disks, as they are homeomorphic images of disks under g^{-1}. Note that the refinement of the cellular decomposition may change the cellular decomposition of strata that are not contained in C, but this does not introduce new injuries, as the underlying set of these strata does not change.

If there is a stratum C' that was mapped onto C, repeat this procedure with C' playing the role of C and C playing the role of A. Continue in this manner until all injuries of the first kind that arise in this way are removed. This happens after finitely many steps as the rank of C increases by one with each repetition. ◀

▶ **Lemma 10.** *If the stratification of T contains an injury of the second kind, then the decomposition can be refined such that the new injury signature is the old injury signature with at least one signature of the form $(2, m, k)$ removed and finitely many signatures added of the form $(i, l, k-1)$.*

Proof. Let A be an injury of the second kind with dimension m and rank k. By assumption it is mapped into a stratum C.

Let $A_1, \ldots, A_s$ be all other injuries of the second kind that get mapped into the same stratum C. By Theorem 8 there exists a semi-algebraic cellular decomposition of $\operatorname{cl}(C)$ such that each of the sets $g(A)$ and $g(A_i)$, and each of the cells of the old cellular decomposition of $\operatorname{cl}(C)$ is a finite union of cells. Replace the cellular decomposition on $\operatorname{cl}(C)$ with this new decomposition. This defines a new cellular decomposition of T that refines the old one. Then C is the disjoint union of $g(A)$, the $g(A_i)$'s, and a finite union of cells $c_1, \ldots, c_k$. Remove C from the stratification and add $g(A)$, the $g(A_i)$'s, and $c_1, \ldots, c_k$ as new strata.

Then A and the A_i's are no longer injuries, as they are mapped onto strata by construction. The c_i are potentially new injuries with rank $k-1$. As A is mapped into C, different strata are disjoint, and g is injective, no stratum was mapped onto C before. For all other strata, neither their underlying set nor the underlying set of their image has changed, so that no further injuries are added. ◀

Now all injuries can be removed by repeatedly applying the two previous lemmas in the correct order:

▶ **Lemma 11.** *Consider the following algorithm:*

1. *Find a semi-algebraic cellular decomposition of T such that each cell has constant rank. Let each cell of the decomposition be a stratum.*
2. *Sort the injuries of the stratification lexicographically, comparing first by dimension, then by negative rank, and then by kind.*
3. *Pick a minimal element with respect to this ordering. If it is of the first kind, remove it using the first lemma. If it is of the second kind, remove it using the second lemma.*
4. *If there are no injuries left, then output the stratification. Otherwise go to (2).*

Then this algorithm terminates in finite time. It returns a g-invariant stratification of T.

Proof. Let N be the highest rank among all injuries. Consider the $N \times n$ matrix $A = ((a_{i,j}, b_{i,j}))$ whose entry at index (i, j) is the pair $(a_{i,j}, b_{i,j}) \in \mathbb{N} \times \mathbb{N}$ where $a_{i,j}$ is the number of injuries of the first kind of rank $N - i$ and dimension j, and $b_{i,j}$ is the number of injuries of the second kind of rank $N - i$ and dimension j. Thus, the j^{th} column of the matrix records all injuries of dimension j and the i^{th} row of the matrix records all injuries of rank $N - i$. In each step the algorithm will process an injury which is recorded in the first non-zero entry in the first non-zero column of the matrix A. Define the rank of a non-zero column to be the highest rank of all injuries that are recorded in the column.

We will now show by induction on the index of the first non-zero column that the algorithm will make the first non-zero column into a zero column in finitely many steps without introducing injuries whose dimension is smaller than or equal to the column index or whose rank is greater than or equal to the column rank. In particular it will make all columns into a zero column in finitely many steps, proving termination.

If the first non-zero column is the first column of the matrix, then the algorithm will first remove all injuries of the first kind and of highest rank. This potentially introduces new injuries of the second kind of the same dimension and the same rank, but no further injuries. It will then remove all injuries of the second kind and of highest rank. This potentially introduces new injuries of any kind and any dimension, but of a rank that is strictly lower than the column rank. Once the first non-zero entry of the column has been made into zero, the algorithm will proceed with the next entry of the column and continue in the same manner until the column is made into a zero column. All injuries introduced by the end of this process have rank strictly less than the initial column rank. Their dimension is higher than the column index since the column was assumed to be the first column in the matrix.

Now assume that we have shown the result for all column indexes strictly smaller than $j > 0$. Assume that the j^{th} column is the first non-zero column, and let $(a_{i,j}, b_{i,j})$ be its first non-zero entry. If $a_{i,j} > 0$ then an application of one step of the algorithm decreases $a_{i,j}$ by one, increases $b_{i,j}$ by an arbitrary amount, and potentially introduces new injuries of the second kind in the i^{th} row and to the left of (i, j). The algorithm will now proceed to remove non-zero columns with smaller index than j. These columns have rank at most $N - i$, so that by induction hypothesis any injuries added in the process of removing these columns have rank strictly smaller than $N - i$, so that in particular the entry at index (i, j) does not change throughout. Therefore, after finitely many steps, the first non-zero entry in the first non-zero column will be $(a_{i,j} - 1, b_{i,j} + c)$ with $c \geq 0$. This shows that the value of the first entry at index (i, j) is strictly decreasing and will hence eventually be equal to zero. Note that at no point injuries of rank greater than or equal to $N - i$ were introduced. If $a_{i,j} = 0$ and $b_{i,j} > 0$, then an application of one step of the algorithm decreases $b_{i,j}$ by at least one, and potentially introduces new injuries of lower rank and any dimension. The algorithm will proceed to remove non-zero columns of lower index, leaving the entries with index (i, j) unaffected by induction hypothesis. Thus, after finitely many steps, the first non-zero entry

in the first non-zero column will be $(0, b'_{i,j})$ with $b'_{i,j} < b_{i,j}$. It follows that the entry is made into zero after finitely many steps. Again, no injuries of rank greater than or equal to $N - i$ were introduced. The algorithm will proceed to process the rest of the entries of the column in the same manner until the column and all columns to the left of it are made into zero, not introducing any injuries of rank $N - i$ or greater. ◀

4 Proof of Lemma 7

In this section we prove Lemma 7. Throughout this section, let $g\colon \mathbb{R}^n \to \mathbb{R}^n$ be a continuous semi-algebraic map, let $K \subseteq \mathbb{R}^n$ be a compact semi-algebraic set such that for all $x \in K$ there exists $m \in \mathbb{N}$ such that $g^m(x) \notin K$, let $T \subseteq \mathbb{R}^N$ be a compact semi-algebraic set, let $p\colon T \to K \cup g(K)$ be a continuous semi-algebraic surjection, and let $\widetilde{g}\colon p^{-1}(K) \to T$ be an injective continuous semi-algebraic map such that the following diagram commutes:

$$\begin{array}{ccc} p^{-1}(K) & \xrightarrow{\widetilde{g}} & T \\ \downarrow{\scriptstyle p} & & \downarrow{\scriptstyle p} \\ K & \xrightarrow{g} & K \cup g(K) \end{array}$$

Write $S = p^{-1}(K)$. Fix a $\widetilde{g}$-invariant stratification of T.

Define a well-founded partial ordering on the strata of T as follows: $S_1 \leq S_2$ if and only if $S_1 = \widetilde{g}^m(S_2)$ for some $m \geq 0$. Arrange the strata in a linear chain $S_1, S_2, \ldots, S_m$ such that $i < j$ implies $\dim S_i \leq \dim S_j$ and $S_j \not\leq S_i$.

Let $F_1 = \{x \in T \mid \exists x' \in S_1 . p(x') = p(x)\}$. For all $k = 2, \ldots, m$, define the set $F_k = \{x \in T \mid \exists x' \in S_k \cup F_{k-1} . p(x') = p(x)\}$. Note that we have $F_k \supseteq F_i$ and $F_k \supseteq S_i$ for all $i \leq k$.

Before we begin with the construction of $\widetilde{f}$, we record some basic properties of the sets S_k and F_k:

▶ **Lemma 12.** *The sets F_k and $F_k \cup S_{k+1}$ are closed for all k.*

Proof. The set S_1 is a zero-dimensional disk and hence a singleton. The set $F_1 = p^{-1}(p(S_1))$ is closed, as S_1 is compact and p is continuous.

Assume now that F_k is closed for a given k. The boundary of S_{k+1} is contained in the union of all strata of dimension $< \dim S_{k+1}$, which by definition are contained in F_k. Hence $F_k \cup S_{k+1}$ is closed. It then follows, using compactness of $F_k \cup S_{k+1}$ and continuity of p, that the set $F_{k+1} = p^{-1}(p(F_k \cup S_{k+1}))$ is closed as well. ◀

▶ **Lemma 13.** *For all $k = 1, \ldots, m$, the set F_k contains all fibres of p that intersect it.*

Proof. Assume that there exists $x' \in p^{-1}(x) \cap F_k$. Then there exists $x'' \in S_k \cup F_{k-1}$ with $p(x'') = x$. Let $x''' \in p^{-1}(x)$. Then $p(x''') = x = p(x'')$ with $x'' \in S_k \cup F_{k-1}$, so that $x''' \in F_k$. ◀

▶ **Lemma 14.** *For all $k = 1, \ldots, m$, if $x \in F_k \cap S$ then $\widetilde{g}(x) \in F_k$.*

Proof. We prove this by induction on k. We have $S_1 \cap S = \emptyset$, for if this were not the case then we could apply $\widetilde{g}$ to S_1 to obtain a stratum $S_j = \widetilde{g}(S_1)$ with $j > 1$. This would imply $S_j \leq S_1$ which directly contradicts the definition of the linear ordering on strata. It follows that $p(S_1) \cap K = \emptyset$ and hence $F_1 \cap S = p^{-1}(p(S_1)) \cap p^{-1}(K) = \emptyset$. This establishes the claim for $k = 1$.

Assume that the claim is true for $k-1$. Let $x \in F_k \cap S$. By definition of F_k there exists $x' \in S_k \cup F_{k-1}$ with $p(x') = p(x)$. Since x is assumed to be contained in $S = p^{-1}(K)$ we have $p(x') \in K$ and thus $x' \in S$, so that $\widetilde{g}$ can be applied to x'.

If $x' \in F_{k-1}$ then $\widetilde{g}(x') \in F_{k-1}$. Since $p \circ \widetilde{g}(x') = p \circ \widetilde{g}(x)$ and F_{k-1} contains all fibres that intersect it, it follows that $\widetilde{g}(x) \in F_{k-1}$. As $F_{k-1} \subseteq F_k$ the claim follows.

Now assume that $x' \in S_k$. Since all strata are disjoint and S is a union of strata, the stratum S_k is either contained in S or disjoint from S. We have seen above that $x' \in S$, so that $S_k \subseteq S$. It follows that $\widetilde{g}$ can be applied to S_k. Since the stratification is $\widetilde{g}$-invariant, the set $\widetilde{g}(S_k)$ is again a stratum, say, $\widetilde{g}(S_k) = S_j$ for some j. By definition of the ordering on strata we have $S_j \leq S_k$ which implies $j \leq k$, and hence $S_j \subseteq F_k$ by definition of F_k. Thus, $\widetilde{g}(x') \in F_k$. Since $p \circ \widetilde{g}(x') = p \circ \widetilde{g}(x)$ and F_k contains all fibres that intersect it, it follows that $\widetilde{g}(x) \in F_k$. ◀

We prove by induction on k that there exists a continuous function $\widetilde{f}\colon F_k \to \mathbb{R}$ which satisfies the functional equation $\widetilde{f}(x) = \widetilde{f}(\widetilde{g}(x)) + 1$ for all $x \in F_k \cap S$ and is constant on all fibres of p which intersect F_k. Note that by Lemma 14, if $x \in F_k \cap S$ then $\widetilde{g}(x) \in F_k$, so that $\widetilde{f}(\widetilde{g}(x))$ is defined, and it makes sense to ask that $\widetilde{f}$ satisfy the functional equation in x.

The stratum S_1 is necessarily a 0-cell and minimal with respect to the ordering on strata. In particular we have $S_1 \cap S = \emptyset$ and hence $F_1 \cap S = \emptyset$ (see also the proof of Lemma 14 above). Put $\widetilde{f}(x) = 0$ on F_1. Then clearly $\widetilde{f}$ has all the required properties, as the functional equation is only required to hold for points in S.

Assume that $\widetilde{f}\colon F_k \to \mathbb{R}$ has been defined. We want to extend $\widetilde{f}$ to F_{k+1}. Either S_{k+1} is minimal with respect to the ordering or it is mapped by $\widetilde{g}$ onto some S_i with $i \leq k$.

If S_{k+1} is minimal with respect to the ordering on strata, then S_{k+1} is not contained in S. Extend $\widetilde{f}$ continuously from $F_k \cap \mathrm{cl}\,(S_{k+1})$ to all of S_{k+1}, ensuring that $p(x) = p(x')$ implies $\widetilde{f}(x) = \widetilde{f}(x')$. This can be achieved as follows: By assumption the map $\widetilde{f}$ is defined and constant on all fibres of p which intersect F_k, so that we obtain a well-defined continuous map $\varphi\colon p(F_k \cap \mathrm{cl}\,(S_{k+1})) \to \mathbb{R}$ by letting $\varphi(p(x)) = \widetilde{f}(x)$. By the Tietze extension theorem the map φ extends continuously to $p(\mathrm{cl}\,(S_{k+1}))$. Then letting $\widetilde{f}(x) = \varphi(p(x))$ on S_{k+1} yields the desired function. This extension is continuous by definition. It still satisfies the functional equation since no points in S were added to its domain.

We hence have a continuous map $\widetilde{f}\colon F_k \cup S_{k+1} \to \mathbb{R}$ which factors through a continuous map $f\colon p(F_k \cup S_{k+1}) \to \mathbb{R}$. This yields a continuous extension to F_{k+1}, by letting $\widetilde{f}(x) = f \circ p(x)$ for all $x \in F_{k+1}$.

The extension still satisfies the functional equation: Let $x \in F_{k+1}$. Then there exists $x' \in S_{k+1} \cup F_k$ with $p(x') = p(x)$. If $x' \in S_{k+1}$ then $x' \notin S$ by minimality of S_{k+1}, so there is no constraint on $\widetilde{f}(x')$. If $x' \in F_k$ then the functional equation $\widetilde{f}(x') = \widetilde{f}(\widetilde{g}(x')) + 1$ is satisfied by induction hypothesis. This yields the functional equation for x as $\widetilde{f}$ is constant on fibres and $\widetilde{g}$ sends fibres to fibres.

This concludes the case where S_{k+1} is minimal with respect to the ordering on strata. Let us now assume that $\widetilde{g}(S_{k+1}) = S_i$ for some i.

On S_{k+1}, put $\widetilde{f}(x) = \widetilde{f}(\widetilde{g}(x)) + 1$. Then clearly $\widetilde{f}$ is continuous and satisfies the functional equation on S_{k+1}. To show that it is continuous on $S_{k+1} \cup F_k$ it suffices to show that it is continuous on the closure of S_{k+1}. Thus, let $(x_n)_n$ be a sequence in S_{k+1} which converges to $x \in F_k$. The set S is closed, so that $x \in S$. As $\widetilde{g}(x_n) \in F_k$ and F_k is closed, we have $\widetilde{g}(x) \in F_k$. By definition we have $\widetilde{f}(x_n) = \widetilde{f}(\widetilde{g}(x_n)) + 1$. Since $\widetilde{f}$ is continuous on F_k by induction hypothesis we have $\widetilde{f}(\widetilde{g}(x_n)) \to \widetilde{f}(\widetilde{g}(x))$. Hence, $\widetilde{f}(x_n) \to \widetilde{f}(\widetilde{g}(x)) + 1 = \widetilde{f}(x)$ where the equality holds because the functional equation is satisfied on F_k by induction hypothesis.

On F_{k+1}, let $\widetilde{f}(x) = \widetilde{f}(x')$ where $x' \in S_{k+1}$ and $p(x') = p(x)$. As above, the extension to F_{k+1} is well-defined and continuous.

It remains to show that the extension satisfies the functional equation on $F_{k+1} \cap S$. Let $x \in F_{k+1} \cap S$. Then $p(x) = p(x')$ where $x' \in S_{k+1} \cup F_k$. If $x' \in F_k$ then the functional equation $\widetilde{f}(x) = \widetilde{f}(\widetilde{g}(x)) + 1$ is satisfied by the same argument as above. If $x' \in S_{k+1}$ then the functional equation $\widetilde{f}(x') = \widetilde{f}(\widetilde{g}(x')) + 1$ is satisfied by construction. By construction, $\widetilde{f}(x') = \widetilde{f}(x)$ and $\widetilde{f}(\widetilde{g}(x')) = \widetilde{f}(\widetilde{g}(x))$ since $\widetilde{g}(x') \in F_k$, the set F_k contains all fibres that intersect it, and $\widetilde{f}$ is constant on fibres.

5 Decidability of termination in the case of linear updates

Theorem 1 raises the question whether the existence of a ranking function or, equivalently, termination is decidable for compact semi-algebraic K and continuous semi-algebraic update $g\colon \mathbb{R}^n \to \mathbb{R}^n$. We can answer this affirmatively in the case where g is linear:

▶ **Theorem 15.** *There exists an algorithm which receives as input a compact semi-algebraic set $K \subseteq \mathbb{R}^n$, encoded as a finite boolean combination of rational polynomial inequalities, and a linear map $A\colon \mathbb{R}^n \to \mathbb{R}^n$, encoded as a rational matrix, and decides whether for all $x \in K$ there exists $i \in \mathbb{N}$ such that $A^i x \notin K$.*

Proof. If every point of K escapes K under A in finitely many steps, then by Proposition 2 there exists a constant $M \in \mathbb{N}$ such that every point of K escapes K in at most M steps. For a fixed $m \in \mathbb{N}$, the statement that every point of K escapes K in at most m steps is a sentence in the first-order theory of the reals and hence decidable. It follows that we can semi-decide if every point escapes K under A.

We will show in Theorem 18 below that the existence of a point in K which does not escape K under A implies the existence of a semi-algebraic invariant for A in K, *i.e.*, a semi-algebraic set $S \subseteq K$ with $A(S) \subseteq S$. The statement that there exists a semi-algebraic invariant which can be expressed as a boolean combination of m inequalities involving polynomials of degree at most d is a sentence in the first-order theory of the reals and hence decidable. It follows that the existence of a non-escaping point is semi-decidable as well. ◀

More generally, termination is decidable for affine maps, *i.e.*, maps of the form $g(x) = Ax + b$ where A is an $n \times n$-matrix and b is an n-dimensional vector:

▶ **Corollary 16.** *There exists an algorithm which receives as input a compact semi-algebraic set $K \subseteq \mathbb{R}^n$, encoded as a finite boolean combination of rational polynomial inequalities, and an affine map $A\colon \mathbb{R}^n \to \mathbb{R}^n$, encoded by a rational matrix, and decides whether for all $x \in K$ there exists $i \in \mathbb{N}$ such that $A^i x \notin K$.*

Proof. Let $A(x) = B(x) + c$. Apply Theorem 15 to the compact semi-algebraic set $K \times \{1\} \subseteq \mathbb{R}^{n+1}$ and the linear map $\widetilde{A}(x, z) = (B(x) + cz, z)$. ◀

Note that the decision procedure in the proof of Theorem 15 combines two unbounded searches, one of which is guaranteed to terminate. We hence do not obtain any non-trivial upper bound on the computational complexity of deciding termination.

To complete the proof of Theorem 15 we require the following version of Kronecker's theorem, as stated in [15, Corollary 3.1]:

▶ **Lemma 17.** *Let $\lambda_1, \ldots, \lambda_f$ be complex numbers of modulus one. Consider the free abelian group*

$$L = \left\{ (v_1, \ldots, v_f) \in \mathbb{Z}^f \mid \lambda_1^{v_1} \cdot \cdots \cdot \lambda_f^{v_f} = 1 \right\}.$$

Let $\{\ell_1, \ldots, \ell_p\}$ be a basis of this group. Then L is a free abelian group with a finite basis $\{\ell_1, \ldots, \ell_p\}$. Let

$$T = \left\{(z_1, \ldots, z_f) \in \mathbb{C}^f \mid |z_1| = \cdots = |z_f| = 1, (z_1 \cdot \cdots \cdot z_f)^{\ell_i} = 1 \text{ for all } i = 1, \ldots, p\right\}.$$

Then the sequence $(\lambda_1^s, \ldots, \lambda_f^s)_{s\in\mathbb{N}}$ is dense in T.

▶ **Theorem 18.** *Let $K \subseteq \mathbb{R}^n$ be a compact semi-algebraic set and let $A\colon \mathbb{R}^n \to \mathbb{R}^n$ be a linear map. Assume that there exists $x \in K$ such that $A^i x \in K$ for all $i \in \mathbb{N}$. Then there exists a closed semi-algebraic set $S \subseteq K$ with $A(S) \subseteq S$.*

Proof. By choosing an appropriate basis of $\mathbb{R}^n$ we may assume that A is given as a matrix in real Jordan normal form, *i.e.*, A can be written as

$$\begin{pmatrix} A_1 & & & \\ & A_2 & & \\ & & \ddots & \\ & & & A_l \end{pmatrix}$$

where the A_k's are Jordan blocks of the form

$$A_k = \begin{pmatrix} \Lambda_k & I & & & \\ & \Lambda_k & I & & \\ & & \ddots & \ddots & \\ & & & \Lambda_k & I \\ & & & & \Lambda_k \end{pmatrix}$$

where, by slight abuse of notation, Λ_k is either a real eigenvalue or a 2×2-matrix corresponding to a pair of complex conjugate eigenvalues, and I is either the number 1 or the 2×2-identity matrix. The s^{th} iterate of A_k is given by:

$$A_k^s = \begin{pmatrix} \Lambda_k^s & \binom{s}{1}\Lambda_k^{s-1} & \binom{s}{2}\Lambda_k^{s-2} & \cdots & \binom{s}{d_k-1}\Lambda_k^{s-d_k+1} \\ & \Lambda_k^s & \binom{s}{1}\Lambda_k^{s-1} & \cdots & \binom{s}{d_k-2}\Lambda_k^{s-d_k+2} \\ & & \ddots & \vdots & \vdots \\ & & & \Lambda_k^s & \binom{s}{1}\Lambda_k^{s-1} \\ & & & & \Lambda_k^s \end{pmatrix}.$$

Here, d_k denotes the size of the Jordan block, *i.e.*, the number of Λ_k's.

Let $x \in K$ be a point whose orbit under A does not escape K. Fix a Jordan block A_k as above. Let $x^k = (x_0^k, \ldots, x_{d_k-1}^k)$ be the corresponding component of x. Again by slight abuse of notation, x_i^k denotes a number if Λ_k is a number and a 2-dimensional vector if Λ_k is a 2×2-matrix.

If Λ_k is real, let $|\Lambda_k|$ be the absolute value of Λ_k. If Λ_k is a 2×2-matrix, then we have $\Lambda_k = \begin{pmatrix} a & -b \\ b & a \end{pmatrix}$. In this case, let $|\Lambda_k| = \sqrt{a^2 + b^2}$.

If $|\Lambda_k| > 1$ then we claim that $x^k = (0, \ldots, 0)$. Indeed, if $x_i^k \neq 0$, then the i^{th} component of $A_k^s(x^k)$ is of the form $\Lambda_k^s x_i^k + O(|\Lambda_k^{s-1}|)$. Hence, the euclidean distance of $A^s x$ to 0 is unbounded as $s \to \infty$, contradicting the assumption that the sequence $A^s x$ stays in the compact set K.

If $|\Lambda_k| < 1$ then each of the expressions $\binom{s}{i}\Lambda_k^{s-i}$ converges to zero as $s \to \infty$, so that $A_k^s x^k \to 0$ as $s \to \infty$.

If $|\Lambda_k| = 1$ then we claim that $x^k = (x_0^k, 0, \ldots, 0)$. Indeed, the first component of $A^s x^k$ is equal to

$$\Lambda_k^s x_0 + \sum_{i=1}^{d_k - 1} \binom{s}{i} \Lambda_k^{s-i} x_i^k.$$

For large s the term $\binom{s}{i}\Lambda_k^{s-i}x_i^k$ with the largest i where $x_i^k \neq 0$ dominates the whole expression. If $i \neq 0$ then the expression is unbounded as $s \to \infty$. The claim follows.

Permute the basis vectors such that A is given in the new basis as

$$A = \left(\begin{array}{cccc|c} \Lambda_1 & & & & \\ & \Lambda_2 & & & \\ & & \ddots & & \\ & & & \Lambda_f & \\ \hline & & & & \text{A'} \end{array}\right)$$

Where $\Lambda_1, \ldots, \Lambda_f$ correspond to all the real and complex eigenvalues of A of modulus one, so that $(A^s x)_i \to 0$ for all $i \geq f + 1$.

Let

$$\bar{A} = \begin{pmatrix} \Lambda_1 & & & & & & \\ & \Lambda_2 & & & & & \\ & & \ddots & & & & \\ & & & \Lambda_f & & & \\ & & & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 0 \end{pmatrix}$$

Then we have $A\bar{A}^s x = \bar{A}^{s+1} x$, so that the orbit of x under $\bar{A}$ is invariant under A. Since A is continuous, the closure of the orbit is still invariant under A. We will now show that the closure of the orbit is a semi-algebraic set.

We can associate with each Λ_i with $i \leq f$ a complex algebraic number λ_i of modulus one. Let

$$L = \left\{ (v_1, \ldots, v_f) \in \mathbb{Z}^f \mid \lambda_1^{v_1} \cdot \cdots \cdot \lambda_f^{v_f} = 1 \right\}.$$

Then L is a free abelian group with a finite basis $\{\ell_1, \ldots, \ell_p\}$. Let

$$T = \left\{ (z_1, \ldots, z_f) \in \mathbb{C}^f \mid |z_1| = \cdots = |z_f| = 1, (z_1 \cdot \cdots \cdot z_f)^{\ell_i} = 1 \text{ for all } i = 1, \ldots, p \right\}$$

Then T is a complex algebraic set and by Lemma 17 the sequence $(\lambda_1^s, \ldots, \lambda_f^s)$ is dense in T. This yields a real algebraic subset T' of $\mathbb{R}^{n \times n}$ such that $\bar{A}^s$ is dense in T'. The matrix evaluation map $\mathbb{R}^{n \times n} \times \mathbb{R}^n \to \mathbb{R}^n$ is a polynomial map, so that the image of $T' \times \{x\}$ under this map is a semi-algebraic subset of $\mathbb{R}^n$. But this is the same as the closure of the orbit of x under $\bar{A}$.

We have now shown that the closure S of the orbit of x under $\bar{A}$ is a semi-algebraic invariant for A. It remains to show that S is contained in K. The above argument establishes that the iterates $\bar{A}^s x$ are dense in S. Now, A can be written as $A = \bar{A} + B$ with $A^s = \bar{A}^s + B^s$

and $B^s x \to 0$ as $s \to \infty$. Since the iterates $\bar{A}^s x$ are dense in S, for every $y \in S$ there exists a sequence $(s_k)_k$ with $\bar{A}^{s_k} x \to y$ as $k \to \infty$. It follows that $A^{s_k} x = \bar{A}^{s_k} x + B^{s_k} x \to y$ as $k \to \infty$. By assumption $A^{s_k} x \in K$ for all k, and since K is closed it follows that $y \in K$. We conclude that $S \subseteq K$. ◀

6 Conclusion

We have shown, by non-constructive means, that a single-path loop with continuous semi-algebraic update function and compact semi-algebraic guard set terminates over the reals if and only if it has a polynomial ranking function. In the case of an affine update we have shown the existence of a polynomial ranking function to be decidable by proving that any non-terminating loop of this form admits a semi-algebraic invariant.

This naturally suggests the question whether the existence of a polynomial ranking function can be decided for non-linear updates as well. A sufficient condition for decidability which may be of independent interest is whether an analogue of Theorem 18 holds true for certain classes of non-linear maps, say for instance whether any non-terminating polynomial loop with compact guard admits a semi-algebraic invariant.

Further, it would be interesting to study the computational complexity of deciding termination in the case of affine updates. Our decidability proof unfortunately does not yield any non-trivial complexity bounds.

Another direction of future research is to ascertain whether Theorem 1 generalises to set-valued semi-algebraic updates with appropriate continuity properties. Also the boundedness assumption on the guard set deserves to be further scrutinised. While the examples after Theorem 1 show that closedness is necessary for the existence of a continuous ranking function and that boundedness is necessary for the existence of a polynomial ranking function, it seems reasonable to conjecture that a terminating semi-algebraic loop with a closed but not necessarily bounded guard has, say, a piecewise-linear ranking function.

References

1. Amir M. Ben-Amram, Jesús J. Doménech, and Samir Genaim. Multiphase-Linear Ranking Functions and Their Relation to Recurrent Sets. In Bor-Yuh Evan Chang, editor, *Static Analysis*, pages 459–480. Springer International Publishing, 2019.
2. Amir M. Ben-Amram and Samir Genaim. Ranking Functions for Linear-Constraint Loops. *J. ACM*, 61(4):26:1–26:55, 2014.
3. Amir M. Ben-Amram and Samir Genaim. Complexity of Bradley-Manna-Sipma Lexicographic Ranking Functions. In *Computer Aided Verification*, volume 9207 of *Lecture Notes in Computer Science*, pages 304–321. Springer, 2015.
4. Amir M. Ben-Amram and Samir Genaim. On Multiphase-Linear Ranking Functions. In *Computer Aided Verification - 29th International Conference, CAV 2017*, volume 10427 of *Lecture Notes in Computer Science*, pages 601–620. Springer, 2017.
5. Jacek Bochnak, Michel Coste, and Marie-Françoise Roy. *Real Algebraic Geometry*. Springer-Verlag Berlin Heidelberg, 1998.
6. Mark Braverman. Termination of Integer Linear Programs. In *Computer Aided Verification, 18th International Conference, CAV, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 372–385. Springer, 2006.
7. Krishnendu Chatterjee, Hongfei Fu, and Amir Kafshdar Goharshady. Termination Analysis of Probabilistic Programs Through Positivstellensatz's. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification - 28th International Conference, CAV 2016, Proceedings, Part I*, volume 9779 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2016.

8 Yinghua Chen, Bican Xia, Lu Yang, Naijun Zhan, and Chaochen Zhou. Discovering Non-linear Ranking Functions by Solving Semi-algebraic Systems. In *Theoretical Aspects of Computing - ICTAC 2007, 4th International Colloquium, 2007, Proceedings*, volume 4711 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2007.

9 Byron Cook, Abigail See, and Florian Zuleger. Ramsey vs. Lexicographic Termination Proving. In *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013*, volume 7795 of *Lecture Notes in Computer Science*, pages 47–61. Springer, 2013.

10 Patrick Cousot. Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming. In *Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Proceedings*, volume 3385 of *Lecture Notes in Computer Science*, pages 1–24. Springer, 2005.

11 Florian Frohn, Marcel Hark, and Jürgen Giesl. On the Decidability of Termination for Polynomial Loops. *CoRR*, abs/1910.11588, 2019. `arXiv:1910.11588`.

12 Mehran Hosseini, Joël Ouaknine, and James Worrell. Termination of Linear Loops over the Integers. In *46th International Colloquium on Automata, Languages, and Programming, ICALP*, volume 132 of *LIPIcs*, pages 118:1–118:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

13 Yi Li. Termination of Single-Path Polynomial Loop Programs. In *Theoretical Aspects of Computing - ICTAC, Proceedings*, volume 9965 of *Lecture Notes in Computer Science*, pages 33–50, 2016.

14 Yi Li, Wenyuan Wu, and Yong Feng. On ranking functions for single-path linear-constraint loops. *Int J Softw Tools Technol Transfer*, 2019. `doi:10.1007/s10009-019-00549-9`.

15 Joël Ouaknine and James Worrell. Positivity Problems for Low-Order Linear Recurrence Sequences. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 366–379. SIAM, 2014.

16 Andreas Podelski and Andrey Rybalchenko. A Complete Method for the Synthesis of Linear Ranking Functions. In *Verification, Model Checking, and Abstract Interpretation, 5th International Conference, VMCAI 2004, Proceedings*, volume 2937 of *Lecture Notes in Computer Science*, pages 239–251. Springer, 2004.

17 Ashish Tiwari. Termination of Linear Programs. In Rajeev Alur and Doron A. Peled, editors, *Computer Aided Verification, 16th International Conference, CAV 2004, Proceedings*, volume 3114 of *Lecture Notes in Computer Science*, pages 70–82. Springer, 2004.

18 Caterina Urban. The Abstract Domain of Segmented Ranking Functions. In *Static Analysis - 20th International Symposium, SAS 2013, Proceedings*, volume 7935 of *Lecture Notes in Computer Science*, pages 43–62. Springer, 2013.

19 Bican Xia and Zhihai Zhang. Termination of linear programs with nonlinear constraints. *Journal of Symbolic Computation*, 45(11):1234–1249, 2010. `doi:10.1016/j.jsc.2010.06.006`.

On the Separability Problem of String Constraints

Parosh Aziz Abdulla
Uppsala University, Sweden
parosh@it.uu.se

Mohamed Faouzi Atig
Uppsala University, Sweden
mohamed_faouzi.atig@it.uu.se

Vrunda Dave
IIT Bombay, India
vrunda@cse.iitb.ac.in

Shankara Narayanan Krishna
IIT Bombay, India
krishnas@cse.iitb.ac.in

Abstract

We address the separability problem for straight-line string constraints. The separability problem for languages of a class C by a class S asks: given two languages A and B in C, does there exist a language I in S separating A and B (i.e., I is a superset of A and disjoint from B)? The separability of string constraints is the same as the fundamental problem of interpolation for string constraints. We first show that regular separability of straight line string constraints is undecidable. Our second result is the decidability of the separability problem for straight-line string constraints by piece-wise testable languages, though the precise complexity is open. In our third result, we consider the positive fragment of piece-wise testable languages as a separator, and obtain an ExpSpace algorithm for the separability of a useful class of straight-line string constraints, and a Pspace-hardness result.

2012 ACM Subject Classification Security and privacy → Logic and verification; Theory of computation → Verification by model checking

Keywords and phrases string constraints, separability, interpolants

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.16

Related Version A full version of the paper is available at `https://arxiv.org/abs/2005.09489`.

1 Introduction

The *string* data type is widely used in almost all modern programming and scripting languages. Many of the well-known security vulnerabilities such as SQL injections and cross-site scripting attacks are often caused by an improper handling of strings. The detection of such vulnerabilities is usually reduced to the satisfiability of a formula which is then solved by SMT solvers (e.g., [39, 40, 46, 30]). Therefore, string constraints solving has received considerable attention in recent years (e.g. [13, 12, 27, 46, 47, 42, 28, 26, 1, 30, 10, 25]) and this has led to the development of many efficient string solvers such as HAMPI [27], Z3-str3 [9], CVC4 [28, 29, 38], S3P [42, 43], Trau [1, 2, 5], SLOTH [25] and OSTRICH [14].

In spite of these advances, most of these tools do not provide any completeness guarantees. The foundational question regarding the decidability of string solving for a large class of string constraints has several challenges to be overcome. A major difficulty is that any reasonably expressive class of string constraints is either undecidable, or has its decidability status open for several years [20, 21, 22]. In fact, the satisfiability problem is undecidable even for the class of string constraints with concatenation (useful to model assignments in the

program) and transduction (useful to model sanitisation and replacement operations) [14]. A direction of research is to find meaningful and expressive subclasses of string constraints for which the satisfiability problem is decidable (e.g., [3, 5, 21, 30, 25, 12]). An interesting subclass, that has been studied extensively, is that of straight-line (SL) string constraints (e.g., [25, 14, 30, 25, 12]). The SL fragment was introduced by Barceló and Lin in [30]. Roughly, an SL constraint models the feasibility of a path of a string-manipulating program that can be generated by symbolic execution. The *satisfiability* of the SL fragment was shown to be EXPSPACE-complete in [30] and forms the basis of many of the tools above [25, 12].

In this paper, we focus on the fundamental problem of *interpolation/separability* for the SL fragment of string constraints. An *interpolant* for a pair of formulas A, B is a formula over their common vocabulary that is implied by A and is inconsistent with B. The Craig-Lyndon interpolation technique is very well-known in mathematical logic. McMillan [32] in his pioneering work, has also recognized interpolation as an efficient method for automated construction of abstractions of systems. Interpolation based algorithms have been developed for a number of problems in program verification [32, 33, 34]. Interpolation procedures have been implemented by many solvers for the theories most commonly used in program verification like linear arithmetic, uninterpreted functions with equality and some combination of such theories. In most of these algorithms, the interpolants were simple. The interpolation technique can also be used to check the unsatisfiability. In fact, the existence of an interpolant for formulas A and B implies the unsatisfiability of $A \wedge B$.

The notion of *separators* in formal language theory is the counterpart of interpolants in logic. The separability problem for languages of a class $\mathcal{C}$ by a class $\mathcal{S}$ asks: given two languages $I, E \in \mathcal{C}$, does there exist a language $S \in \mathcal{S}$ separating I and E? That is, $I \subseteq S$ and $S \cap E = \emptyset$. The language S is called the separator of I, E. Separability is a classical problem of fundamental interest in theoretical computer science, and has recently received a lot of attention. For instance, regular separability has been studied for one-counter automata [16], Parikh automata [15], and well-structured transition systems [17]. In the following, we use the terms interpolant or separator of two SL string constraints to mean the same thing, since the solutions of a string constraint can be interpreted as a language.

In this paper, we first show that any string constraint ϕ can be written as the conjunction of two SL string constraints A and B. Therefore, the interpolation problem for the pair A and B can be used to check the unsatisfiability of the string constraint ϕ. (Recall that the satisfiability problem for general string constraints is undecidable [14].)

Then, we consider the regular separability problem for SL string constraints. We show that this problem is undecidable (Theorem 2) by a reduction from the halting problem of Turing Machines. The main technical difficulty here is to ensure that the encoding of a sequence of configurations of a Turing machine results in SL string constraints.

Due to this undecidability, we focus on the separability problem of SL string constraints by piece-wise testable languages (PTL). A PTL is a finite Boolean combination of special regular languages called *piece languages* of the form $\Sigma^* a_1 \Sigma^* a_2 \dots \Sigma^* a_n \Sigma^*$, where all $a_j \in \Sigma$. PTL is a well-studied class of languages in the context of the separability problem (e.g. [36, 18, 19]). Furthermore, among the various separator classes considered in the literature, the class of piecewise testable languages (PTL) seems to be the most tractable: PTL-separability of regular languages is in PTIME [36, 18]. To decide the PTL-separability of SL string constraints, we first encode the solutions of an SL string constraint as the language of an Ordered Multi-Pushdown Automaton (OMPA) (Section 4.1). Then, we show that the PTL-separability of SL constraints can be reduced to the PTL-separability of OMPAs. To show the decidability of the latter problem, we first prove that the language of an OMPA: (1)

```
str old = real_escape_string(oldIn);
str new1 = real_escape_string(newIn1);
str new2 = real_escape_string(newIn2);
str pass = database_query("SELECT password FROM users WHERE userID=" userID);
if (old == pass AND new1 == new2 AND new1 != old )
    if (newIn1==newIn2 AND newIn1 != oldIn)
       str query = "UPDATE users SET password=" + new1 + "WHERE userID=" + userID;
       database_query(query);
```

Figure 1 A pseudo PHP code for changing password.

is a full trio [23] and (2) has a semilinear Parikh image. Using (1), we obtain the equivalence of the PTL separability problem and the *diagonal problem* for OMPAs from [19], where the equivalence has been shown to hold for full trios. Next, the decidability of PTL-separability problem for OMPAs is obtained from the decidability of the diagonal problem for OMPAs: the latter is obtained using (2) and [19] where the decidability of the diagonal problem has been shown for languages having a semilinear Parikh image. As a corollary of these results, we obtain the decidability of the PTL-separability problem for SL string constraints and OMPAs; however the exact complexity is still an open question. In fact, it is an open problem in the case of OMPAs with one stack (i.e., Context-Free Languages (CFLs)) [19].

Given the complexity question, we propose the class of positive piecewise testable languages (PosPTL) as separators. PosPTL is obtained as a negation-free Boolean combination of piece languages. As a first result (Theorem 12) we show that deciding PosPTL-separability for any language class has a very elegant proof: it suffices to check if the upward (downward) closure of one of the languages is disjoint from the other language. Using this result, we prove the PSPACE-completeness of the PosPTL-separability for CFLs, thereby progressing on the complexity front with respect to a problem which is open in the case of PTL-separability for CFLs. Then, we focus on a class of SL string constraints where the variables used in outputs of the transducers are independent of each other. This class contains SL string constraints with functional transducers (computing partial functions, by associating at most one output with each input). We prove the decidability and EXPSPACE membership for the PosPTL-separability of this class by first encoding the solutions of string constraints as outputs of two way transducers (2NFT), and then proving the decidability of PosPTL-separability for 2NFT.

Due to lack of space, missing proofs of all results can be found in the full version [4].

As a practical motivation of PosPTL (and PTL), consider the following pseudo-PHP code in Figure 1 obtained as a variation of the code at [31]. In this code, a user is prompted to change his password by entering the new password twice. In this code, database_query represents the function which executes the query given as its parameter. We use '+' operator as concatenation of strings and variables (variables are represented using blue color). The user inputs the old password oldIn and the new password twice : newIn1 and newIn2. These are sanitized and assigned to old, new1 and new2 respectively. The old sanitized password is compared with the value pass from the database to authenticate the user, and also with the new sanitized password to check that a different password has been chosen, and finally, the sanitized new passwords entered twice are checked to be the same.

Sanitization ensures that there are no SQL injections. To ensure the absence of SQL attacks, we require that the query query does not belong to a regular language Bad of bad patterns over some finite alphabet Σ (i.e., the program is safe). This safety condition can be expressed as the unsatisfiability of the following formula φ given by

$\texttt{new1} = \texttt{T}(\texttt{newIn1}) \wedge \texttt{new2} = \texttt{T}(\texttt{newIn2}) \wedge \texttt{old} = \texttt{T}(\texttt{oldIn}) \wedge \texttt{new1} = \texttt{new2} \wedge \texttt{pass} = \texttt{old} \wedge$
$\texttt{old} \neq \texttt{new1} \wedge \texttt{newIn1} = \texttt{newIn2} \wedge \texttt{query} = \texttt{u} \cdot \texttt{new1} \cdot \texttt{v} \cdot \texttt{userID} \wedge \texttt{query} \in \texttt{Bad}.$

Note that the check $\texttt{new1} = \texttt{new2}$ has to be done by the server to ensure the sanitized new passwords entered twice are same; however, the check $\texttt{newIn1} = \texttt{newIn2}$ is not redundant, since it can happen that post sanitization, the passwords may agree, but not before. The sanitization on lines 1, 2 and 3 is represented by the transducer $\texttt{T}$ and $\texttt{u}, \texttt{v}$ are the constant strings from line 7. It is easy to see that the program given here is safe iff the formula φ is unsatisfiable. Observe that the formula φ is not in the straight line fragment [30] since variable $\texttt{new1}$ has two assignments. Further, it also has a non-benign chain making it fall out of the fragment of string programs handled in [5]. However the formula φ can be rewritten as a conjunction of the two formula φ_1 and φ_2 in straight-line form where

$\varphi_1 : \texttt{new1} = \texttt{T}(\texttt{newIn1}) \wedge \texttt{old} = \texttt{T}(\texttt{oldIn}) \wedge \texttt{pass} = \texttt{old} \wedge \texttt{query} = \texttt{u} \cdot \texttt{new1} \cdot \texttt{v} \cdot \texttt{userID} \wedge \texttt{query} \in \texttt{Bad}$

$\varphi_2 : \texttt{new2} = \texttt{T}(\texttt{newIn2}) \wedge \texttt{new1} = \texttt{new2} \wedge \texttt{old} \neq \texttt{new1} \wedge \texttt{newIn1} = \texttt{newIn2}.$

It is easy to see that the program is safe if solution sets of φ_1 and φ_2 are separable by some PosPTL set, in that case, we can say that there is no solution which is common to φ_1 and φ_2 and thus $\varphi = \varphi_1 \wedge \varphi_2$ is unsatisfiable.

Related work. The satisfiability problem for string constraints is an active research area and there is a lot of progress in the last decade (e.g., [37, 27, 30, 12, 14, 5, 21, 22, 3, 45]). An interpolation based semi-decision procedure for string constraints has been proposed in [3]. As far as we know, this is the first time the separability problem has been studied in the context of string constraints.

2 Preliminaries

Notations. Let $[i, j]$ denote the set $\{i, \ldots, j\}$ for $i, j \in \mathbb{N}$. Let Σ be a finite alphabet. Σ^* denotes the set of all finite words over Σ and Σ^+ denotes $\Sigma^* \backslash \{\epsilon\}$ where ϵ is the empty word. We denote $\Sigma \cup \{\epsilon\}$ by Σ_ϵ. Let $u \in \Sigma^*$. We use u^R to denote the reverse of u. The length of the word u is denoted $|u|$ and the i^{th} symbol of u by $u[i]$. Given two words $u \in \Sigma^*$ and $v \in \Sigma^*$, we say that u is a subword of v (denoted $u \preceq v$) if there is a mapping $h : [1, |u|] \mapsto [1, |v|]$ such that (1) $u[i] = v[h(i)]$ for all $i \in [1, |u|]$, and (2) $h(i) < h(j)$ for all $i < j$.

(Multi-tape)-Automata. A *Finite State Automaton* (FSA) over an alphabet Σ is a tuple $\mathcal{A} = (Q, \Sigma, \delta, I, F)$, where Q is a finite set of *states*, $\delta \subseteq Q \times \Sigma_\epsilon \times Q$ is a set of *transitions*, and $I \subseteq Q$ (resp. $F \subseteq Q$) are the *initial* (resp. *accepting*) states. $\mathcal{A}$ accepts a word w iff there is a sequence $q_0 a_1 q_1 a_2 \cdots a_n q_n$ such that $(q_{i-1}, a_i, q_i) \in \delta$ for all $1 \leq i \leq n$, $q_0 \in I$, $q_n \in F$, and $w = a_1 \cdot \cdots \cdot a_n$. The *language* of $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the set all accepted words.

Given $n \in \mathbb{N}$, a *n-tape automaton* $\mathcal{T}$ is an automaton over the alphabet $(\Sigma_\epsilon)^n$. It *recognizes* the relation $\mathcal{R}(\mathcal{T}) \subseteq (\Sigma^*)^n$ that contains the n-tuple of words $(w_1, w_2, \ldots, w_n)$ for which there is a word $(a_{(1,1)}, a_{(2,1)}, \ldots, a_{(n,1)}) \cdots (a_{(1,m)}, a_{(2,m)}, \ldots, a_{(n,m)}) \in \mathcal{L}(\mathcal{T})$ with $w_i = a_{(i,1)} \cdots \cdots a_{(i,m)}$ for all $i \in \{1, \ldots, n\}$. A *transducer* is a 2-tape automaton.

Well-quasi orders. Given a (possibly infinite set) C, a quasi-order on C is a reflexive and transitive relation $\sqsubseteq \subseteq C \times C$. An infinite sequence $c_1, c_2, \ldots$ in C is said to be saturating if there exists indices $i < j$ s.t. $c_i \sqsubseteq c_j$. A quasi-order $\sqsubseteq$ is said to be a well-quasi order (wqo) on C if every infinite sequence in C is saturating. Observe that the subword ordering $\preceq$ between words u, v over a finite alphabet Σ is well-known to be a wqo on Σ^* [24].

Upward and Downward Closure: Given a wqo $\sqsubseteq$ on a set C, a set $U \subseteq C$ is said to be upward closed if for every $a \in U$ and $b \in C$, with $a \sqsubseteq b$, we have $b \in U$. The upward closure of a set $U \subseteq C$ is defined as $U{\uparrow} = \{b \in C \mid \exists a \in U, a \sqsubseteq b\}$. It is known that every

upward closed set U can be characterized by a finite *minor*. A minor $M \subseteq U$ is s.t. (i) for each $a \in U$, there is a $b \in M$ s.t. $b \sqsubseteq a$, and (ii) for all $a, b \in M$ s.t. $a \preceq b$, we have $a = b$. For an upward closed set U, let min be the function that returns the minor of U. Downward closures are defined analogously. The downward closure of a set $D \subseteq C$ is defined as $D{\downarrow} = \{b \in C \mid \exists a \in D, b \sqsubseteq a\}$. The notion of subword relation and thus upward and downward closures naturally extends to n-tuples of words. The subword relation here is component wise i.e. $(u_1, \ldots, u_n) \preceq_n (v_1, \ldots, v_n)$ iff $u_i \preceq v_i$ for all $i \in [1, n]$.

String Constraints. An atomic string constraint φ over an alphabet Σ and a set of string variables $\mathcal{X}$ is either: (1) a *membership constraint* of the form $x \in \mathcal{L}(\mathcal{A})$ where $x \in \mathcal{X}$ and $\mathcal{A}$ is a FSA (i.e., the evaluation of x is in the language of a FSA $\mathcal{A}$ over Σ), or (2) a *relational constraint* of the form $(t', t) \in \mathcal{R}(\mathcal{T})$ where t and t' are string terms (i.e., concatenation of variables in $\mathcal{X}$) and $\mathcal{T}$ is a transducer over Σ, and t and t' are related by a relation recognised by the transducer $\mathcal{T}$. $(t', t) \in \mathcal{R}(\mathcal{T})$ can also be written as $t' = \mathcal{T}(t)$, that is, $\mathcal{T}$ produces t' as the output on input t. For a given term t, $|t|$ denotes the number of variables appearing in t.

A string constraint Ψ is a conjunction of atomic string constraints. We define the semantics of string constraints using a mapping η, called *evaluation*, that assigns for each variable a word over Σ. The evaluation η can be extended in the straightforward manner to string terms as follows $\eta(t_1 \cdot t_2) = \eta(t_1) \cdot \eta(t_2)$. We extend also η to atomic constraints as follows: (1) $\eta(x \in \mathcal{L}(\mathcal{A})) = \top$ iff $\eta(x) \in \mathcal{L}(\mathcal{A})$, and (2) $\eta((t, t') \in \mathcal{R}(\mathcal{T})) = \top$ iff $(\eta(t), \eta(t')) \in \mathcal{R}(\mathcal{T})$. The truth value of Ψ for an evaluation η is defined in the standard manner. If $\eta(\Psi) = \top$ then η is a *solution* of Ψ, written $\eta \models \Psi$. The formula Ψ is *satisfiable* iff it has a solution.

A string constraint is said to be *Straight Line*[1] (SL) if it can be rewritten as $\Psi' \wedge \bigwedge_{i=1}^{k} \varphi_i$ where Ψ' is a conjunction of membership constraints, and $\varphi_1, \ldots, \varphi_k$ are relational constraints such that (1) there is a sequence of different string variables $x_1, x_2, \ldots, x_n$ with $n \geq k$, and (2) φ_i is of the form $(x_i, t_i) \in \mathcal{R}(\mathcal{T}_i)$ such that if a variable x_j is appearing in t_i then $j > i$. A string constraint in the SL form is called an SL formula. Observe that any string formula can be rewritten as a conjunction of two SL formulas (by using extra-variables).

▶ **Lemma 1.** *Given a string constraint Ψ, it is possible to construct two SL string constraints Ψ_1 and Ψ_2 such that Ψ is satisfiable iff $\Psi_1 \wedge \Psi_2$ is satisfiable.*

Let Ψ be a string constraint and $x_1, \ldots, x_n$ be the set of variables appearing in Ψ. We use $\mathcal{L}(\Psi)$ to denote the language of Ψ which consists of the set of n-tuple of words $(u_1, \ldots, u_n)$ such that there is an evaluation η with $\eta(\Psi) = \top$ and $\eta(x_i) = u_i$ for all $i \in [1, n]$.

The Separability Problem. Given two classes of languages $\mathcal{C}$ and $\mathcal{S}$, the separability problem for $\mathcal{C}$ by the separator class $\mathcal{S}$ is defined as follows: Given two languages I and E from the class $\mathcal{C}$, does there exist a separator $S \in \mathcal{S}$ such that $I \subseteq S$ and $E \cap S = \emptyset$.

3 Regular Separability of String Constraints.

Let Σ be an alphabet and k, n be two natural numbers. A set R of n-tuples of words over Σ is said to be regular (REG) iff there is a sequence of finite-state automata $\mathcal{A}_{(i,1)}, \ldots, \mathcal{A}_{(i,n)}$ for every $i \in [1, k]$ such that $R = \bigcup_{i=1}^{k} [\mathcal{L}(\mathcal{A}_{(i,1)}) \times \cdots \times \mathcal{L}(\mathcal{A}_{(i,n)})]$. The REG separability

[1] In [30], the authors consider Boolean combinations of membership constraints. Our results can be extended to handle such formulas. In [30], they also consider constraints of the form $x = t$. Such constraints can be encoded using our relational constraints.

problem for string constraints consists in checking for two given string constraints Ψ and Ψ' over the string variables $x_1, \ldots, x_n$ whether there is a regular set $R \subseteq (\Sigma^*)^n$ such that $\mathcal{L}(\Psi) \subseteq R$ and $R \cap \mathcal{L}(\Psi') = \emptyset$.

The regular separability problem is undecidable in general. This can be seen as an immediate corollary of the fact that the satisfiability problem of string constraints is undecidable [35, 14] even for a simple formula of the form $(x, x) \in \mathcal{R}(\mathcal{T})$ where $\mathcal{T}$ is a transducer and x is a string variable. To see why, consider Ψ to be $(x, x) \in \mathcal{R}(\mathcal{T})$ and Ψ' such that $\mathcal{L}(\Psi') = \Sigma^*$. It is easy to see that Ψ' and Ψ are separable by a regular set iff Ψ is unsatisfiable. In the following, we show a stronger result, namely that this undecidability still holds even for REG separability between two SL formulas.

▶ **Theorem 2.** *The* REG *separability problem is undecidable even for SL string constraints.*

4 PTL-Separability of String Constraints

Given the undecidability of REG separability, we focus on the separability problem using piece-wise testable languages (PTL). We show that the problem is in general undecidable and then we show its decidability in the case of SL formulas. The undecidability proof is exactly the same as in the case of the REG separability (since Σ^* is a PTL) while the decidability proof is done by reduction to its corresponding problem for the class of Ordered Multi Pushdown Automata (OMPA) [7, 11] (which we show its decidability). In the rest of this section, we first recall the definition of PTL and extend it to n-tuples of words. Then, we define the class of OMPAs and show the decidability of its separability problem by PTL. Finally, we show the decidability of the separability problem for SL formulas by PTL.

Piece-wise testable languages. Let Σ be an alphabet. A piece-language is a regular language of the form $\Sigma^* a_1 \Sigma^* a_2 \Sigma^* \ldots \Sigma^* a_k \Sigma^*$ where $a_1, a_2, \ldots, a_k \in \Sigma$. The class of piecewise testable languages (PTL) is defined as a finite Boolean combination of piece languages [41]. We can define PTL for an n-tuple alphabet with $n \in \mathbb{N}$, as follows: The class of PTL over n-tuple words (denoted n-PTL) is defined as the finite Boolean combination of languages of the form $(\Sigma^*)^n \mathbf{v}_1 (\Sigma^*)^n \cdots (\Sigma^*)^n \mathbf{v}_k (\Sigma^*)^n$ where $\mathbf{v}_i \in (\Sigma_\epsilon)^n$ for all $i \in [1, k]$.

Ordered Multi Pushdown Automata. Let Σ be a finite alphabet and $n \geq 1$ a natural number. Ordered multi-pushdown automata extend the model of pushdown automata with multiple stacks. An n-Ordered Multi Pushdown Automaton (OMPA or n-OMPA) is a tuple $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, Q_0, F)$ where (1) Q, Q_0 and F are finite sets of states, initial states and final states, respectively, (2) Γ is the stack alphabet and it contains the special symbol $\perp$, and (3) δ is the transition relation. OMPA are restricted in a sense that pop operations are only allowed from the first non-empty stack. A transition in δ is of the form $(q, \perp, \ldots, \perp, A_j, \epsilon, \ldots, \epsilon) \to^a (q', \gamma_1, \ldots, \gamma_n)$ where $A_j \in \Gamma_\epsilon$ represents the symbol that will be popped from the stack j on reading the input symbol $a \in \Sigma_\epsilon$, and $\gamma_i \in \Gamma^*$ represents the sequence of symbols which is going to be pushed on the stack i. The condition that $A_1 = \ldots = A_{j-1} = \perp$ (resp. $A_{j+1} = \ldots = A_n = \epsilon$) corresponds to the fact that the stacks $1, \ldots, j-1$ (resp. $j+1, \ldots, n$) are required to be empty (resp. inaccessible).

A configuration of $\mathcal{A}$ is of the form $(q, w, \alpha_1, \ldots, \alpha_n)$ where $q \in Q$, $w \in \Sigma^*$ and $\alpha_1, \ldots, \alpha_n \in (\Gamma \setminus \{\perp\})^* \cdot \{\perp\}$. The transition relation $\to$ between the set of configurations of $\mathcal{A}$ is defined as follows: Given two configurations $(q, w, \alpha_1, \ldots, \alpha_n)$ and $(q', w', \alpha'_1, \ldots, \alpha'_n)$, we have $(q, w, \alpha_1, \ldots, \alpha_n) \to (q', w', \alpha'_1, \ldots, \alpha'_n)$ iff there is a transition $(q, A_1, \ldots, A_n) \to^a (q', \gamma_1, \ldots, \gamma_n) \in \delta$ such that $w = aw'$ and $\alpha'_i = \gamma_i u_i$ where $\alpha_i = A_i u_i$ for all $i \in [1, n]$. We

use $\rightarrow^*$ to denote the transitive and reflexive closure of $\rightarrow$. A word $w \in \Sigma^*$ is accepted by $\mathcal{A}$ if there exists a sequence of configurations $c_1, \ldots, c_m$ such that: (1) c_1 is of the form $(q_0, w, \bot, \ldots, \bot)$, with $q_0 \in Q_0$, (2) c_m is of the form $(q_f, \epsilon, \bot, \ldots, \bot)$, with $q_f \in F$, and (3) $c_i \rightarrow c_{i+1}$ for all $i \in [1, m-1]$. The language of $\mathcal{A}$ (denoted by $\mathcal{L}(\mathcal{A})$) is defined as the set of words accepted by $\mathcal{A}$. The languages accepted by OMPA are referred to as OMPL.

In the following, we show that the separability problem for OMPL by PTL is decidable. As a first step, we show that the class of OMPL forms a *full trio* [23, 19]. We first recall the definition of a full-trio. Let L be a language over an alphabet A, and let $B \subseteq A$. The B-projection of a word $w \in A^*$ is the longest scattered subword containing only symbols from B. For example, if $A = \{a, b, c\}$, $B = \{b, c\}$, then the B-projection of $w = ababac$ is bbc. The B-upward closure of L is the set of all words that can be obtained by taking a word in L and padding it with symbols from B. For example, if $L = \{w\}$ for w as above, then the B-upward closure of L is the set $B^*aB^*bB^*aB^*bB^*aB^*cB^*$. A class of languages $\mathcal{C}$ is a full trio if it is effectively closed under (1) B-projection for every finite alphabet B, (2) B-upward closure for every finite alphabet B, and (3) intersection with regular languages.

▶ **Lemma 3.** *The class of OMPLs forms a full trio.*

To connect the PTL separability problem of SL string constraints to that of OMPL, we first use lemma 4. Lemma 4 states that the PTL separability problem for OMPL is equivalent to the *diagonal problem* for OMPL. We recall the diagonal problem [19]. Fix a class of languages $\mathcal{C}$ as above and a language $L \in \mathcal{C}$ over alphabet $\Sigma = \{a_1, \ldots, a_n\}$. Assume an ordering $a_1 < \cdots < a_n$ on Σ. For $a \in \Sigma$ and $w \in L$, let $\#_a(w)$ denote the number of occurrences of a in w. The *Parikh image* of w is the n-tuple $(\#_{a_1}(w), \ldots, \#_{a_n}(w))$. The *Parikh image* of L is the set of all Parikh images of words in L. An n-tuple $(m_1, \ldots, m_n) \in \mathbb{N}^n$ is dominated by another n-tuple $(d_1, \ldots, d_n)$ iff $m_i \leq d_i$ for all $1 \leq i \leq n$. The *diagonal problem* for $\mathcal{C}$ is the decision problem, which, given as input, a language L from $\mathcal{C}$ asks whether each n-tuple $(m, \ldots, m) \in \mathbb{N}^n$ is dominated by some Parikh image of L.

▶ **Lemma 4.** *The* PTL*-separability and diagonal problems are equivalent for OMPLs.*

Proof. This equivalence has been shown for full trios in [19] and by Lemma 3, OMPLs form a full trio. ◀

▶ **Lemma 5.** *Each language L in OMPL has a semilinear Parikh image and its representation can be effectively computable.*

▶ **Theorem 6.** *Given two OMPAs $\mathcal{A}_1$ and $\mathcal{A}_2$, checking whether there is a* PTL *L such that $\mathcal{L}(\mathcal{A}_1) \subseteq L$ and $L \cap \mathcal{L}(\mathcal{A}_2) = \emptyset$ is decidable.*

Proof. The proof follows from Lemmas 5, 4 and [19], from where we know that the diagonal problem is decidable for classes of languages having effectively semilinear Parikh images. ◀

▶ Remark 7. For the case of 1−OMPA, the PTL separability problem is already known to be decidable [19] but its complexity is still an open problem.

4.1 From SL formula to OMPA

In the following, we show that the n-PTL separability problem for SL formulas can be reduced to the PTL separability problem for OMPLs. To that aim, we proceed as follows: First, we show how to encode an n-tuple of words ($\in (\Sigma^*)^n$) as a word over $(\Sigma \cup \{\#\})^*$. Then, we show how to encode the set of solutions of an atomic relational constraint $(x, t) \in \mathcal{R}(\mathcal{T})$ using

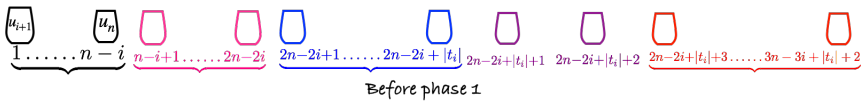
the stacks of an OMPA. Finally, we construct an OMPA that accepts exactly the language of a given SL formula Ψ. This construction will make use of the constructed OMPAs that encode the set of atomic relational constraints appearing in Ψ. Let Σ be an alphabet.

Encoding an n-tuple of words. Let n be a natural number. We assume w.l.o.g. that the special symbol $\#$ does not belong to Σ. We define the function `Encode` that maps any n-tuple word $\mathbf{w} = (w_1, \ldots, w_n) \in (\Sigma^*)^n$ to the word $w_1 \# w_2 \# \cdots \# w_n$.

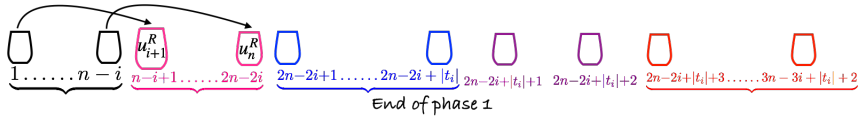
From SL atomic relational constraints to OMPAs. Let $x_1, x_2, \ldots, x_n$ be a sequence of string variables. Let P_i be a relational constraint of the form $(x_i, t_i) \in \mathcal{R}(\mathcal{T}_i)$ such that if a variable x_j is appearing in the term t_i, then $j > i$. In the following, we show that we can construct an OMPA $\mathcal{A}_i$ with $(3n+|t_i|+2-3i)$ stacks such that if $\mathcal{A}_i$ starts with a configuration where the first $(n-i)$ stacks contain, respectively, the evaluations $\eta(x_{i+1}), \ldots, \eta(x_n)$ (and all the other stacks are empty), then it can compute an evaluation $\eta(x_i)$ of the variable x_i such that: (1) $(\eta(x_i), \eta(t_i)) \in \mathcal{R}(\mathcal{T}_i)$ and the evaluations $\eta(x_i), \ldots, \eta(x_n)$ are stored in the last $n-i+1$ stacks of $\mathcal{A}_i$. Such an OMPA $\mathcal{A}_i$ will be used as a gadget when constructing the OMPA $\mathcal{A}$ that accepts exactly the language of a given SL formula Ψ.

▶ **Lemma 8.** *We can construct an OMPA $\mathcal{A}_i = (Q_i, \Sigma, \{\bot\} \cup \Sigma, \delta_i, \{q_i^{init}\}, \{q_i^{final}\})$ with $(3n + |t_i| + 2 - 3i)$-stacks such that for every $u_i, \ldots, u_n \in \Sigma^*$, we have $(q_i^{init}, \epsilon, u_{i+1}\bot, \ldots, u_n\bot, \bot, \ldots, \bot) \to^* (q_i^{final}, \epsilon, \bot, \ldots, \bot, u_i\bot, u_{i+1}\bot, \ldots, u_n\bot)$ iff $(\eta(x_i), \eta(t_i)) \in \mathcal{R}(\mathcal{T}_i)$ with $\eta(x_j) = u_j$ for all $j \in [i, n]$.*

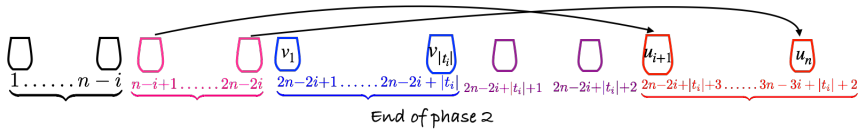
Proof. In the proof, we omit the input ϵ from the OMPA configurations, and only write the state, and stack contents. Let us assume that the string term t_i is of the form $y_1 y_2 \cdots y_{|t_i|}$. Observe that $y_j \in \{x_{i+1}, \ldots, x_n\}$. The OMPA $\mathcal{A}_i$ proceeds in phases starting from the configuration $(q_i^{init}, u_{i+1}\bot, \ldots, u_n\bot, \bot, \ldots, \bot)$. To begin, stacks 1 to $n-i$ contain $u_{i+1}, \ldots, u_n$, the evaluations of $x_{i+1}, \ldots, x_n$, and all other stacks are empty. The computation proceeds in 4 phases. The stacks indexed $1, \ldots, n-i$ and $n-i+1, \ldots, 2n-2i$ will be used in the first phase below. The second phase uses stacks indexed $n-i+1, \ldots, 2n-2i$ and $2n-2i+1, \ldots, 2n-2i+|t_i|$ along with the last $n-i$ stacks indexed $2n-2i+|t_i|+3$ to $3n-3i+|t_i|+2$. In the third phase, stacks indexed $2n-2i+1, \ldots, 2n-2i+|t_i|, 2n-2i+|t_i|+1$ are used. In the last phase, stacks indexed $2n-2i+|t_i|+1$ and $2n-2i+|t_i|+2$ are used. At the end of the 4 phases, stacks indexed $2n-2i+|t_i|+2, \ldots, 3n-3i+|t_i|+2$ hold the evaluations of $x_i, x_{i+1}, \ldots, x_n$, and all other stacks are empty.



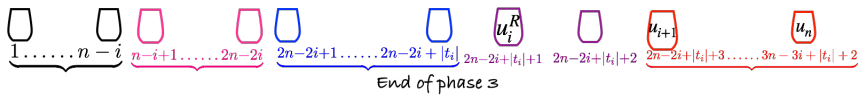
Phase 1. The OMPA $\mathcal{A}_i$ pops the symbols, one by one, from the first $(n-i)$-stacks $1, \ldots, n-i$ and pushes them into the stacks from index $(n-i+1)$ to $(2n-2i)$, respectively. At the end of this phase, the new configuration of the OMPA $\mathcal{A}_i$ is $(q_i^{init}, \bot, \ldots, \bot, u_{i+1}^R\bot, \ldots, u_n^R\bot, \bot, \ldots, \bot)$. That is, stacks $n-i+1, \ldots, 2n-2i$ have $u_{i+1}^R, \ldots, u_n^R$, while all other stacks are empty.



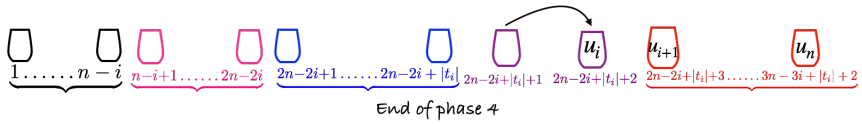
Phase 2. We do two things. (1) the contents of the $n-i$ stacks $n-i+1,\ldots,2n-2i$ are moved (in reverse) into the $n-i$ stacks $2n-2i+|t_i|+3,\ldots,3n-3i+|t_i|+2$. This results in the stacks $2n-2i+|t_i|+3,\ldots,3n-3i+|t_i|+2$ containing $u_{i+1},\ldots,u_n$. (2) If y_j appearing in t_i is the variable $x_{i+\ell}$, then the content of stack $n-i+\ell$ (with $n-i+1\leq n-i+\ell\leq 2n-2i$) is also moved (in reverse) to stack $2n-2i+j$, $1\leq j\leq |t_i|$. This results in stack $2n-2i+j$ containing $u_{i+\ell}$. Thus, at the end of (1), (2), the stacks $n-i+1,\ldots,2n-2i$ are empty, the stack $2n-2i+|t_i|+\ell+2$ contains $u_{i+\ell}$, the evaluation of $x_{i+\ell}$ for $\ell\geq 1$, while stack $2n-2i+k$ for $1\leq k\leq |t_i|$ contains u_{i+m} if $y_k=x_{i+m}$. The two stacks $2n-2i+|t_i|+1$ and $2n-2i+|t_i|+2$ are empty at the end of this phase. Stack contents of $2n-2i+k$, $1\leq k\leq |t_i|$ are referred to as v_k in the figure.



Phase 3. The OMPA $\mathcal{A}_i$ mimics the transducer $\mathcal{T}_i$. The current state of $\mathcal{A}_i$ is the same as the current state of the simulated transducer. Each transition of $\mathcal{T}_i$ of the form $(q,(a,b),q')$ is simulated by (1) moving the state of $\mathcal{A}_i$ from q to q', (2) pushing the symbol a into the stack $(2n-2i+|t_i|+1)$, and (3) popping the symbol b from the first non-empty stack having an index between $2n-2i+1$ to $2n-2i+|t_i|$. Recall that the stacks $2n-2i+1$ to $2n-2i+|t_i|$ contain the evaluations of $y_1,\ldots,y_{|t_i|}$, for $(\eta(x_i),\eta(y_1).\eta(y_2).\ldots\eta(y_{|t_i|}))\in\mathcal{R}(\mathcal{T}_i)$. When the current state of $\mathcal{A}_i$ is in a final state of $\mathcal{T}_i$ and the stacks from index $2n-2i+1$ to $2n-2i+|t_i|$ are empty, then we know that $\eta(y_1)\ldots\eta(y_{|t_i|})$ is indeed related by $\mathcal{T}_i$ on $\eta(x_i)$. Then $\mathcal{A}_i$ changes its state to q_i^{final}. Observe that, in case $\eta(y_1)\ldots\eta(y_{|t_i|})$ does not belong to the domain of $\mathcal{T}_i$, then the outgoing transition is not defined.



The Last Phase. At the end of the third phase, the current configuration of $\mathcal{A}_i$ is $(q_i^{final},\bot,\ldots,\bot,u_i^R,\bot,u_{i+1}\bot,\ldots,u_n\bot)$ such that $(u_i,v_1\cdots v_{|t_i|})\in\mathcal{R}(\mathcal{T}_i)$: that is, the last $n-i$ stacks $2n-2i+|t_i|+3,\ldots,3n-3i+|t_i|+2$ contain $u_{i+1},\ldots,u_n$, and stack $2n-2i+|t_i|+1$ contains the reverse of u_i. Then, $\mathcal{A}_i$ pops, one-by-one, the symbols from the $(2n-2i+|t_i|+1)$-th stack and pushes them, in the reverse order, into the stack $(2n-2i+|t_i|+2)$. Thus, the new configuration of $\mathcal{A}_i$ is of the form $(q_i^{final},\bot,\ldots,\bot,u_i\bot,u_{i+1}\bot,\ldots,u_n\bot)$ such that $(u_i,v_1\cdots v_{|t_i|})\in\mathcal{R}(\mathcal{T}_i)$ where $v_j=u_\ell$ if $y_j=x_\ell$ for $1\leq j\leq |t_i|$.



◀

From SL formula to OMPAs. In the following, we first construct an OMPA that accepts the encoding of the set of solutions of an SL formula.

▶ **Lemma 9.** *Given an SL formula Ψ, with $x_1,\ldots,x_n$ as its set of variables, it is possible to construct an OMPA $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A})=\texttt{Encode}(\mathcal{L}(\Psi))$.*

Proof. Let us assume that Ψ is of the form $\bigwedge_{i=1}^{n} x_i \in \mathcal{L}(\mathcal{A}_i) \wedge \bigwedge_{i=1}^{k} \varphi_i$ where $\varphi_1, \ldots, \varphi_k$ are relational constraints such that φ_i is of the form $(x_i, t_i) \in \mathcal{R}(\mathcal{T}_i)$. The OMPA $\mathcal{A}$ will have $(n - k + \sum_{i=1}^{k}(2n - 2i + 2 + |t_i|))$ stacks. $\mathcal{A}$ first guesses an evaluation for the variables $x_{k+1}, \ldots, x_n$ in the first $n - k$ stacks and then starts simulating the OMPA $\mathcal{A}_k$ (see Lemma 8 for the definition of $\mathcal{A}_k$) in order to compute a possible evaluation of the variable x_k such that the relational constraint $(x_k, t_k) \in \mathcal{R}(\mathcal{T}_k)$ holds for that evaluation. After this step, the stacks from index $(2n-2k+|t_k|+2)$ to $(3n-3k+|t_k|+2)$ contain the evaluation of the string variables $x_k, \ldots, x_n$, and all remaining stacks are empty. Now $\mathcal{A}$ can start the simulation of the OMPA $\mathcal{A}_{k-1}$ (Lemma 8) in order to compute a possible evaluation of the variable x_{k-1} such that $(x_k, t_k) \in \mathcal{R}(\mathcal{T}_k) \wedge (x_{k-1}, t_{k-1}) \in \mathcal{R}(\mathcal{T}_{k-1})$ holds for that evaluation. At the start of the simulation of $\mathcal{A}_{k-1}$ by $\mathcal{A}$, the $n - k + 1$ stacks (indexed $(2n - 2k + |t_k| + 2)$ to $(3n-3k+|t_k|+2)$) contain the evaluations of $x_k, \ldots, x_n$, and the next $2n-2(k-1)+|t_{k-1}|+2$ stacks are used to simulate phases 2-4 of $\mathcal{A}_{k-1}$. At the end of this, the $n - k + 2$ stacks backwards from the stack indexed $(3n - 3k + |t_k| + 2) + 2n - 2(k - 1) + |t_{k-1}| + 2$ contain the evaluations of $x_{k-1}, \ldots, x_n$. Now, $\mathcal{A}$ simulates $\mathcal{A}_{k-2}, \ldots, \mathcal{A}_n$ in the same way. At the end of this simulation phase, the last n-stacks of $\mathcal{A}$ contain an evaluation of the string variables $x_1, \ldots, x_n$ that satisfies $\bigwedge_{i=1}^{k} \varphi_i$. Let us assume that the current configuration of $\mathcal{A}$ at the end of this is of the form $(q^{final}, \bot, \ldots, \bot, u_1\bot, u_2\bot, \ldots, u_n\bot)$. Then, $\mathcal{A}$ starts popping, one-by-one, from the n-th stack from the last and outputs the read stack symbol $\in \Sigma$ while ensuring that the evaluation u_1 of x_1 belongs to $\mathcal{L}(\mathcal{A}_1)$. When the n-th stack from the last is empty, $\mathcal{A}$ outputs the special symbol $\#$. Then, $\mathcal{A}$ does the same for the i-th stack from last, with $i \in [1, n-1]$, which contains the evaluation of x_{i+1}. If $\mathcal{A}$ succeeds to empty all stacks, then this means that the evaluation η which associates to the variable x_i, the word u_i for all $i \in [1, n]$ satisfies $\bigwedge_{i=1}^{n} x_i \in \mathcal{L}(\mathcal{A}_i)$. Hence, $u_1\#u_2\#\cdots\#u_n$ is accepted by $\mathcal{A}$ iff $\eta \models \Psi$. ◀

The following lemma shows that the PTL-separability problem for SL formulas can be reduced to the PTL-separability problem for OMPLs.

▶ **Lemma 10.** *Let Ψ_1 and Ψ_2 be two SL formulae with $x_1, \ldots, x_n$ as their set of variables. Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two OMPAs such that $\mathcal{L}(\mathcal{A}_1)$=`Encode`$(\mathcal{L}(\Psi_1))$ and $\mathcal{L}(\mathcal{A}_2)$=`Encode`$(\mathcal{L}(\Psi_2))$. Ψ_1, Ψ_2 are n-PTL separable iff $\mathcal{A}_1, \mathcal{A}_2$ are PTL-separable.*

As an immediate corollary of Theorem 6, Lemma 10, we obtain our main result:

▶ **Theorem 11.** *The n-PTL separability problem of SL formulae is decidable.*

5 PosPTL-Separability of String Constraints

In this section, we address the separability problem for string constraints by a sub-class of PTL, called positive piece-wise testable languages (PosPTL). A language is in PosPTL iff it is defined as a finite positive Boolean combination (i.e., union and intersection but no complementation) of piece-languages. Given a natural number $n \in \mathbb{N}$, this definition can naturally be extended to n-tuples of words in the straightforward manner (as in the case of PTL) to obtain the class of n-PosPTL. In the following, we first provide a necessary and sufficient condition for the n-PosPTL separability problem of any two languages. Following result follows easily from PosPTL being upward closed.

▶ **Theorem 12.** *Two languages I and E are n-PosPTL separable iff $I{\uparrow} \cap E = \emptyset$ iff $I \cap E{\downarrow} = \emptyset$.*

The rest of this section is structured as follows: First, we show that the PosPTL separability is decidable for OMPLs; in the particular case of CFLs, this problem is PSPACE-complete. Then, we use the encoding of SL formulas to OMPAs (as defined in Section 4), and show that n-PosPTL separability of SL formulas reduces to the PosPTL-separability of corresponding OMPLs. Finally, we consider the PosPTL-separability problem for a subclass of SL formulas, called *right sided* SL formulas. We show that the PosPTL separability problem for this subclass is PSPACE-HARD and is in EXPSPACE.

5.1 PosPTL-Separability of SL formulas

First, we show that the PosPTL separability for OMPLs is decidable.

▶ **Theorem 13.** PosPTL *separability of OMPLs is decidable.*

Proof. Consider $\mathcal{C}$ to be the class of OMPLs in Theorem 12. Let I and E be two languages belonging to $\mathcal{C}$ as stated in Theorem 12. Then, the set $\mathsf{min}(I\uparrow)$ is effectively computable as an immediate consequence of the *Generalized Valk-Jantzen* construction [6]. The main idea behind this construction is to start with an empty minor set M (so to begin, $M \subseteq I$) and keep adding new words $w \in I$ to M if w is not already in $M\uparrow$. Before adding a new word, we need to test that $I \cap \overline{M\uparrow} \neq \emptyset$ (the complement of $M\uparrow$ intersects with I). This test is decidable since (i) OMPLs are closed under intersection with regular languages and (ii) the emptiness problem for OMPA is decidable [7]. At each step, we remove all the non-minimal words from M (since M is finite). The algorithm terminates due to the Higman's Lemma [24] (the minor of an upward closed set is finite). When the algorithm terminates, $I \subseteq M\uparrow$ and thus $I\uparrow \subseteq M\uparrow$. By construction, $M \subseteq I$ and $M\uparrow \subseteq I\uparrow$. Thus, $M\uparrow = I\uparrow$. Since M is a minor set, we have $\mathsf{min}(I\uparrow) = M$. Using (i) and (ii), we obtain the decidability of checking the emptiness of $I\uparrow \cap E$, and thus PosPTL separability of OMPL is decidable. ◀

As mentioned in section 4, the complexity of PTL-separability for 1-OMPL is open; however, we show that the PosPTL separability problem for 1-OMPL is PSPACE-COMPLETE.

▶ **Theorem 14.** *The* PosPTL*-separability for CFLs is* PSPACE*-*COMPLETE*.*

For the decidability of the n-PosPTL separability of SL formulas, we use the encoding of SL formulas to OMPAs (as defined in section 4), and show that the n-PosPTL separability of SL formulas reduces to the PosPTL separability of their corresponding OMPLs. The decidability of the n-PosPTL separability of SL formulas follows from Theorem 13.

▶ **Lemma 15.** *Given two SL formulas Ψ and Ψ', with $x_1, \ldots, x_n$ as their set of variables. Let $\mathcal{A}$ and $\mathcal{A}'$ be two OMPAs such that $\mathcal{L}(\mathcal{A}) = \texttt{Encode}(\mathcal{L}(\Psi))$ and $\mathcal{L}(\mathcal{A}') = \texttt{Encode}(\mathcal{L}(\Psi'))$. Then, Ψ and Ψ' are separable by an n-*PosPTL *iff $\mathcal{A}$ and $\mathcal{A}'$ are separable by a* PosPTL*.*

As an immediate corollary of Lemma 13 and 15, we obtain the following theorem:

▶ **Theorem 16.** *The n-*PosPTL *separability problem of SL formulae is decidable.*

5.2 PosPTL-Separability of Right-sided SL formula

Unfortunately, the proof of Theorem 16 does not allow us to extract any complexity result. Therefore, we consider in this subsection a useful fragment of SL formulas, called *right-sided* SL formulas. Roughly speaking, an SL formula Ψ is *right-sided* iff any variable appearing on the right-side of a relational constraint can not appear on the left-side of any relational constraint. Let us formalize the notion of right-sided SL formulas. Let us assume an SL

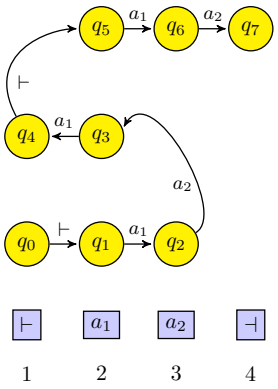
formula Ψ of the form $\bigwedge_{i=1}^{n} x_i \in \mathcal{L}(\mathcal{A}_i) \wedge \bigwedge_{i=1}^{k} (x_i, t_i) \in \mathcal{R}(\mathcal{T}_i)$ with $x_1, \ldots, x_n$ as set of variables. Then, Ψ is said to be *right-sided* if none of the variables $x_1, \ldots, x_k$ appear in any of $t_1, \ldots, t_k$. We call $x_{k+1}, \ldots, x_n$ (resp. $x_1, \ldots, x_k$) *independent* (resp. *dependent*) variables. Observe that the class of SL formulas with functional transducers can be rewritten as right-sided SL formulas (detailed proof can be found at [4]). A transducer $\mathcal{T}$ is functional if for every word w, there is at most one word w' such that $(w', w) \in \mathcal{R}(\mathcal{T})$ ($\mathcal{T}$ computes a function). An example of a functional transducer is the one implementing the identity relational constraint (allowing to express the equality $x = t$).

In the following, we show that the PosPTL-separability problem for right-sided SL formulas is in EXPSPACE. To show this result, we will reduce the PosPTL-separability problem for right-sided SL formulas to its corresponding problem for two-way transducers.

Two way transducers. Let Σ be a finite input alphabet and let $\vdash, \dashv$ be two special symbols not in Σ. We assume that every input string $w \in \Sigma^*$ is presented as $\vdash w \dashv$, where $\vdash, \dashv$ serve as left and right delimiters that appear nowhere else in w. We write $\Sigma_{\vdash\dashv} = \Sigma \cup \{\vdash, \dashv\}$. A two-way automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ has a finite set of states Q, subsets $I, F \subseteq Q$ of initial and final states and a transition relation $\delta \subseteq Q \times \Sigma_{\vdash\dashv} \times Q \times \{-1, 1\}$. The -1 represents that the reading head moves to left after taking the transition while a 1 represents that it moves to right. The reading head cannot move left when it is on $\vdash$, and cannot move right when it is on $\dashv$. A configuration of $\mathcal{A}$ on reading $w' = \vdash w \dashv$ is represented by (q, i) where $q \in Q$ and i is a position in the input, $1 \leq i \leq |w| + 2$, which will be read in state q. An initial configuration is of the form $(q_0, 1)$ with $q_0 \in I$ and the reading head on $\vdash$. If $w' = w_1 a w_2$ and the current configuration is $(q, |w_1| + 1)$, and $(q, a, q', -1) \in \delta$, then there is a transition from the configuration $(q, |w_1| + 1)$ to $(q', |w_1|)$ (hence $a \neq \vdash$). Likewise, if $(q, a, q', 1) \in \delta$, we obtain a transition from $(q, |w_1| + 1)$ to $(q', |w_1| + 2)$. A run of $\mathcal{A}$ on reading $\vdash w \dashv$ is a sequence of transitions; it is accepting if it starts in an initial configuration and ends in a configuration of the form $(q, |w| + 2)$ with $q \in F$ and the reading head on $\dashv$. The language of $\mathcal{A}$ (denoted $\mathcal{L}(\mathcal{A})$) is the set of all words $w \in \Sigma^*$ s.t. $\mathcal{A}$ has an accepting run on $\vdash w \dashv$.

We extend the definition of a two-way automaton $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ into a two-way transducer (2NFT) $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, I, F)$ where Γ is a finite output alphabet. The transition relation is defined as a *finite* subset $\delta \subseteq Q \times \Sigma_{\vdash\dashv} \times Q \times \Gamma^* \times \{-1, 1\}$. The output produced on each transition is appended to the right of the output produced so far. $\mathcal{A}$ defines a relation $\mathcal{R}(\mathcal{A}) = \{(w, u) \mid w \text{ is the output produced on an accepting run of } u\}$. The acceptance condition is the same as in two-way automata. Sometimes, we use the macro-notation $(p, a, q, \alpha, 0)$ to denote a sequence of consecutive transitions (p, a, s, α, d) and (s, b, q, ϵ, d') in δ with $d + d' = 0$, $b \in \Sigma_{\vdash\dashv}$ and s is an extra intermediary state of $\mathcal{A}$ that is not used anywhere else (and that we omit from the set of states of $\mathcal{A}$).

PosPTL separability of 2NFT. In the following, we study the PosPTL separability of 2NFT. We define here the notion of *visiting sequences* (similar to *crossing sequences* of 2NFT [8]), which will be used in the proof of Lemma 17. Let $w = \vdash a_1 \ldots a_n \dashv$ be an input word and let ρ be a run of the 2NFT on w. A visiting sequence at a position x of a word w, in a run ρ of w captures the states visited in order in the run, each time the reading head is on position x, along with the information pertaining to the direction of the outgoing transition from that state. For example, in run ρ, if position x is visited for the first time in state q, and the outgoing transition chosen in ρ from q during that visit had direction $+1$, then q^+ will be the first entry in the visiting sequence. For a run ρ, the visiting sequence at a position x is defined as the tuple $\rho|x = (q_1^{d_1}, q_2^{d_2}, \ldots, q_h^{d_h})$ of states that have, in order, visited position x



in ρ, and whose outgoing transitions had direction $d_1, \ldots, d_h$. In the example, the visiting sequence at position 2 is (q_1^+, q_3^-, q_5^+), while those at 1 and 3 respectively are (q_0^+, q_4^+) and (q_2^-, q_6^+).

▶ **Lemma 17.** *Given a* 2NFT $\mathcal{T}$*, if* (v, u) *is in* $\mathsf{min}(\mathcal{R}(\mathcal{T})\uparrow)$ *then* $|u|$ *and* $|v|$ *are of at most exponential length in the size of* $\mathcal{T}$*.*

Proof. In the following, we show that, if $(v, u) \in \mathsf{min}(\mathcal{R}(\mathcal{T})\uparrow)$, then $|u| \leq \mathsf{in}_{\max} = \sum_{i=1}^{|Q|} ((2|Q|)^i \cdot |\Sigma|)$ and $|v| \leq \mathsf{out}_{\max} = \sum_{i=1}^{|Q|}((2|Q|)^i \cdot |\Sigma| \cdot |Q| \cdot \gamma_{\max})$ where $\gamma_{\max}$ represents the maximum length of an output on any transition in $\mathcal{T}$. To show this result, we need to define normalized runs as follows: A run is *normalized* if it visits each state at most once on each position x. In the following, we show that (v, u) can be generated by a normalized run ρ. Assume that (v, u) is accepted by a run ρ' which is not normalized. Then, we will have, in ρ', two visits to some position x of the word in the same state q. After the first visit to position x in state q, the transducer has explored some positions till its second visit to position x in state q. This part does not produce any output since (v, u) is a minimal word. We can delete this explored part of the run in between, obtaining again, an accepting run, which reads u while producing v. For example, in the figure if we have $q_6 = q_2$, then we have another run without visiting positions 1, 2 for a second time. Observe that repeating this procedure will lead to a normalized run ρ accepting (v, u). The length of visiting sequences in a normalized run is $\leq |Q|$ and hence the number of visiting sequences is at most exponential in $|Q|$, precisely it is $\leq \sum_{i=1}^{|Q|} (2|Q|)^i$.

Suppose $|u| > \mathsf{in}_{\max}$. Then there exists a visiting sequence which is repeated on reading the same input symbol in the accepting run of u, at positions $i \neq j$. By deleting the part between the ith and $(j-1)$th position, we again obtain an accepting run over a word u', which is a strict subword of u, and whose output v' is also a subword (may not be strict) of v, a contradiction to $(v, u) \in \mathsf{min}(\mathcal{R}(\mathcal{T})\uparrow)$. Now suppose $|u| \leq \mathsf{in}_{\max}$ but $|v| > \mathsf{out}_{\max}$. We saw that in the normalized run, each visiting sequence has length at most $|Q|$. Then, since $|u| \leq \mathsf{in}_{\max}$, on reading each position of u, at most $(|Q|)\gamma_{\max}$ symbols can be produced. Hence, we have $|v| \leq (|Q|) \cdot \gamma_{\max} \cdot |u|$. ◀

From Theorem 12 and Lemma 17, the following result holds:

▶ **Lemma 18.** *The 2-*PosPTL *separability problem for* 2NFT *is in* ExpSpace*.*

Proof. Using Theorem 12, we know that $\mathcal{R}(\mathcal{T}_1)\uparrow \cap \mathcal{R}(\mathcal{T}_2) = \emptyset$ iff $\mathcal{T}_1$ and $\mathcal{T}_2$ are 2-PosPTL separable. Here is an NExpSpace algorithm.

(1) Guess some (v, u) s.t. the lengths of v, u are at most as given by the proof of Lemma 17.
(2) Check if $(v, u) \in \mathcal{R}(\mathcal{T}_1)$. If yes, then do (3). Else exit.
(3) Check if $(v, u) \uparrow \cap \mathcal{R}(\mathcal{T}_2) \neq \emptyset$.

The guessed word (v, u) has exponential length in the size of $\mathcal{T}_1$. To check if $(v, u) \in \mathcal{R}(\mathcal{T}_1)$, we construct another transducer $\mathcal{T}_1'$ that first checks that its input word is u, then it comes back to $\vdash$ and starts simulating $\mathcal{T}_1$, while also keeping track, longer and longer prefixes of v. We then compare those prefixes with the output produced by $\mathcal{T}_1$. This gives rise to exponentially many states (maintaining prefixes of u and v) and we finish when $\mathcal{T}_1$ enters an accepting state, and at the same time, the produced word is v. Since $\mathcal{R}(\mathcal{T}_1') = \{(v, u)\} \cap \mathcal{R}(\mathcal{T}_1)$ by construction, checking if $(v, u) \in \mathcal{R}(\mathcal{T}_1)$ can be reduced to the emptiness problem of $\mathcal{T}_1'$. After this, we check the emptiness of $(v, u) \uparrow \cap \mathcal{R}(\mathcal{T}_2)$. This is done as follows. First, construct automata $\mathcal{A}_u, \mathcal{A}_v$ accepting languages $\{u\}\uparrow$ and $\{v\}\uparrow$ respectively. The number of states of $\mathcal{A}_u, \mathcal{A}_v$ are exponential in the number of states of $\mathcal{T}_1$, since the lengths of u, v have this bound. Then, we construct a transducer $\mathcal{T}_2'$ such that $\mathcal{R}(\mathcal{T}_2') = \{(v, u)\} \uparrow \cap \mathcal{R}(\mathcal{T}_2)$ in a similar manner as $\mathcal{T}_1'$. $\mathcal{T}_2'$ reads the input word while simulating $\mathcal{A}_u$. On entering an accepting state of $\mathcal{A}_u$, it comes back to $\vdash$. Then it simulates $\mathcal{T}_2$, and, on the outputs produced, simulates $\mathcal{A}_v$. If $\mathcal{A}_v$ enters an accepting state at the same time $\mathcal{T}_2$ accepts, then we are done. The state space of $\mathcal{T}_2'$ is exponential in the states of $\mathcal{T}_1$ and linear in the states of $\mathcal{T}_2$. Since $\mathcal{R}(\mathcal{T}_2') = \{(v, u)\} \uparrow \cap \mathcal{R}(\mathcal{T}_2)$, checking the emptiness of $(v, u) \uparrow \cap \mathcal{R}(\mathcal{T}_2)$ can be reduced to checking the emptiness problem of $\mathcal{T}_2'$. The emptiness problem for 2NFT is known to be PSPACE-COMPLETE [44]. Thus, in our case, the emptiness of $\mathcal{T}_1'$ and $\mathcal{T}_2'$ can be achieved in space exponential in $\mathcal{T}_1$. Since we can handle the second and third steps in exponential space, we obtain an NEXPSPACE algorithm. By Savitch's Theorem, we obtain the EXPSPACE complexity. ◀

From Right-sided SL formulas to 2NFT. Hereafter, we show how to encode the set of solutions of a right-sided SL formula using 2NFT. Let Σ be an alphabet and $\# \notin \Sigma$.

▶ **Lemma 19.** *Let Ψ be a right-sided SL formula over Σ, with $x_1, x_2, \ldots, x_n$ as its set of variables. Then, it is possible to construct, in polynomial time, a* 2NFT *$\mathcal{A}_\Psi$ such that $\mathcal{R}(\mathcal{A}_\Psi) = \{(u_1\#u_2\#\cdots\#u_n, w_1\#w_2\#\ldots\#w_n) | u_1\#u_2\#\cdots\#u_n \in \mathtt{Encode}(\mathcal{L}(\Psi))$ and $w_i = u_i$ if x_i is an independent variable $\}$.*

Proof. Let us assume that Ψ is of the form $\bigwedge_{i=1}^{n} y_i \in \mathcal{L}(\mathcal{A}_i) \wedge \bigwedge_{i=1}^{k} (y_i, t_i) \in \mathcal{R}(\mathcal{T}_i)$ with $y_1, \ldots, y_n$ is a permutation of $x_1, \ldots, x_n$. Let $\pi : [1, n] \to [1, n]$ be the mapping that associates to each index $i \in [1, n]$, the index $j \in [1, n]$ s.t. $x_i = y_j$ (or $x_i = y_{\pi(i)}$). We construct $\mathcal{A}_\Psi$ as follows: $\mathcal{A}_\Psi$ reads n words over Σ separated by $\#$ as input. We explain hereafter the working of $\mathcal{A}_\Psi$ when it produces the assignment for x_1 (the other variables are handled in similar manner).
• Assume that x_1 is a dependent variable. Let $\varphi_{\pi(1)} = (y_{\pi(1)}, t_{\pi(1)}) \in \mathcal{R}(\mathcal{T}_{\pi(1)})$, with $t_{\pi(1)} = x_{i_1} x_{i_2} \ldots x_{i_c}$ and $x_{i_j} \in \{y_{k+1}, y_{k+2}, \ldots, y_n\}$ for all j. First, $\mathcal{A}_\Psi$ reads x_{i_1} i.e. the first variable in $t_{\pi(1)}$. To read x_{i_1}, it skips $(i_1 - 1)$ many blocks separated by #s of the input, and comes to w_{i_1}. On the first symbol of w_{i_1}, $\mathcal{A}_\Psi$ starts mimicking transitions of $\mathcal{T}_{\pi(1)}$ from its initial state, while producing the same output as $\mathcal{T}_{\pi(1)}$. On the same output, $\mathcal{A}_\Psi$ mimics the transitions of $\mathcal{A}_{\pi(1)}$ starting from the initial state to check the membership constraint of $y_{\pi(1)}$. This can be done by a product construction between $\mathcal{A}_{\pi(1)}$ and $\mathcal{T}_{\pi(1)}$. For instance, $\mathcal{A}$ will have a transition $((p, q), a, (p', q'), b, 1)$ (resp. $((p, q), a, (p', q'), b, 0)$), if there are transitions $(p, (b, a), p')$ (resp. $(p, (b, \epsilon), p')$) in $\mathcal{T}_{\pi(1)}$ and (q, b, q') in $\mathcal{A}_{\pi(1)}$. If it reaches # or $\dashv$ in the input, it remembers the current states of $\mathcal{T}_{\pi(1)}$ and $\mathcal{A}_{\pi(1)}$, say (p_1, q_1) in its control state. Next, $\mathcal{A}_\Psi$ reads x_{i_2} in the input. To read x_{i_2}, $\mathcal{A}_\Psi$ moves to $\vdash$ and then changes direction.

As before it reaches x_{i_2} by skipping $(i_2 - 1)$ many #s, and starts reading the input (the first symbol of w_{i_2}) from the state (p_1, q_1) stored in the finite control. Transitions are similar to explained above. This procedure is repeated to read $x_{i_3} \dots x_{i_c}$. After reading x_{i_c}, if the next state contains the pair (p_c, q_c), where p_c (resp. q_c) is a final state of $\mathcal{T}_{\pi(1)}$ (resp. $\mathcal{A}_{\pi(1)}$), we can say that the output produced till now satisfies $\varphi_{\pi(1)}$ and $y_{\pi(1)} \in \mathcal{L}(\mathcal{A}_{\pi(1)})$. Observe that this procedure requires $|t_{\pi(1)}|$ reversals, and thus we need at most $|t_{\pi(1)}|$ copies of transducer $\mathcal{T}_{\pi(1)}$. Thus the number of states required are at most polynomial.

• Assume now that x_1 is an independent variable, then $\mathcal{A}_\Psi$ needs to read x_1. We need a single pass of the input which verifies if the first block corresponding to value of x_1 in input indeed satisfies its corresponding membership constraint. During this pass $\mathcal{A}_\Psi$ mimics transitions of $\mathcal{A}_{\pi(1)}$ starting from its initial states, and outputs the same letter as input.

The above procedure is repeated for all variables from x_2 to x_n. After each pass, $\mathcal{A}_\Psi$ moves to $\vdash$ and then changes direction. Irrespective of whether x_i is dependent or not, while going from x_i to x_{i+1}, $i \in [1, n-1]$, $\mathcal{A}_\Psi$ outputs a # as separator. From the description above, it can be seen that if x_i is independent, then its evaluation u_i given as the ith block of the input is equal to the output w_i, and if x_i is a dependent variable, then the output block w_i is the output of $\mathcal{T}_{\pi(i)}$. It is clear from the construction that $\mathcal{A}_\Psi$ requires at most polynomial states in input. More details can be found at [4]. ◀

Notice that the above construction of 2NFT relies on the right-sidedness: if a variable x_i appears in the output of $\mathcal{T}_i$ and also in the input of $\mathcal{T}_k$ for some k, then we will have to store the produced evaluation of x_i in order to use it later on when processing $\mathcal{T}_k$. However, there is no way to store the produced evaluation of x_i or compare it with its input evaluation. Next, we show that the PosPTL separability problem for right-sided formulas can be reduced to its corresponding problem for 2NFT.

▶ **Lemma 20.** *Let Ψ_1 and Ψ_2 be two right-sided SL formula, with $x_1, \dots, x_n$ as their set of variables. Let $\mathcal{A}_{\Psi_1}$ and $\mathcal{A}_{\Psi_2}$ be the two* 2NFT*s encoding, respectively, the set of solutions of Ψ_1 and Ψ_2 (as described in Lemma 19). Then, the two formulae Ψ_1 and Ψ_2 are separable by n-*PosPTL *iff $\mathcal{R}(\mathcal{A}_{\Psi_1})$ and $\mathcal{R}(\mathcal{A}_{\Psi_2})$ are separable by a 2-*PosPTL*.*

Proof. Let $\mathcal{R}(\mathcal{A}_{\Psi_1})$ and $\mathcal{R}(\mathcal{A}_{\Psi_2})$ be separable by a 2-PosPTL L. By definition, L is a Boolean combination (except complementation) of piece languages of words over the two tuple alphabet $(\Sigma \cup \{\#\})^2$. We can assume w.l.o.g. that L is the union of piece languages. This is possible since the intersection of two piece languages can be rewritten as a union of piece languages. Consider $L' = L \cap (R \times R)$, where R is a regular language consisting of words having exactly $(n-1)$ #s. We claim that L' can be rewritten as the union of languages of the form $[L_1 \# L_2 \# \dots \# L_n] \times [R_1 \# R_2 \# \dots \# R_n]$ where the L_is and R_is are piece languages over Σ, and that L' is also a separator of $\mathcal{R}(\mathcal{A}_{\Psi_1})$ and $\mathcal{R}(\mathcal{A}_{\Psi_2})$.

We prove this claim inductively. As a base case consider L to be a piece language $((\Sigma \cup \{\#\})^*)^2 (a_1, b_1)((\Sigma \cup \{\#\})^*)^2 \dots (a_m, b_m)((\Sigma \cup \{\#\})^*)^2$. Let S be a finite set containing only the *minimal* words of the form (w, w') such that $a_1 a_2 \dots a_m \preceq w$, $b_1 b_2 \dots b_m \preceq w'$, and the symbol # appears exactly $(n-1)$-times in w and w'. Thus $L \cap (R \times R) = \bigcup_{(a'_1 \dots a'_k, b'_1 \dots b'_\ell) \in S} \left[\Sigma^* a'_1 \Sigma^* a'_2 \dots a'_k \Sigma^*\right] \times \left[\Sigma^* b'_1 \Sigma^* b'_2 \dots b'_\ell \Sigma^*\right]$. So $L \cap (R \times R)$ is the union of piece languages of the form $[L_1 \# L_2 \# \dots \# L_n] \times [R_1 \# R_2 \# \dots \# R_n]$ where the L_is and R_is are piece languages over Σ. Now assume that L is of the form $L_1 \cup L_2$. It is easy to see that $L \cap (R \times R)$ is equivalent to $(L_1 \cap (R \times R)) \cup (L_2 \cap (R \times R))$. Thus we can use our induction hypothesis to show that $L \cap (R \times R)$ is the union of languages of the form $[L_1 \# L_2 \# \dots \# L_n] \times [R_1 \# R_2 \# \dots \# R_n]$ where L_i and R_is are piece languages over Σ.

Next we prove that L' is a separator of $\mathcal{R}(\mathcal{A}_{\Psi_1})$ and $\mathcal{R}(\mathcal{A}_{\Psi_2})$. Indeed if $(v,u) \in \mathcal{R}(\mathcal{A}_{\Psi_1})$, then $(v,u) \in R \times R$, by definition of $\mathcal{R}(\mathcal{A}_{\Psi_1})$. Since L is a separator, we have $(v,u) \in L$ and hence $(v,u) \in L'$. Suppose $(v,u) \in \mathcal{R}(\mathcal{A}_{\Psi_2}) \cap L'$, then $(v,u) \in L \cap \mathcal{R}(\mathcal{A}_{\Psi_1})$ since $(v,u) \in L'$, and $L' \subseteq L$, which is a contradiction with the assumption that L is a separator.

Now we are in a condition to provide n-PosPTL separator for $\mathcal{L}(\Psi_1)$ and $\mathcal{L}(\Psi_2)$, using L'. Given a language of the form $[L_1\#L_2\#\ldots\#L_n] \times [R_1\#R_2\#\ldots\#R_n]$ where L_i and R_is are piece languages, we associate to it an n-PosPTL equivalent to $((L_1 \cap R_1) \times (L_2 \cap R_2) \times \ldots \times (L_n \cap R_n))$: the idea is to generate the n dimensions in the n-PosPTL from the n #-separated blocks in two dimensions. This definition is extended in the straightforward manner to union of piece languages. Let K be the n-PosPTL associated to L'. K is indeed a separator of $\mathcal{L}(\Psi_1)$ and $\mathcal{L}(\Psi_2)$: Suppose $\mathbf{v} = (w_1,\ldots,w_n) \in \mathcal{L}(\Psi_1)$, then $(w_1\#\ldots\#w_n, w_1\#\ldots\#w_n) \in \mathcal{R}(\mathcal{A}_{\Psi_1})$ (from the definition of $\mathcal{A}_{\Psi_1}$). Since L' is a separator, $(w_1\#\ldots\#w_n, w_1\#\ldots\#w_n) \in L'$. By construction of K, $(w_1,\ldots,w_n) \in K$. Assume by contradiction $\mathbf{v} = (w_1,\ldots,w_n) \in \mathcal{L}(\Psi_2) \cap K$, then $(w_1\#\ldots\#w_n, w_1\#\ldots\#w_n) \in L'$. Since $L' \cap \mathcal{R}(\mathcal{A}_{\Psi_2}) = \emptyset$, then $(w_1\#\ldots\#w_n, w_1\#\ldots\#w_n) \notin \mathcal{R}(\mathcal{A}_{\Psi_2})$. By definition of $\mathcal{A}_{\Psi_2}$, if $(w_1,\ldots,w_n) \in \mathcal{L}(\Psi_2)$, then $(w_1\#\ldots\#w_n, w_1\#\ldots w_n) \in \mathcal{R}(\mathcal{A}_{\Psi_2})$. Hence contradiction.

For the other direction of the proof, assume the n-PosPTL S is a separator of $\mathcal{L}(\Psi_1)$ and $\mathcal{L}(\Psi_2)$. Then S can be rewritten as the union of $(L_1 \times L_2 \times \ldots \times L_n)$ where L_is are piece languages. Replace each n-piece language $(L_1 \times L_2 \times \ldots \times L_n)$ of S with the 2-piece language $(L_1'\#L_2'\#\ldots\#L_n') \times ((\Sigma \cup \{\#\})^*\#\ldots\#(\Sigma \cup \{\#\})^*)$, where $L_1' = (\Sigma \cup \{\#\})^* a_1 (\Sigma \cup \{\#\})^* \ldots a_n (\Sigma \cup \{\#\})^*$ if $L_1 = \Sigma^* a_1 \Sigma^* \ldots a_n \Sigma^*$. Denote the union of such languages by S'. It is a 2-PosPTL over $(\Sigma \cup \{\#\})$. We show that S' is a 2-PosPTL separator of $\mathcal{R}(\mathcal{A}_{\Psi_1})$ and $\mathcal{R}(\mathcal{A}_{\Psi_2})$. Let $\mathbf{(v,\ u)} = (v_1\#\ldots\#v_n, u_1\#\ldots\#u_n) \in \mathcal{R}(\mathcal{A}_{\Psi_1})$, then $(v_1,\ldots,v_n) \in \mathcal{L}(\Psi_1)$, and thus $(v_1,\ldots,v_n) \in S$ (since S is a separator). This implies that $\mathbf{(v,\ u)} \in S'$ by its construction. Suppose $\mathbf{(v,\ u)} = (v_1\#\ldots\#v_n, u_1\#\ldots\#u_n) \in \mathcal{R}(\mathcal{A}_{\Psi_2}) \cap S'$, then $(v_1, v_2, \ldots, v_n) \in \mathcal{L}(\Psi_2)$. Also $(v_1, v_2, \ldots, v_n) \in S$ by construction of S'. This leads to a contradiction that S is separator of $\mathcal{L}(\Psi_1)$ and $\mathcal{L}(\Psi_2)$. So S' is a 2-PosPTL separator of $\mathcal{R}(\mathcal{A}_{\Psi_1})$ and $\mathcal{R}(\mathcal{A}_{\Psi_2})$. ◀

▶ **Theorem 21.** *The n-PosPTL separability of right-sided SL formulas is in Expspace and is Pspace-Hard.*

Proof. Given two right-sided SL formulas Ψ_1 and Ψ_2, one can construct corresponding two way transducers $\mathcal{A}_{\Psi_1}$ and $\mathcal{A}_{\Psi_2}$ with polynomial states, as mentioned in Lemma 19. Thanks to Lemma 20, the n-PosPTL separability reduces to 2-PosPTL separability of $\mathcal{R}(\mathcal{A}_{\Psi_1})$ and $\mathcal{R}(\mathcal{A}_{\Psi_2})$. The 2-PosPTL separability of 2NFTs is in Expspace (Lemma 18). Hence n-PosPTL separability of SL formulae is also in Expspace. For the Pspace-Hard lower bound, we reduce the emptiness of k-NFA intersection to PosPTL separability of right sided SL. Let $\mathcal{A}_1,\ldots,\mathcal{A}_k$ be k-NFA. We want to decide if $\bigcap_{i=1}^k \mathcal{A}_i = \emptyset$. Let Ψ_1 be $\bigwedge_{i=1}^k x_i = x \wedge \bigwedge_{i=1}^k (x_i \in \mathcal{A}_i)$, and Ψ_2 be $x \in \Sigma^* \wedge \bigwedge_{i=1}^k (x_i \in \Sigma^*)$. Ψ_1 and Ψ_2 are PosPTL separable iff $\bigcap_{i=1}^k \mathcal{A}_i = \emptyset$. ◀

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Bui Phi Diep, Lukás Holík, Ahmed Rezine, and Philipp Rümmer. Flatten and conquer: a framework for efficient analysis of string constraints. In *PLDI*. ACM, 2017.

2 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Bui Phi Diep, Lukás Holík, Ahmed Rezine, and Philipp Rümmer. Trau: SMT solver for string constraints. In *FMCAD*. IEEE, 2018.

3 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Yu-Fang Chen, Lukás Holík, Ahmed Rezine, Philipp Rümmer, and Jari Stenman. String constraints for verification. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 150–166. Springer, 2014.

4 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Vrunda Dave, and Shankara Narayanan Krishna. On the separability problem of string constraints. *CoRR*, abs/2005.09489, 2020. `arXiv:2005.09489`.

5 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Bui Phi Diep, Lukás Holík, and Petr Janku. Chain-free string constraints. In Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza, editors, *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings*, volume 11781 of *Lecture Notes in Computer Science*, pages 277–293. Springer, 2019.

6 Parosh Aziz Abdulla and Richard Mayr. Priced timed petri nets. *Logical Methods in Computer Science*, 9(4), 2013. `doi:10.2168/LMCS-9(4:10)2013`.

7 Mohamed Faouzi Atig, Benedikt Bollig, and Peter Habermehl. Emptiness of Ordered Multi-Pushdown Automata is 2ETIME-Complete. *Int. J. Found. Comput. Sci.*, 28(8):945–976, 2017.

8 Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. Minimizing resources of sweeping and streaming string transducers. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 114:1–114:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

9 Murphy Berzish, Vijay Ganesh, and Yunhui Zheng. Z3str3: A string solver with theory-aware heuristics. In *2017 Formal Methods in Computer Aided Design, FMCAD 2017, Vienna, Austria, October 2-6, 2017*, pages 55–59. IEEE, 2017.

10 Murphy Berzish, Yunhui Zheng, and Vijay Ganesh. Z3str3: A string solver with theory-aware branching. *CoRR*, abs/1704.07935, 2017. `arXiv:1704.07935`.

11 Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996. `doi:10.1142/S0129054196000191`.

12 Taolue Chen, Yan Chen, Matthew Hague, Anthony W. Lin, and Zhilin Wu. What is decidable about string constraints with the replaceall function. *PACMPL*, 2(POPL):3:1–3:29, 2018. `doi:10.1145/3158091`.

13 Taolue Chen, Matthew Hague, Anthony W. Lin, Philipp Rümmer, and Zhilin Wu. Decision procedures for path feasibility of string-manipulating programs with complex operations. *Proc. ACM Program. Lang.*, 3(POPL), January 2019. `doi:10.1145/3290362`.

14 Taolue Chen, Matthew Hague, Anthony W. Lin, Philipp Rümmer, and Zhilin Wu. Decision procedures for path feasibility of string-manipulating programs with complex operations. *PACMPL*, 3(POPL):49:1–49:30, 2019.

15 Lorenzo Clemente, Wojciech Czerwinski, Slawomir Lasota, and Charles Paperman. Regular separability of parikh automata. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 117:1–117:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

16 Wojciech Czerwinski and Slawomir Lasota. Regular separability of one counter automata. *Logical Methods in Computer Science*, 15(2), 2019. URL: `https://lmcs.episciences.org/5563`, `doi:10.23638/LMCS-15(2:20)2019`.

17 Wojciech Czerwinski, Slawomir Lasota, Roland Meyer, Sebastian Muskalla, K. Narayan Kumar, and Prakash Saivasan. Regular separability of well-structured transition systems. In *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, volume 118 of *LIPIcs*, pages 35:1–35:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

18 Wojciech Czerwinski, Wim Martens, and Tomás Masopust. Efficient separability of regular languages by subsequences and suffixes. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 150–161. Springer, 2013.

19 Wojciech Czerwinski, Wim Martens, Lorijn van Rooijen, Marc Zeitoun, and Georg Zetzsche. A characterization for decidable separability by piecewise testable languages. *Discrete Mathematics & Theoretical Computer Science*, 19(4), 2017.

20 Vijay Ganesh and Murphy Berzish. Undecidability of a theory of strings, linear arithmetic over length, and string-number conversion. *CoRR*, abs/1605.09442, 2016. `arXiv:1605.09442`.

21 Vijay Ganesh, Mia Minnes, Armando Solar-Lezama, and Martin C. Rinard. Word equations with length constraints: What's decidable? In *Hardware and Software: Verification and Testing - 8th International Haifa Verification Conference, HVC 2012, Haifa, Israel, November 6-8, 2012. Revised Selected Papers*, volume 7857 of *Lecture Notes in Computer Science*, pages 209–226. Springer, 2012.

22 Vijay Ganesh, Mia Minnes, Armando Solar-Lezama, and Martin C. Rinard. (Un)Decidability results for word equations with length and regular expression constraints. *CoRR*, abs/1306.6054, 2013. `arXiv:1306.6054`.

23 Seymour Ginsburg and Sheila A. Greibach. Abstract families of languages. In *8th Annual Symposium on Switching and Automata Theory, Austin, Texas, USA, October 18-20, 1967*. IEEE Computer Society, 1967.

24 G. Higman. Ordering by divisibility in abstract algebras. *Proc. London Math. Soc. (3)*, 2(7), 1952.

25 Lukás Holík, Petr Janku, Anthony W. Lin, Philipp Rümmer, and Tomás Vojnar. String constraints with concatenation and transducers solved efficiently. *PACMPL*, 2(POPL):4:1–4:32, 2018. `doi:10.1145/3158092`.

26 Scott Kausler and Elena Sherman. Evaluation of string constraint solvers in the context of symbolic execution. In *ASE '14*. ACM, 2014.

27 Adam Kiezun, Vijay Ganesh, Philip J. Guo, Pieter Hooimeijer, and Michael D. Ernst. HAMPI: a solver for string constraints. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis, ISSTA 2009, Chicago, IL, USA, July 19-23, 2009*, pages 105–116. ACM, 2009.

28 Tianyi Liang, Andrew Reynolds, Cesare Tinelli, Clark Barrett, and Morgan Deters. A DPLL(T) theory solver for a theory of strings and regular expressions. In *CAV'14*, volume 8559 of *LNCS*. Springer, 2014.

29 Tianyi Liang, Andrew Reynolds, Nestan Tsiskaridze, Cesare Tinelli, Clark W. Barrett, and Morgan Deters. An efficient SMT solver for string constraints. *Formal Methods in System Design*, 48(3):206–234, 2016. `doi:10.1007/s10703-016-0247-6`.

30 Anthony Widjaja Lin and Pablo Barceló. String solving with word equations and transducers: towards a logic for analysing mutation XSS. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 123–136. ACM, 2016.

31 MakePHPSites. Php tutorials.`https://makephpsites.com/php-tutorials/user-management-tools/changing-passwords.php`, 2015.

32 Kenneth L. McMillan. Interpolation and SAT-based model checking. In *Computer Aided Verification, 15th International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings*, volume 2725 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2003.

33 Kenneth L. McMillan. An interpolating theorem prover. In *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, volume 2988 of *Lecture Notes in Computer Science*, pages 16–30. Springer, 2004.

34 Kenneth L. McMillan. Lazy abstraction with interpolants. In *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4144 of *Lecture Notes in Computer Science*, pages 123–136. Springer, 2006.

35 Christophe Morvan. On rational graphs. In *Foundations of Software Science and Computation Structures*, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

36 Thomas Place, Lorijn van Rooijen, and Marc Zeitoun. Separating regular languages by piecewise testable and unambiguous languages. In *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, volume 8087 of *Lecture Notes in Computer Science*, pages 729–740. Springer, 2013.

37 Wojciech Plandowski. An efficient algorithm for solving word equations. In *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, STOC '06, pages 467–476, New York, NY, USA, 2006. ACM. doi:10.1145/1132516.1132584.

38 Andrew Reynolds, Maverick Woo, Clark W. Barrett, David Brumley, Tianyi Liang, and Cesare Tinelli. Scaling up DPLL(T) string solvers using context-dependent simplification. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 453–474. Springer, 2017.

39 Prateek Saxena, Devdatta Akhawe, Steve Hanna, Feng Mao, Stephen McCamant, and Dawn Song. A symbolic execution framework for javascript. In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*, pages 513–528. IEEE Computer Society, 2010.

40 Prateek Saxena, Steve Hanna, Pongsin Poosankam, and Dawn Song. FLAX: systematic discovery of client-side validation vulnerabilities in rich web applications. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2010, San Diego, California, USA, 28th February - 3rd March 2010*. The Internet Society, 2010.

41 Imre Simon. Piecewise testable events. In *Automata Theory and Formal Languages, 2nd GI Conference, Kaiserslautern, May 20-23, 1975*, volume 33 of *Lecture Notes in Computer Science*, pages 214–222. Springer, 1975.

42 Minh-Thai Trinh, Duc-Hiep Chu, and Joxan Jaffar. S3: A symbolic string solver for vulnerability detection in web applications. In *CCS'14*. ACM, 2014.

43 Minh-Thai Trinh, Duc-Hiep Chu, and Joxan Jaffar. Progressive reasoning over recursively-defined strings. In *CAV'16*, volume 9779 of *LNCS*. Springer, 2016.

44 Moshe Y. Vardi. A note on the reduction of two-way automata to one-way automata. *Inf. Process. Lett.*, 30(5):261–264, 1989. doi:10.1016/0020-0190(89)90205-6.

45 Hung-En Wang, Tzung-Lin Tsai, Chun-Han Lin, Fang Yu, and Jie-Hong R. Jiang. String analysis via automata manipulation with logic circuit representation. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, volume 9779 of *Lecture Notes in Computer Science*, pages 241–260. Springer, 2016.

46 Fang Yu, Muath Alkhalaf, and Tevfik Bultan. Stranger: An automata-based string analysis tool for PHP. In *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 154–157. Springer, 2010.

47 Yunhui Zheng, Xiangyu Zhang, and Vijay Ganesh. Z3-str: A Z3-based string solver for web application analysis. In *ESEC/FSE'13*. ACM, 2013.

Weighted Transducers for Robustness Verification

Emmanuel Filiot
Université Libre de Bruxelles, Belgium
efiliot@ulb.ac.be

Nicolas Mazzocchi
Université Libre de Bruxelles, Belgium

Jean-François Raskin
Université Libre de Bruxelles, Belgium

Sriram Sankaranarayanan
University of Colorado Boulder, CO, USA

Ashutosh Trivedi
University of Colorado Boulder, CO, USA

Abstract

Automata theory provides us with fundamental notions such as languages, membership, emptiness and inclusion that in turn allow us to specify and verify properties of reactive systems in a useful manner. However, these notions all yield "yes"/"no" answers that sometimes fall short of being satisfactory answers when the models being analyzed are imperfect, and the observations made are prone to errors. To address this issue, a common engineering approach is not just to verify that a system satisfies a property, but whether it does so *robustly*. We present notions of robustness that place a metric on words, thus providing a natural notion of distance between words. Such a metric naturally leads to a topological neighborhood of words and languages, leading to quantitative and robust versions of the membership, emptiness and inclusion problems. More generally, we consider weighted transducers to model the cost of errors. Such a transducer models neighborhoods of words by providing the cost of rewriting a word into another. The main contribution of this work is to study robustness verification problems in the context of weighted transducers. We provide algorithms for solving the robust and quantitative versions of the membership and inclusion problems while providing useful motivating case studies including approximate pattern matching problems to detect clinically relevant events in a large type-1 diabetes dataset.

2012 ACM Subject Classification Computer systems organization → Reliability; Theory of computation → Formal languages and automata theory

Keywords and phrases Weighted transducers, Quantitative verification, Fault-tolerance

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.17

Funding This work is partially supported by the PDR project Subgame perfection in graph games (F.R.S.-FNRS), the MIS project F451019F (F.R.S.-FNRS), the ARC project Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond (Fédération Wallonie-Bruxelles), the EOS project Verifying Learning Artificial Intelligence Systems (F.R.S.-FNRS and FWO), the COST Action 16228 GAMENET (European Cooperation in Science and Technology), and the US National Science Foundation (NSF) under award numbers CPS 1836900 and CCF 1815983.
Emmanuel Filiot: Associate researcher at F.R.S.-FNRS
Nicolas Mazzocchi: PhD student funded by a FRIA fellowship from the F.R.S.-FNRS.

1 Introduction

Automata theoretic verification commonly uses an automaton S to specify the behaviors of a system being analyzed and another automaton P to specify the property of interest. These automata are assumed to be finite state machines accepting finite or infinite words. The key

step is to verify whether the language inclusion $L(S) \subseteq L(P)$ holds. Failing this inclusion, a counterexample σ is generated such that $\sigma \in L(S)$ whereas $\sigma \notin L(P)$. Another important area lies in runtime verification, wherein given a sequence of observations represented by $\sigma \in \Sigma^*$, we wish to check whether these observations satisfy the specification: $\sigma \in L(P)$. The verification community has considered numerous extensions to these basic ideas such as richer models of the system S that allow for succinct specifications (e.g., hierarchical state machines, state-charts), or go beyond finite state machines and include features such as real-time (timed automata) [4], physical quantities (hybrid automata) [3], and matching calls/returns [8, 23, 6]. The complexity of the language inclusion and membership problems in these settings are also well understood [11].

However, inclusion and membership problems lead to yes/no Boolean answers. The no answer for an inclusion problem is witnessed by a counterexample trace. However, the yes answer provides nothing further. A quantitative approach to these questions was proposed independently by Fainekos et al. [16], Donze et al. [14] and Rizk et al. [21] for the satisfaction of metric/signal temporal logic formula φ for a trace σ generated by continuous and hybrid systems. Therein, the authors use the euclidean metric over real-valued traces that defines a metric distance $d(\sigma, \sigma')$ between traces σ, σ' in order to check whether traces that are in the epsilon neighborhood of a given trace σ also satisfy the formula: $(\forall \sigma')\ d(\sigma, \sigma') < \epsilon \ \Rightarrow\ \sigma' \models \varphi$. Recent work, notably by Hasuo et al [24, 1] and Deshmukh et al [12] generalizes these notions to time domain as well as the signal data domain. Efficient algorithms for computing the robustness of a trace with respect to metric (signal) temporal formulas are known, and furthermore, the theory led to numerous approaches to finding falsifications of complex Simulink/Stateflow models, mining robust requirements and other monitoring problems [7].

Robustness Using Weighted Transducers. In this paper, we specify distances between finite words over Σ^*, using the notion of *cost functions*. A cost function assigns a non-negative rational cost to each pair of words $(w_1, w_2) \in \Sigma^* \times \Sigma^*$, modelling the cost of rewriting w_1 into w_2. By bounding the costs of rewritings, it models how words can be transformed. As a result, a neighborhood can be defined for each word, assuming that the cost of "rewriting" a word w back to itself is 0. This, in turn, allows to reason about robustness of languages. In order to model cost functions, we use *weighted transducers* with non-negative weights [15] along with an *aggregator* that combines the cost of each individual rewriting of the transducer into an overall cost between the input and output words. We now provide motivating examples for the cost functions that can be specified by such a model. A formal definition is provided in Section 2.

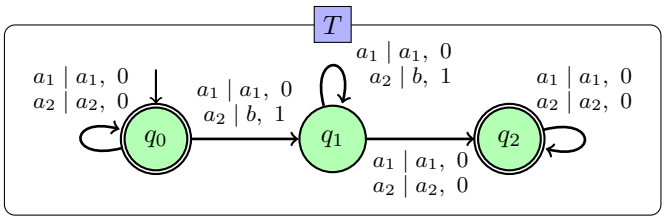


Figure 1 A weighted-transducer over $\Sigma = \{a_1, a_2, b\}$.

Motivating Example. Consider the transducer T of Figure 1. This transducer is over alphabet $\Sigma:\ \{a_1, a_2, b\}$. It allows to rewrite the letter a_1 into a_1 (at cost 0), and the letter a_2 into either a_2 (at cost 0) or b (at cost 1). Additionally, these rewritings are possible only

at state q_1. This allows us to have a model wherein errors appear in bursts rather than individually: I.e, an error at a location increases the likelihood of one at the subsequent location. Thus, the transducer models all possible words w' that a given input word w can be rewritten into. As an example, the word $w : a_1a_2a_2a_2$ into $w' : a_1bba_2$ through transitions that rewrite the first two occurrences of a_2 into b. At the same time, the transducer forbids certain rewritings. For instance, the word w above cannot be rewritten into the word $w'' : bba_2a_1$ since the rewrite from an a_1 into a b or an a_2 into an a_1 is clearly disallowed by the transducer T in Figure 1.

While the transducer T specifies the cost for individual rewritings through its transitions, we define the cost of rewriting the entire word w into another w' by additionally specifying an *aggregator* function. For simplicity, we assume that there is exactly one run of the transducer that rewrites w into w'. The case of nondeterministic transducers is defined in Section 2.

1. *Discounted Sum (`DSum`):* Given a discount factor $\lambda \in \mathbb{Q} \cap (0, 1)$, the cost of rewriting a word w into another word w' is defined as $\sum_{i=1}^{n} \lambda^{(i-1)}\tau_i$, wherein n is the size of a run through the transducer and τ_i is the cost associated with the i^{th} transition.
2. *Average (`Mean`):* This aggregator computes the mean cost: $\frac{1}{n}\sum_{i=1}^{n} \tau_i$ for $n > 0$.
3. *Sum (`Sum`):* This aggregator computes the sum: $\sum_{i=1}^{n} \tau_i$ for $n > 0$.

Returning to our example, the `Sum`-cost of rewriting $a_1a_2a_2a_2$ into a_1bba_2 is 2, for the `DSum`-cost with discount factor $1/2$, it is $3/4$, and for `Mean`-cost it is $1/2$.

Our approach handles a more general nondeterministic transducer model that can allow for insertions of new letters, deletion of letters, transpositions and arbitrary substitutions of one letter by a finite word. Cost functions defined by such transducers may not satisfy the axioms of a metric, however many commonly encountered type of metrics between words such as the Cantor distance and the Levenstein (or edit) distance can be modeled as weighted transducers [13]. For example, edit distance is naturally modelled by a sum-transducer. Cantor distance maps any pair of word (w_1, w_2) of same length to 2^{-i} where i the first position where w_1 and w_2 differ, and to 0 if $w_1 = w_2$. This metric can be modelled by a discounted-sum transducer with discount factor $1/2$.

Robustness problems. Given a cost function $c : (\Sigma^* \times \Sigma^*) \to \mathbb{Q}_{\geq 0}$ defined by a weighted-transducer with an aggregator function, we can define "neighborhoods" of languages for a given distance $\nu \geq 0$. For a regular language $N \subseteq \Sigma^*$ and a threshold $\nu \in \mathbb{Q}_{\geq 0}$, let us define its ν-neighborhood $N_\nu : \{w' \in \Sigma^* \mid (\exists w \in N)\ c(w, w') \leq \nu\}$. Given a property $L \subseteq \Sigma^*$, we consider the following robustness problems:

- **Robust inclusion**: Given N, ν and L, check whether $N_\nu \subseteq L$.
- **Threshold synthesis**: Given N, L, find the largest threshold ν such that $N_\nu \subseteq L$.
- **Robust kernel synthesis**: given N, ν, L, find the largest $M \subseteq N$ s.t. $M_\nu \subseteq L$.

▶ **Example 1.** Consider the transducer of Figure 1 using the the `Sum` aggregator. We take L as the set of words which does not have bbb as a subword. Now, any word of the form $(a_1a_2)^*$ are ν-robust for any threshold ν since the letter a_1 is not rewritten by the transducer T. Such questions are tackled using the robust inclusion problem. On the other hand, let us choose a word $w \in a_2a_2a_2(a_1^*)$. It is ν-robust for all the thresholds $\nu \leq 2$ but not for $\nu \geq 3$. This is determined using the threshold synthesis problem. For all $\nu \geq 3$, the set of ν-robust words in $N = \Sigma^*$ is $(a_1 + a_1a_2 + a_1a_2a_2)^*$, and for $\nu \leq 2$, any word in Σ^* is ν-robust. Such questions are solved using the robust kernel synthesis problem.

Contributions. We show that the robust inclusion problem is solvable in PTIME when N and L are regular languages (given as NFA and DFA respectively) and the weighted-transducer defining the cost function is also given as input (Corollary 12). To obtain this result, we show that we can effectively compute in PTIME the largest threshold ν as defined before, thus solving the threshold synthesis problem (Theorem 11). This result holds for the three measures `Sum`, `DSum` and `Mean`. For `Sum`, we show that the robust kernel is effectively regular (Lemma 14) and testing its non-emptiness is PSPACE-complete (Theorem 15). For `Mean`, we show that the robust kernel is not regular in general (Lemma 16), and its non-emptiness is undecidable (Theorem 17). For `DSum`, we leave those questions partially open. We conjecture that the robust kernel is non-regular in general and provide a sufficient condition under which it is regular (Theorem 22).

Next, we present an implementation of the algorithms to synthesize robustness thresholds and report some experiments with our implementation, illustrating its application to analyzing manual control strategies under the presence of human error and approximate pattern analysis in type-1 diabetes data. Here we analyze a publicly available dataset of blood glucose values for people with type-1 diabetes. In both cases, we use a weighted transducer to model some of the specifics of human error and glucose sensor noise patterns. For the type-1 diabetes application, we use a robust pattern matching to detect behaviors that are clinically significant while accounting for the peculiarities of the glucose sensor.

Our work bears some similarities with earlier work by Henzinger et al [17, 22]. In these papers, notions of robustness for string to string transformations are studied and the notion of continuity of these transformations is defined. This is different from our setting, in which we use weighted transducers to define notions of distances, and these transducers are not necessarily continuous. Our notion of robustness is with respect to the rewriting of the words of one language and not about the transducers. The transducers themselves serve to define neighborhoods of strings.

2 Preliminaries and Problem Statements

Let Σ be an alphabet. We denote the empty word by the symbol $\varepsilon \notin \Sigma$ and we write Σ^* for the set of finite words over Σ. Let $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$. As usual, we write $\mathbb{Q}$ for the set of rationals, $\mathbb{N} = \{0, 1, \dots\}$ for naturals, and $\mathbb{N}^*$ for the words over the infinite alphabet $\mathbb{N}$.

A finite automaton over Σ is a tuple $A = (Q, Q_I, Q_F, \Delta)$ where Q is the finite set of states, $Q_I \subseteq Q$ is the set of initial states, $Q_F \subseteq Q$ is the set of final states and $\Delta \subseteq Q \times \Sigma \times Q$ is the set of transitions. A *run* r of A over a word $u = a_1 \dots a_n \in \Sigma^*$ of length $n > 0$ is a sequence of transitions $t_1 \dots t_n \in \Delta^*$ such that there exist $q_0, q_1, \dots, q_n$ and for all $1 \leq i \leq n$, $t_i = (q_{i-1}, a_i, q_i)$. The run r is *simple* if no state repeats along r, i.e. $i \neq j$ implies that $q_i \neq q_j$ and, it is a cycle if $q_0 = q_n$. We say that r is a *simple cycle* if its a cycle and $t_2 \dots t_n$ is simple. Also, r is *accepting* if it starts from an initial state $q_0 \in Q_I$ and ends into a final state $q_n \in Q_F$. We denote by $\mathrm{AccRun}_A(u)$ the set of accepting runs of A on the word u. The language defined by A is the set of words $L(A) = \{u \mid \mathrm{AccRun}_A(u) \neq \varnothing\}$. The automaton A is called *deterministic* (DFA for short) if Q_I is a singleton and Δ is a function from $Q \times \Sigma$ to Q. We define the representation size of an automaton $A = (Q, Q_I, Q_F, \Delta)$ as $|A| = |Q| + |\Delta|$.

Weighted transducers extend finite automata with string outputs and weights on transitions [15]. Any accepting run over some input word rewrites each input symbol into a (possibly empty) word, with some cost in $\mathbb{N}$. Transducers can also have ϵ-input transitions with non-empty outputs, such that output symbols can be produced even though nothing is read on the input (e.g. allowing for symbol insertions). The output of a run is the

concatenation of all output words occurring on its transitions. Its cost is defined by an *aggregator function* $\mathsf{C} : \mathbb{N}^* \to \mathbb{Q}_{\geq 0}$, which associates a rational number to a sequence of non-negative integers.

We consider three different aggregator functions, given later. Since there are possibly several accepting runs over the same input, and generating the same output, we take the minimal cost of them to compute the value of a pair of input and output words.

▶ **Definition 2** (C-transducers). *Let $\mathsf{C}: \mathbb{N}^* \to \mathbb{Q}_{\geq 0}$ be an aggregator function. A C-transducer T is a tuple $(A, \mathtt{W})$ where $A = (Q, Q_I, Q_F, \Delta)$ is an NFA over $(\Sigma_\varepsilon \times \Sigma^*) \setminus \{(\varepsilon, \varepsilon)\}$ and the function $\mathtt{W}: \Delta \to \mathbb{N}$ associates weights to each transition.*

Given a transition $t = (q, a, v, q') \in Q \times \Sigma_\varepsilon \times \Sigma^* \times Q$, we write $\mathtt{Orig}(t) = q$, $\mathtt{In}(t) = a$, $\mathtt{Out}(t) = v$, and $\mathtt{Dest}(t) = q'$. We say that a transition $t \in \Delta$ can be triggered by T if it is in state $\mathtt{Orig}(t)$ and reads $\mathtt{In}(t)$ on its input (note that it is always possible to read $\mathtt{In}(t) = \varepsilon$). It, then, moves to $\mathtt{Dest}(t)$ and rewrites its input into $\mathtt{Out}(t)$. A run $r = t_1 \dots t_n$ of T is a run of A. We write $\mathtt{In}(r) = \mathtt{In}(t_1) \dots \mathtt{In}(t_n)$ and $\mathtt{Out}(r) = \mathtt{Out}(t_1) \dots \mathtt{Out}(t_n)$ and say that r is a run of T on the pair of words $(\mathtt{In}(r), \mathtt{Out}(r))$. Let $(u_1, u_2) = (\mathtt{In}(r), \mathtt{Out}(r))$. If moreover r is accepting, we say that (u_1, u_2) is accepted by T, and denote by $\mathrm{AccRun}_T(u_1, u_2)$ the set of accepting runs over (u_1, u_2). We also say that u_1 is accepted by T if (u_1, u_2) is accepted by T for some $u_2 \in \Sigma^*$. We denote the *weight sequence* of r by $\mathtt{W}(r) = \mathtt{W}(t_1) \dots \mathtt{W}(t_n)$ and its corresponding (aggregated) *cost* is $\mathsf{C}(r) = \mathsf{C}(\mathtt{W}(r))$.

A transducer T defines a relation from Σ^* to itself, called a *translation*, denoted R_T and defined by: $R_T = \{(u_1, u_2) \mid \mathrm{AccRun}_T(u_1, u_2) \neq \varnothing\}$. The *domain* of T, denoted $\mathrm{dom}(T)$ is the set of words u_1 for which there exists u_2 such that $(u_1, u_2) \in R_T$. The cost of a pair of words (u_1, u_2) is given by:

$$\mathsf{C}_T(u_1, u_2) = \begin{cases} +\infty & \text{if } (u_1, u_2) \notin R_T \\ \min\{\mathsf{C}(r) \mid r \in \mathrm{AccRun}_T(u_1, u_2)\} & \text{otherwise.} \end{cases}$$

Note that since runs consume at least one symbol of the input or one of the output, there are finitely many runs on a pair (u_1, u_2), hence the min is well-defined. Finally, given $\nu \in \mathbb{Q}$ and an input word $u_1 \in \mathrm{dom}(T)$, we define the threshold output language $T_{\leq \nu}(u_1)$ of u_1 as: $T_{\leq \nu}(u_1) = \{u_2 \mid \mathsf{C}_T(u_1, u_2) \leq \nu\}$. This notation extends naturally to languages $N \subseteq \Sigma^*$ by setting: $T_{\leq \nu}(N) = \bigcup_{u_1 \in N \cap \mathrm{dom}(T)} T_{\leq \nu}(u_1)$.

▶ **Assumption 3.** *We restrict our attention to C-transducers T that satisfy the condition that for all $u \in \mathrm{dom}(T)$, $\mathsf{C}_T(u, u) = 0$ (in particular $(u, u) \in R_T$). In other words, it is always possible to rewrite u into itself at zero cost.*

This assumption requires that each point must belong to any of its neighborhoods, which naturally comes from the indiscernibility axiom of distance. However, we do not require the triangle inequality axiom, that the edit distance does not satisfy.

Cost functions. We consider three aggregator functions, namely the sum, the mean and the discounted-sum. Let $\lambda \in \mathbb{Q} \cap (0, 1)$ be a discount factor. Given a sequence of weights $\overline{\tau} = \tau_1 \dots \tau_n$, those three functions are defined by:

$$\mathtt{Sum}(\overline{\tau}) = \sum_{i=1}^{n} \tau_i \qquad \mathtt{Mean}(\overline{\tau}) = \begin{cases} 0 & \text{if } \overline{\tau} = \varepsilon \\ \frac{\mathtt{Sum}(\overline{\tau})}{n} & \text{otherwise} \end{cases} \qquad \mathtt{DSum}(\overline{\tau}) = \sum_{i=1}^{n} \lambda^{(i-1)} \tau_i$$

Weighted-automata. When a C-transducer outputs only empty words, then its output component can be removed and we get what is called a *C-automaton*, which defines a function from words to costs. For $\mathsf{C} = \texttt{Sum}$, this definition of Sum-automaton coincides with the classical notion weighted automata over the semiring $(\mathbb{N} \cup \{+\infty\}, \min, +)$ from [15].

Robustness problems. We study the following three fundamental problems related to robustness for three different aggregator functions $\mathsf{C} \in \{\texttt{Sum}, \texttt{Mean}, \texttt{DSum}\}$. Given a threshold $\nu \in \mathbb{Q}$, a C-transducer T and a regular language L, a word $u \in \mathrm{dom}(T)$ is said to be *ν-robust* (or just robust if ν is clear from the context) if $T_{\leq\nu}(u) \subseteq L$. In other words, all its rewritings of cost ν at most are in L. A language $N \subseteq \Sigma^*$ is said to be ν-robust if $N \cap \mathrm{dom}(T)$ contains only ν-robust words. Finally, the *ν-robust kernel* of T is the set $\texttt{Rob}_T(\nu, L)$ of ν-robust words: $\texttt{Rob}_T(\nu, L) = \{u \in \mathrm{dom}(T) \mid T_{\leq\nu}(u) \subseteq L\}$. We prove that as the error threshold grows, so does the robust kernel.

▶ **Proposition 4.** *Given $\nu, \nu' \in \mathbb{Q}_{>0}$, a C-transducer T and a regular language L, we have that $\nu' \leq \nu \implies \mathit{Rob}_T(\nu', L) \subseteq \mathit{Rob}_T(\nu, L)$.*

Proof. By definition $T_{\leq\nu}(u_1) = \{u_2 \mid \mathsf{C}_T(u_1, u_2) \leq \nu\}$. For all $u_1 \in \mathrm{dom}(T)$ we have that $u_1 \in \texttt{Rob}_T(\nu, L)$ iff for all u_2 both $u_2 \in L$ and $\mathsf{C}_T(u_1, u_2) \leq \nu$ hold. Clearly $u_1 \in \texttt{Rob}_T(\nu, L)$ implies $u_1 \in \texttt{Rob}_T(\nu', L)$ for any $\nu' \leq \nu$. ◀

We are in a position to formally define the three key problems studied in this paper. For these definitions, we let $\mathsf{C} \in \{\texttt{Sum}, \texttt{Mean}, \texttt{DSum}\}$.

▶ **Problem 5** (Robust Inclusion)**.** *Given a C-transducer T, a regular language $N \subseteq \Sigma^*$ as an NFA, a threshold $\nu \in \mathbb{Q}_{\geq 0}$ and a language $L \subseteq \Sigma^*$ as a DFA, the* robust inclusion problem *is to decide whether $N \subseteq \mathit{Rob}_T(\nu, L)$, i.e. whether $T_{\leq\nu}(N) \subseteq L$.*

Note that we consider our specification language L deterministically presented, for tractability.

▶ **Problem 6** (Threshold Synthesis)**.** *Given a C-transducer T, a regular language $N \subseteq \Sigma^*$ as an NFA, and a regular language $L \subseteq \Sigma^*$ as a DFA, the* threshold synthesis problem *is to output a partition of the set of thresholds $\mathbb{Q}_{\geq 0} = G \uplus B$ into sets G and B of* good *and bad thresholds, i.e.*

$$G = \{\nu \in \mathbb{Q}_{\geq 0} \mid N \subseteq \mathit{Rob}_T(\nu, L)\} \text{ and } B = \{\nu \in \mathbb{Q}_{\geq 0} \mid N \not\subseteq \mathit{Rob}_T(\nu, L)\}.$$

As direct consequence of Proposition 4, the sets G and B are intervals of values, that is for all $\nu_1, \nu_2 \in \mathbb{Q}_{\geq 0}$, if $\nu_1 < \nu_2$ and $\nu_2 \in G$, then $\nu_1 \in G$, and if $\nu_1 \in B$ then $\nu_2 \in B$.

▶ **Problem 7** (Robust Kernel Non-emptiness)**.** *Given a C-transducer T, a regular language $L \subseteq \Sigma^*$ as a DFA, a threshold $\nu \in \mathbb{Q}_{\geq 0}$, the* robust kernel non-emptiness *problem is to decide if* there exists $u \in \mathit{Rob}_T(\nu, L)$.

For the cases where we provide algorithms for solving the non-emptiness of the robust kernel, we also succeed in synthesizing the robust kernel as an automaton.

3 Robust Verification

Given an instance of the threshold synthesis problem, we show how to compute the interval of good thresholds G and the interval of bad thresholds B in PTime for all the three measures we consider. As a corollary, we show that the robust inclusion problem for Sum, Mean, DSum measures is in PTime.

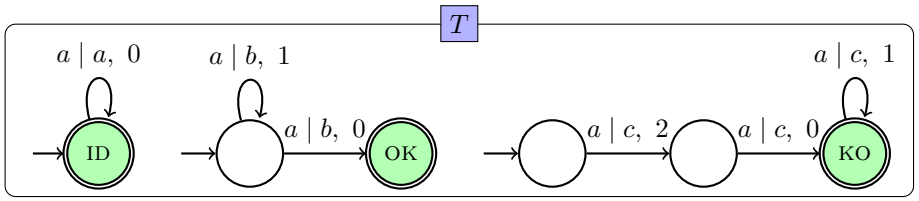


Figure 2 Transducer T for which the infimums $\nu_{T,L}^{\texttt{Mean}} = 1$ and $\nu_{T,L}^{\texttt{DSum}} = 2$ are bad thresholds for T interpreted as `Mean`- and `DSum`-transducer with discount factor $\frac{1}{2}$ respectively, and for $L = a^* + b^*$.

In the following, we assume that $N = \mathrm{dom}(T)$. This is w.l.o.g. as transducers are closed (in polynomial time) under regular domain restriction (using a product construction of T with the automaton for N). With this assumption, the set of good thresholds G becomes $G = \{\nu \in \mathbb{Q}_{\geq 0} \mid \mathrm{dom}(T) \subseteq \mathsf{Rob}_T(\nu, L)\}$ and dually for the set of bad thresholds B. We let $\nu_{T,L}$ be the infimum of the set of bad thresholds, i.e. $\nu_{T,L} = \inf B = \inf\{\nu \in \mathbb{Q}_{\geq 0} \mid \mathrm{dom}(T) \not\subseteq \mathsf{Rob}_T(\nu, L)\}$. As illustrated by the following example, computing $\nu_{T,L}$ allows us to compute $G = [0, \upsilon_{T,L}]$ and $B = [\upsilon_{T,L}, +\infty)$.

▶ **Example 8.** Let $\Sigma = \{a, b, c\}$ and $\mathsf{C} \in \{\texttt{Mean}, \texttt{DSum}\}$. Consider the best threshold problem for T the C-transducer of Figure 2, $N = \mathrm{dom}(T) = a^*$ and $L = a^* + b^*$. Note that the translations accepted by OK and ID belong to L. On the contrary, translations accepted by KO do not belong to L and so they are not robust w.r.t. L for any threshold. For `Mean` measure, the cost of a translation into c^* is exactly 1 while the one into b^* range over $[0, 1)$. Hence $\nu_{T,L}^{\texttt{Mean}} = 1$ and the set partition of good and bad thresholds is $G^{\texttt{Mean}} = [0, 1)$ and $B^{\texttt{Mean}} = [1, +\infty)$. In the case of `DSum` with discount factor 0.5, the cost of a translation into c^* range over $[2, 2.5)$ while the one into b^* range over $[0, 2)$. So $\nu_{T,L}^{\texttt{DSum}} = 2$ and the thresholds are partitioned by $G^{\texttt{DSum}} = [0, 2)$ and $B^{\texttt{DSum}} = [2, +\infty)$.

Then, we associate with every transducer T and property L given by some DFA A (assumed to be complete), a graph called the *weighted-graph associated with T and A*, and denoted by $G_{T,A}$. Intuitively, $G_{T,A}$ is obtained by first taking the synchronised product of T and A (where A is simulated on the outputs of T) and then by projecting this product on the inputs.

Formally, given $T = (Q, Q_I, Q_F, \Delta, \mathtt{W})$ and $A = (P, p_I, P_F, \delta)$, the synchronised product $G_{T,A} = (V, E, \mathtt{W}' \colon E \to \mathbb{N})$ is such that:

- $V = Q \times P$
- E is the set of edges $e = (q, p) \to (q', p')$ such that there exists $a \in \Sigma_\varepsilon$ and a transition $t = (q, a, u, q') \in \Delta$ such that $p' = \delta(p, u)$ where δ has been extended to words in the expected way. We say that e is compatible with t.
- For all $e \in E$, $\mathtt{W}'(e) = \min\{\mathtt{W}(t) \mid e \text{ is compatible with some } t \in \Delta\}$.

Additionally, we note $V_I = Q_I \times \{p_I\}$ the set of *initial vertices* and $V_F = Q_F \times (P \setminus P_F)$ the set of *final vertices* of this graph. Given a path π in this graph as a sequence of edges $e_1 \dots e_n$, we let $\mathsf{C}(\pi) = \mathsf{C}(\mathtt{W}'(e_1) \dots \mathtt{W}'(e_n))$.

The following lemma establishes some connection between $\nu_{T,L}$ and the paths of $G_{T,A}$.

▶ **Lemma 9.** *The infimum cost of paths from a vertex in V_I to a vertex in V_F is equal to $\nu_{T,L}$, i.e. $\nu_{T,L} = \inf\{\mathsf{C}(\pi) \mid \exists s_0 \in V_I \ \exists s_f \in V_F \ s_0 \xrightarrow{\pi}_{G_{T,A}} s_f\}$.*

Proof. We first show that any path π from V_I to V_F satisfies $\mathsf{C}(\pi) \geq \nu_{T,L}$. Take such a path. By construction of $G_{T,A}$, there exists an input word $u_1 \in \mathrm{dom}(T)$, some output word $u_2 \notin L$ and an accepting run r of T on (u_1, u_2) of value $\mathsf{C}(r) = \mathsf{C}(\pi)$. Since the value $\mathsf{C}_T(u_1, u_2)$ is

the minimal value of all accepting runs of T over (u_1, u_2), we have $\mathsf{C}(r) \geq \mathsf{C}_T(u_1, u_2)$ and u_1 is not robust for threshold $\mathsf{C}_T(u_1, u_2)$, *a fortiori* for threshold $\mathsf{C}(r)$, from which we get $\mathsf{C}(r) = \mathsf{C}(\pi) \geq \nu_{T,L}$. This shows that $\nu_{T,L} \leq \inf\{\mathsf{C}(\pi) \mid \exists s_0 \in V_I \exists s_f \in V_F \ s_0 \xrightarrow{\pi}_{G_{T,A}} s_f\}$.

Suppose that $\nu_{T,L}$ is strictly smaller than this infinimum (that we denote m) and take some rational number ν such that $\nu_{T,L} < \nu < m$. Since $\nu_{T,L} < \nu$, it is a bad threshold which means that there exists $u_1 \in \mathrm{dom}(T)$ such that $u_1 \notin \mathtt{Rob}_T(\nu, L)$. Hence there exists $u_2 \notin L$ such that $\mathsf{C}_T(u_1, u_2) \leq \nu$, and by definition of $G_{T,A}$, there exists a path π from V_I to V_F of value $\mathsf{C}(\pi) \leq \nu$. This contradicts the fact that $\nu < m$ by definition of m. Hence, $\nu_{T,L} = m$, concluding the proof. ◀

The next lemma establishes that the infimum of values of paths between two sets of states in a weighted graph can be computed in PTime and it is also decidable in PTime if the infimum is realized by a path, for all the three measures considered in this paper. As a direct corollary of this lemma we obtain the main theorem of the section. The full proof can be found in Appendix.

▶ **Lemma 10.** *For a weighted graph $G = (V, E, \mathtt{W} \colon E \to \mathbb{Q}_{\geq 0})$, a set of sources $V_I \subseteq V$ and a set of targets $V_F \subseteq V$, the infimum of the weights of paths from V_I to V_F can be computed in PTime for all $\mathsf{C} \in \{\mathtt{Sum}, \mathtt{DSum}, \mathtt{Mean}\}$. Moreover, we can decide in PTime if this infimum is realizable by a path.*

Sketch of proof. First, if no state of V_F are reachable from some state of V_I, we have $\nu_{T,L} = +\infty$. Otherwise we use different procedures, depending on the aggregator C.

For `Sum`, the infimum can be computed in Ptime using Dijkstra algorithm and it is always feasible. For `Mean`, we first note that the infimum is the `Mean` value of either a simple path or the value of a reachable cycle that can be iterated before moving to some target. In the latter case, the infimum is not feasible but can be approximated as close as possible by iterating the cycle. So, the infimum is feasible iff it is the `Mean` value of a simple path. The minimal `Mean` values amongst simple paths and cycles can be computed in Ptime with dynamic programming thanks to [18]. For `DSum`, Theorem 1 of [5] provides a PTime algorithm that computes for all $v \in V$, the infimum of `DSum` values x_v of paths reaching the target V_F from v. ◀

▶ **Theorem 11.** *For a given C-transducer T, a language $N \subseteq \Sigma^*$ given as an NFA and $L \subseteq \Sigma^*$ given as a DFA, the set partition of good and bad thresholds (G, B) for $\mathsf{C} \in \{\mathtt{Sum}, \mathtt{DSum}, \mathtt{Mean}\}$ can be computed in PTime.*

Proof. First, we restrict the domain of T to N by taking the product of T and the automaton for N (simulated over the input of T). Then, according to Lemma 10, we can compute in PTime the value $\nu_{T,L}$. This value is the infimum of B. If this infimum is feasible then the interval B is left closed and equal to $[\nu_{T,L}, +\infty)$ while $G = [0, \nu_{T,L})$, and on the contrary, if this infimum is not feasible, then B is left open and equal to $(\nu_{T,L}, +\infty)$, while $G = [0, \nu_{T,L}]$. Note that when $\nu_{T,L} = 0$ and is feasible, then $G = [0, 0) = \varnothing$. ◀

As a direct consequence, the robust inclusion problem for a threshold ν can be solved by checking if $\nu \in G$, and so we have the following corollary.

▶ **Corollary 12.** *Let $\mathsf{C} \in \{\mathtt{Sum}, \mathtt{DSum}, \mathtt{Mean}\}$. Given T a C-transducer, $N \subseteq \Sigma^*$ given as an NFA, $L \subseteq \Sigma^*$ given as a DFA and $\nu \in \mathbb{Q}$. The language inclusion $N \subseteq \mathtt{Rob}_T(\nu, L)$ can be decided in PTime.*

4 Robust Kernel Synthesis

In this section, we show that the robust kernel is regular for Sum-transducers, and checking its emptiness is PSPACE-complete. For Mean, we show that it is not necessarily regular, and checking its emptiness is undecidable. For DSum, we conjecture that the robust kernel is non-regular and give sufficient condition under which it is regular and computable, implying decidability of its emptiness.

4.1 Sum measure

To show robust kernel regularity, we rely on the construction of Theorem 2 of [2] in the context of weighted automata over the semiring $(\mathbb{N} \cup \{+\infty\}, \min, +)$. The following lemma, use the same automata construction and provides an upper bound on the number of states required to denote a threshold language with a DFA.

▶ **Lemma 13.** *Let U be an n state Sum-automaton and $\nu \in \mathbb{N}$. The threshold language $L_\nu(U) = \{w \mid U(w) \geq \nu\}$, where $U(w)$ is defined as $+\infty$ if there is no accepting run on w, otherwise as the minimal sum of the weights along accepting runs on w, is regular. Moreover $L_\nu(U)$ is recognized by a DFA with $O\big((\nu+2)^n\big)$ states.*

Proof. First, let assume that U has universal domain (i.e. any word has some accepting run), otherwise we complete it by assigning value ν to each word of its complement.

Then, $U(w) \geq \nu$ iff all the accepting runs on w have value at least ν. We design a DFA D that accepts exactly those words. Since the weights of U are non-negative, D just has to monitor the sum of all runs up to ν, by counting in its states. If Q is the set of states of U, the set of states of D is $2^{Q \times \{0,\dots,\nu-1,\nu^+\}}$, where ν^+ intuitively means any value $\geq \nu$. We extend natural addition to $X = \{0,\dots,\nu-1,\nu^+\}$ by letting $a + b = \nu^+$ iff $a = \nu^+$, or $b = \nu^+$, or $a + b \geq \nu$. Then, D is obtained by subset construction: there is a transition $P \xrightarrow{\sigma} P'$ in D iff $P' = \{(q', i+j) \mid (q,i) \in P \wedge q \xrightarrow{\sigma|j}_U q'\}$. A state P is accepting if $P \cap \big((Q \setminus F) \times \{0,\dots,\nu-1\}\big) = \varnothing$, where F are the accepting states of U.

Though simple, the latter construction does not give the claimed complexity, as the number of states of D is $2^{n\nu}$. But the following simple observation allows us to get a better state complexity. Consider an input word of the form uv. If after reading u, D reaches some state P such that for some state q, there exists $(q,i),(q,j) \in P$ such that $i < j$, then if there is an accepting run of U from q on v, with sum s, there is an accepting run on uv with sum $i+s$ and one with sum $j+s$. Therefore if $i+s \geq \nu$, then $j+s \geq \nu$ and the pair (q,j) is useless in P. So, we can keep only the minimal elements in the states of D, where minimality is defined with respect to the partial order $(q,i) \preceq (p,j)$ if $q = p$ and $i \leq j$. Let us call D_{opt} the resulting "optimised" DFA. Its states can be therefore seen as functions from Q to $\{0,\dots,\nu-1,\nu^+\}$, so that we get the claimed state-complexity. ◀

▶ **Lemma 14** (Robust language regularity)**.** *Let T be a Sum-transducer, $\nu \in \mathbb{N}$ and L be regular language. The language of* robust words *$\mathtt{Rob}_T(\nu, L)$ is a regular language. Moreover, if L is given by a DFA with n_L states and T has n_T states, then $\mathtt{Rob}_T(\nu, L)$ is recognisable by a DFA with $O\big((\nu+2)^{n_T \times n_L}\big)$ states.*

Proof of this lemma is provided in the appendix.

▶ **Theorem 15.** *Let T be a* Sum*-transducer, $\nu \in \mathbb{N}$ given in binary and L a regular language given as a* DFA*. Then, it is* PSPACE*-complete to decide whether there exists a robust word $w \in \mathtt{Rob}_T(\nu, L)$. The hardness holds even if ν is a fixed constant, T is letter-to-letter[1] and io-unambiguous[2], and its weights are fixed constants in $\{0, 1\}$.*

Proof. From Lemma 14, $\mathtt{Rob}_T(\nu, L)$ is recognisable by a DFA with $O\big((\nu+2)^{n_T \times n_L}\big)$ states, where n_T is the number of states of T and n_A the number of states of the DFA defining L. Checking emptiness of this automaton can be done in PSPACE (apply the standard NLOGSPACE emptiness checking algorithm on an exponential automaton that needs not be constructed explicitly, but whose transitions can be computed on-demand).

To show PSPACE-hardness, we reduce the problem from [19] of checking the non-emptiness of the intersection of n regular languages given by n DFA $A_1, \dots, A_n$, over some alphabet Γ. In particular, we construct T, ν and a DFA A such that $\bigcap_i L(A_i) \neq \varnothing$ iff there exists a robust word with respect to T,ν and L.

We define the alphabet as $\Sigma = \Gamma \cup \{\#_1, \dots, \#_n, \dashv\}$ where we assume that $\#_1, \dots, \#_n, \dashv \notin \Gamma$, and construct a transducer T which reads a word $w\dashv$ of length $k = |w| + 1$ with $w \in \Gamma^*$, and rewrites it into either itself, or $(\#_i)^k$ for all $i \in \{1, \dots, n\}$. The identity rewriting has total weight 0 while the rewriting into $\#_i^k$ has total weight 1 if $w \in L(A_i)$, and 0 otherwise. The transducer T is constructed as the disjoint union of $n+1$ transducers $T_1, \dots, T_n, T_\dashv$. For all $i \in \{1, \dots, n\}$, T_i simulates A_i on the input and outputs $\#_i$ whenever it reads an input letter different from $\dashv$, with weight 0. When reading $\dashv$ from an accepting state of A_i, it outputs $\dashv$ with weight 1, and if it reads $\dashv$ from a non-accepting state, it outputs $\dashv$ with weight 0. Finally, $T_\dashv$ just realizes the identify function with weight 0. Note that T has polynomial size in $A_1, \dots, A_n$ and it is letter-to-letter and (input,output)-deterministic.

Now we prove that a word $w \dashv$ is robust iff $w \in \bigcap_i L(A_i)$. Assume that there exists a robust word $w \dashv$ for the property $L = (\Gamma \cup \{\dashv\})^*$ and threshold $\nu = 0$. Equivalently, it means that for all rewritings $\alpha \in \Sigma^*$, if $\mathtt{Sum}_T(w\dashv, \alpha) \leq 0$ then $\alpha \in L$. It is equivalent to say that all its rewritings α satisfies either $\mathtt{Sum}_T(w\dashv, \alpha) \geq 1$ or $\alpha \in L$. By definition of T, it is equivalent to say that all rewritings α are such that either $\alpha \in (\#_i)^* \cdot \dashv$ for some i and $w \in L(A_i)$, or $\alpha = w \dashv$. Since T necessarily rewrites $w\dashv$ into $w\dashv$, as well as into $(\#_1)^k, \dots, (\#_n)^k$, where $k = |w|+1$, the latter assumption is equivalent to saying that $w \in L(A_i)$ for all $i \in \{1, \dots, n\}$, concluding the proof. ◀

4.2 Mean measure

Let us first establish non-regularity of the robust kernel.

▶ **Lemma 16.** *Given a regular language L, a* Mean*-transducer T and $\nu \in \mathbb{Q}_{\geq 0}$, the language $\mathtt{Rob}_T(\nu, L)$ is not necessarily regular, but recursive.*

Proof. Consider the language $L = \{w \mid \exists i \in \mathbb{N} : w(i) = a\}$ on the alphabet $\Sigma = \{a, b\}$, i.e. the set of words on Σ that contain at least one a. Now, consider a (one state) transducer T that can non-deterministically copy letters or change the current letter from a to b with weight one. Now, if we fix ν to be equal to $\frac{1}{2}$, then all the translations of w by T of cost less than $\frac{1}{2}$ are included in L, i.e. each translation of w will contain at least one letter a, if and only if, the number of a's in w is larger than the number of b's in w, i.e. $\mathtt{Rob}_T(\frac{1}{2}, L) = \{w \mid w_{\sharp a} > w_{\sharp b}\}$, which is not regular. Note that in general $\mathtt{Rob}_T(\nu, L)$ is recursive because the membership problem to it, is decidable by Corollary 12 (applied on a singleton language). ◀

[1] A transducer is letter-to-letter if $\Delta \subseteq Q \times \Sigma \times \Sigma \times Q$.
[2] For all word pairs (w_1, w_2), there exists at most one run of T on w_1 outputting w_2.

We now show that testing the non-emptiness of the robust kernel is undecidable.

▶ **Theorem 17.** *Let L be a regular language, T be a* Mean*-transducer and $\nu \in \mathbb{Q}_{\geq 0}$. Determine whether $\mathtt{Rob}_T(\nu, L) \neq \varnothing$ is undecidable. It holds even if T is io-unambiguous.*

Proof. Let A be a Sum-automaton weight by integers. The proof goes by reduction from determining whether all words admits a run of non-positive cost in A which is known to be undecidable [10, 2]. From A, we construct L as the set of *non* accepting runs of A union Σ^*, the threshold ν as the maximal absolute weight of A and T such that:

$$\mathtt{Mean}_T = \bigcup \begin{array}{l} \{(w, w) \mapsto 0 \mid w \in \Sigma^*\} \\ \{(w, r_w) \mapsto X_{r_w} + \nu|w| \mid r_w \text{ run of } A \text{ over } w \in \Sigma^* \text{ with value } X_{r_w}\} \end{array}$$

We can construct T as the disjoint union between a single-state transducer with weights zero realising the identity, and a transducer that outputs all the possible runs of A on its input, such that each T-transition simulating an A-transition t of value x (in A) has value $\nu + x$, which is positive by definition of ν. Hence T is indeed weighted over non-negative numbers. Note that T is io-unambiguous: if the input and output are fixed, there is at most one run of T. Now, we show that $\mathtt{Rob}_T(\nu, L) = \varnothing$ iff $\forall w \cdot A(w) \leq 0$, i.e.

$$\forall w_1 \exists w_2 \in \overline{L}\ \mathtt{Mean}_T(w_1, w_2) \leq \nu \text{ iff } \forall w\ A(w) \leq 0.$$

We have the following equivalences: $\forall w_1 \exists w_2 \in \overline{L} \cdot \mathtt{Mean}_T(w_1, w_2) \leq \nu$ iff for all w_1, there exists an accepting run r of A on w_1 such that $\mathtt{Mean}_T(w_1, r) \leq \nu$, i.e. $\mathtt{Sum}_T(w_1, r) \leq \nu|w_1|$ and by definition of T, it is equivalent to asking that $\mathtt{Sum}_A(r) + \nu|w_1| \leq \nu|w_1|$, i.e. $\mathtt{Sum}_A(r) \leq 0$. Hence, the latter statement is equivalent to the fact that for all words w_1, there exists an accepting run of A of value ≤ 0. Since A takes the minimal value of all accepting runs to compute the value of a word, it is equivalent to saying that for all w_1, $A(w_1) \leq 0$, i.e., A is universal, concluding the proof. ◀

4.3 Discounted sum measure

For DSum-transducer, we conjecture that $\mathtt{Rob}_T(\nu, L)$ is in general non-regular. This claim is substantiated by the fact that DSum-automata over $\mathbb{Q}$ and ω-words have in general non-regular cut-point languages, i.e. the set of words of DSum value below a given threshold is in general non-regular [9]. With a proof similar to that of Theorem 17 for Mean-transducers, it is possible to show that the universality problem for DSum-automata, which is open to the best of our knowledge, reduces to checking the emptiness of the robust language of a DSum-transducer.

Following an approach that originates from the theory of probabilistic automata, it is has been shown that cut-point languages are regular when the threshold is ϵ-isolated [9]. Formally, a threshold $\nu \in \mathbb{Q}$ is ϵ-isolated, for $\epsilon > 0$ and for some DSum-transducer T if, for all accepting runs r of T, $\mathtt{DSum}_T(r) \in [0, \nu - \epsilon] \cup [\nu + \epsilon, +\infty)$. It is *isolated* if it is ϵ-isolated for some ϵ. Our objective now is to show that when ν is isolated, then $\mathtt{Rob}_T(\nu, L)$ is regular and one can effectively construct an automaton recognizing it. We will also give a (possibly non-terminating) algorithm which, when it terminates, returns an automaton recognising $\mathtt{Rob}_T(\nu, L)$, and which is guaranteed to terminate whenever ν is ϵ-isolated for some ϵ. Towards these results, we first give intermediate useful results. For a state q of T, we call *continuation* of q any run from q leading to some accepting state of T. By extension, we also call continuation of a run r any continuation of the last state of r. A transducer T is said to be *trim* if all its states admits some continuation. Note that any transducer can be transformed into an equivalent trim one in PTime, just by removing states that do not admit any continuation (this can be tested in PTime).

▶ **Lemma 18.** *Let T be a trim DSum-transducer and $\nu \in \mathbb{Q}$. If ν is ϵ-isolated for some ϵ, then there exists $n^* \in \mathbb{N}$ such that any run r of length at least n^* satisfies one of the following properties:*

1. *$\mathtt{DSum}(r) \leq \nu - \epsilon$ and any continuation r' of r satisfies $\mathtt{DSum}(rr') \leq \nu - \epsilon$*
2. *$\mathtt{DSum}(r) \geq \nu + \epsilon/2$ and any continuation r' of r satisfies $\mathtt{DSum}(rr') \geq \nu + \epsilon$.*

Proof of this lemma is provided in the appendix.

We now show how to construct better and better regular under-approximations of the set of *non*-robust words, show that they "finitely" converge to the set of non-robust words when ν is isolated.

▶ **Lemma 19.** *Let T be a DSum-transducer, $\nu \in \mathbb{Q}$ and L a regular language (given as a DFA). For all n, we can construct an NFA A_n such that:*

1. *$L(A_n) \subseteq L(A_{n+1})$*
2. *$L(A_n) \subseteq \overline{\mathtt{Rob}_T(\nu, L)} \cap \mathrm{dom}(T)$*

Moreover, if ν is isolated, there exists n^* such that $L(A_{n^*}) = \overline{\mathtt{Rob}_T(\nu, L)} \cap \mathrm{dom}(T)$.

Proof of this lemma is provided in the appendix.

We also show that one can test whether given n, we have $\overline{\mathtt{Rob}_T(\nu, L)} \cap \mathrm{dom}(T) \subseteq L(A_n)$, as stated by the following lemma:

▶ **Lemma 20.** *Given a regular language N (given as some NFA), it is decidable to check whether $\overline{\mathtt{Rob}_T(\nu, L)} \cap \mathrm{dom}(T) \subseteq N$ holds.*

Proof. We take the synchronised product of T, $\overline{L}$ (on the output) and $\overline{N}$ (on the input), project the output, and check whether a path from an initial to a final vertex exists with discounted sum $\leq \nu$. ◀

Those results allow us to define the following semi-algorithm:

1. **Compute-Rob**(T, ν, L)
2. **for** n **from** 1 **to** $+\infty$
3. **compute** A_n // as in Lemma 19
4. **if** $\overline{\mathtt{Rob}_T(\nu, L)} \cap \mathrm{dom}(T) \subseteq L(A_n)$ **return** A_n // using Lemma 20

▶ **Lemma 21.** *The algorithm **Compute-Rob**(T, ν, L) satisfies the following properties:*

1. *if it terminates, then it returns an automaton recognising $\overline{\mathtt{Rob}_T(\nu, L)} \cap \mathrm{dom}(T)$,*
2. *if ν is isolated, it terminates.*

Proof. If it terminates at steps n, then by Lemma 19 and the test at line 4 we know that $L(A_n) = \overline{\mathtt{Rob}_T(\nu, L)} \cap \mathrm{dom}(T)$, and if ν is isolated, the test will eventually succeed. ◀

Note that the algorithm may terminate even if ν is not isolated. It is the case for instance when the threshold is ϵ-isolated for "long" runs only, but not necessarily for small runs, in the sense that it is only required that for some n, any accepting runs of length at least n satisfies either $\mathtt{DSum}(r) \leq \nu - \epsilon$ or $\mathtt{DSum}(r) \geq \nu + \epsilon$. As a corollary of Lemma 21, $\mathtt{Rob}_T(\nu, L)$ is regular when ν is isolated: it suffices to run Algorithm **Compute-Rob**, complement the automaton and restrict its language to $\mathrm{dom}(T)$.

▶ **Theorem 22.** *Let T be a DSum-transducer and $\nu \in \mathbb{Q}$ and L a regular language. If ν is isolated, then $\mathtt{Rob}_T(\nu, L)$ is regular.*

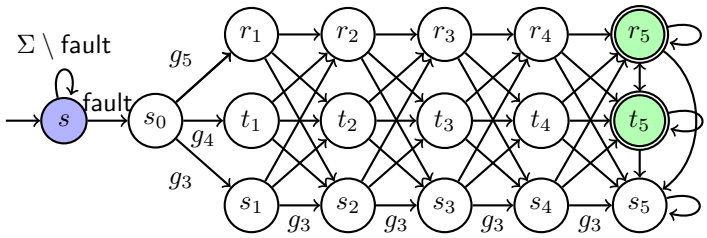


Figure 3 Finite state automaton P showing a desired property for the automatic transmission system. All incoming edges to $s_1, \ldots, s_5$ have label g_3, incoming edges to $t_1, \ldots, t_5$ have label g_4 and $r_1, \ldots, r_5$ have incoming edges labelled g_5. All edges not shown lead to a rejecting sink state.

5 Implementation and Case Study

We describe an evaluation of the ideas presented thus far and their application to two case studies: one involving robustness of control strategies to human mistakes and the other involving glucose values for patients with type-1 diabetes. We have implemented in Python the threshold synthesis problem (Problem 6) for the discounted and average costs. Our implementation supports the specification of a language L specified as an NFA, a weighted transducer T and a property P specified as some DFA. The implementation is available upon request.

5.1 Robustness of Human Control Strategies

An industrial motor operates under many gears $g_1, \ldots, g_5$. Under fault, the human operator must take control of the machine and achieve the following: *If the system goes into a fault the operator must ensure that (a) the system is immediately set in gears* $3-5$*. Subsequently, for the next* 5 *cycles: (b) it must never go to gear* g_1 *or* g_2*; and (c) must shift and stay at a higher gear* g_4 *or* g_5 *after the* 5^{th} *cycle until the fault is resolved.*

Figure 3 shows a finite state machine P that accepts all words satisfying this property: fault is not in the operator's control but $g_1, \ldots, g_5$ are operator actions. Consider that the operator can perform this task in two different ways: σ_1 : fault g_4 g_4 g_4 g_5 g_5 versus σ_2 : fault g_3 g_3 g_3 g_3 g_4. The input σ_1 induces the run $s, s_0, t_1, t_2, t_3, r_4, r_5$ whereas the input σ_2 induces the run $s, s_0, s_1, s_2, s_3, s_4, t_5$. Both σ_1, σ_2 satisfy the property of interest and as such there is nothing to choose one over the other. Suppose the human operator can make mistakes, especially since they are under stress. We will consider that the operator can substitute a command for gear g_i with g_{i-1} (for $i > 1$) or g_{i+1} (for $i < 5$). We use a weighted transducer T_0 shown in Figure 4 to model these substitutions. The transducer defines possible ways in which a string σ can be converted to σ' with a notion of cost for the conversion. In this example we consider two notions of cost: the `DSum`-cost, and the `Mean`-cost. These costs now allow us to compare σ_1 versus σ_2. For instance, under both notions we will discover that σ_1 is much more robust than σ_2. The robustness of σ_1 under both cost models is ∞ since any change to σ_1 under the transducer continues to satisfy the desired property. On the other hand σ_2 has a finite robustness, since operator mistakes can cause violations.

The use of a transducer allows for a richer specification of errors. For instance, transducer T_2 in Fig. 4 shows a model of "bounded" number of mistakes that assume that the operator makes at most 2 mistakes whereas T_3 in Fig. 4 shows a model with "bursty" mistakes that

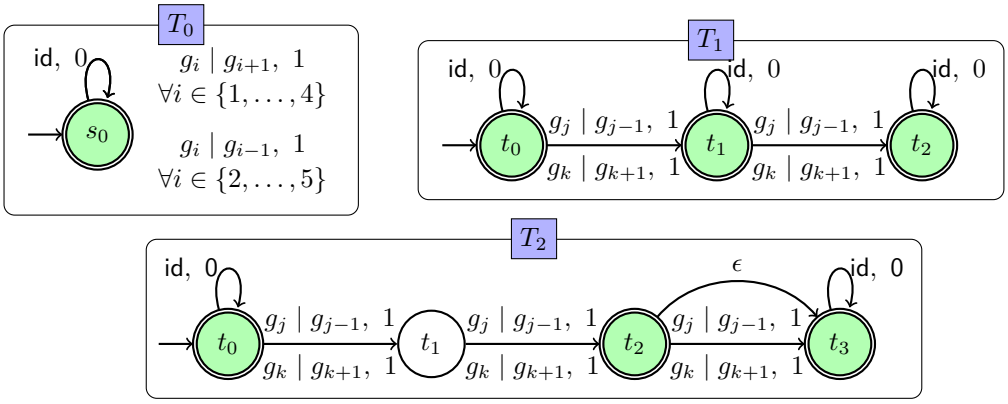


Figure 4 Transducers modeling potential human operator mistakes along with their costs: T_0 allows arbitrarily many mistakes whereas T_1 restricts the number of mistakes to at most 2, whereas T_2 models a "bursty" set of mistakes. The edge $a \mid b, w$ denotes a replacement of the letter a by b with a cost w. For convenience T_2 uses an ϵ transition that can be removed.

Table 1 Running times and robustness values computed for various input strings (the first letter **fault** is common to all the strings and is omitted). All timings are measured in seconds, ϵ denotes time < 0.01 seconds.

String	T_0			T_1			T_2		
	Disc.	Avg.	Time	Disc.	Avg.	Time	Disc.	Avg.	Time
$g_4g_4g_4g_4g_5g_5$	∞	∞	ϵ	∞	∞	ϵ	∞	∞	ϵ
$g_3g_3g_3g_4g_4g_4$	2^{-5}	$\frac{1}{6}$	0.03	2^{-5}	$\frac{1}{6}$	0.03	$\frac{7}{32}$	$\frac{1}{2}$	0.03
$g_3g_4g_4g_4g_5g_4g_4g_4g_3g_4$	0	0	0.04	0	0	0.06	0	0	0.06
$g_3^{10}g_4^{10}$	0	0	0.07	0	0	0.09	0	0	0.1
$g_3^5g_4^{15}g_5^5g_4^3g_5$	$7.45e-9$	0.035	0.12	$7.45e-9$	0.035	0.2	$2.6e-8$	0.103	0.2
$g_3^4g_4^{25}g_5^{25}$	3.7e-9	0.019	0.15	3.73e-9	0.019	0.4	6.52e-9	0.056	0.3

assume that mistakes occur in bursts of at least 2 but at most 3 mistakes at a time. These models are useful in capturing fine grained assumptions about errors that are often the case in the study of human error or errors in physical systems.

Using the prototype implementation, we report on the robustness of various inputs for this motivating example under the three transducer error models. The property P is as shown in Figure 3 and the transducers $T_0 - T_2$ are as shown in Fig. 4. Table 1 reports the robustness values for various input strings and the running time. We note that while our approach takes about 0.3 seconds for a string of length 50, the prototype can be made much more efficient to reduce the time to compute robustness. Also we note that discounted sum becomes smaller as the strings grow larger while the average robustness value does not. We conclude that average robustness is a more useful measure due to this property in this particular example.

5.2 Robust Pattern Matching in Type-1 Diabetes Data

We will now apply our ideas to the *robust pattern matching* problem for analyzing clinical data for patients with type-1 diabetes. People with type-1 diabetes are required to monitor their blood glucose levels periodically using devices such as continuous glucose monitors (CGMs). Data from CGMs is uploaded online and available for review by clinicians during periodic doctor visits. Many applications such as Medtronic Carelink(tm) support the automatic upload and visualization of this data by clinicians. Physicians are commonly

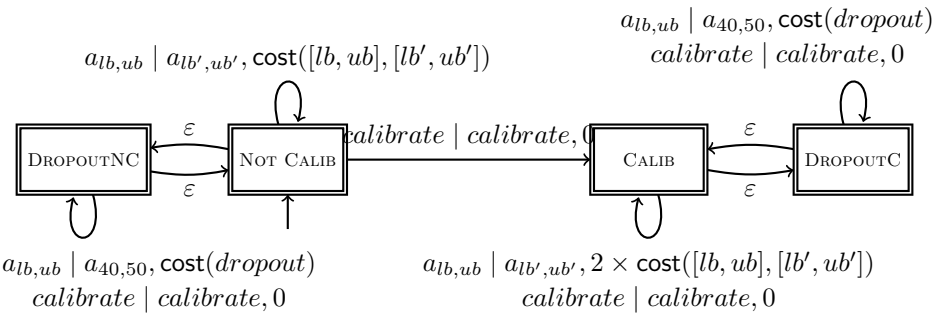


Figure 5 Transducer model for capturing the errors made by continuous glucose monitors.

interested in analyzing the data to reveal potentially dangerous patterns of blood glucose levels: (a) *Prolonged Hypoglycemia (P1):* Do the blood glucose levels stay below 70 mg/dl (hypoglycemia) for more than 3 hours continuously? [3] (b) *Prolonged Hyperglycemia (P2):* Do the blood glucose levels remain above 300 mg/dl (hyperglycemia) for more than 3 hours continuously? [4]; and (c) *Rebound Hyperglycemia (P3):* Do the blood glucose levels go below 70 mg/dl and then rise rapidly up to 300 mg/dl or higher within 2 hours? [5]

Note that these patterns specify "bad" events that should not happen. A straightforward and strict pattern matching approach based on specifying the properties above will "hide" potentially bad scenarios that "nearly" match the desired pattern for two main reasons. First, the CGM can be noisy and inaccurate in a way that depends on the actual blood glucose value measured and when it was last calibrated. (see Figure 5 and more detailed description below). Secondly, the cutoffs involved such as 70 mg/dl and 3 hours are not "set in stone". For instance, a clinician will consider a scenario wherein the patient's blood glucose levels stays at 71 mg/dl for 2.75 hours as a serious case of prolonged hypoglycemia even though such a scenario would not satisfy the property P1.

We propose to solve the approximate "pattern matching" problem. I.e, given a string w, a transducer T and a language L, we are looking for a word w' such that $w' \in L$ and $C_T(w', w)$ is *as small as possible*. In other words, we solve the threshold synthesis problem (Problem 6) for a language L that is the complement of P1 (P2 or P3).

We partition the range of CGM outputs $[40, 400]$ mg/dl into intervals of size 10 mg/dl over the range $[40, 80]$ mg/dl and 20 mg/dl intervals over the remaining range $[80, 400]$ mg/dl. This yields a finite alphabet Σ where $|\Sigma| = 20$. For instance $a_{60,70} \in \Sigma$ represents a range $[60, 70] mg/dl$. CGMs provide a reading periodically at 5 minute intervals. This yields a string where each letter describes the interval that contains the glucose value.

Transducer. The CGM error model is given by a transducer that considers possible errors that a CGM can make (see Fig. 5). The transducer has four states: (a) Not Calib denoting that no calibration has happened, (b) Calib: denoting a calibration event in the past, (c) DropoutNC: a sensor drops out under the non calibrated mode and (d) DropoutC: a calibration event has happened and sensor drops out. The cost of changing a reading in the range $[lb, ub]$ to one in the range $[lb', ub']$ is denoted by a function $\mathsf{cost}(lb, ub, lb', ub')$ These

[3] Such an event can lead to dangerous (and silent) nighttime seizures.
[4] Such an event can lead to a potentially dangerous condition called diabetic ketacidosis.
[5] Rebound hyperglycemia can lead to large future swings in the blood glucose level, raising the burden on the patient for managing their blood glucose levels.

costs are set to be higher for ranges $[lb, ub]$ that are close to hypoglycemia. Also note that we can model calibration events and the doubling of costs if the sensor is in the calibrated mode.

Property Specifications. We specify the three different properties described above formally using finite state machines over the alphabet Σ as defined above. The prolonged hypoglycemia property can be written as a regular expression: $\Sigma^*(a_{40,50} + a_{50,60} + a_{60,70})^{36}\Sigma^*$ which can be easily translated into an NFA with roughly 38 states. The number 36 represents a period of 180 minutes since CGM values are sampled at 5 minute intervals. Similarly, the other two properties are also easily expressed as NFAs.

Finally, we compose the transducer model with the properties P1-P3 individually and calculate the mean robustness. More precisely, for each sequence of measures w, we compute the minimal threshold ν such that w can be rewritten by T at mean cost ν into some w' satisfying P1 (and P2, P3 respectively). The discounted sum robustness is not useful in this situation since the patterns can match approximately anywhere in the middle of a trace. Also, in most cases the discounted sum robustness value was very close to zero for any discount factor < 1 or became forbiddingly large for discount factors slightly larger than 1, due to the large size of the traces.

Patient Data. We used actual patient data involving nearly 50 patients with type-1 diabetes undergoing a clinical trial of an artificial pancreas device, and nearly 40 nights of data per patient, leading to an overall 2032 nights. Each night roughly corresponds to a 12 hour period when CGM data was recorded [20]. This is converted to a string of size 140 (or slightly larger, depending on how many calibration events occurred). The threshold synthesis problem (Problem 6) was solved for each of the input strings, and the results were sorted by the threshold robustness value for properties P1-P3.

Table 2 Total time taken per property and number of matches for various ranges of the threshold.

Prop.	Total Time	Threshold Values synthesized				
		0	$(0, 0.1]$	$(0.1, 1.0]$	> 1.0	∞
$P1$	$4hr10m31s$	0	8	2	95	1927
$P2$	$2hr10m30s$	0	28	13	0	1991
$P3$	$2h0m9s$	0	11	10	0	2011

Table 2 shows for each property, the total time taken to complete the analysis of the full patient data, and the number of matches obtained corresponding to various threshold values. As the table reveals, *no single trace matches any of the properties perfectly.* However, our approach is more nuanced, and thus, allows us to find numerous approximate matches that can be sorted by their robustness threshold values. Note that many of the input traces yield a threshold value of ∞: this signifies that no possible translation as specified by the transducer can cause the property to hold.

Figure 6 shows two of the approximate pattern matches obtained with a small robustness value. Notice that the CGM values on the left do not satisfy the criterion for a "prolonged hypoglycemia" for 3 hours (P1) in a strict sense due to a single point at the end of the trace that is slightly above the 70 mg/dl threshold. Nevertheless, our approach assigns this trace a very low robustness. Likewise, the plot on the right shows a rapid rise from a hypoglycemia to a hyperglycemia within 120 minutes (P3) towards the beginning, except that the peak value just falls short of the threshold of 300 mg/dl.

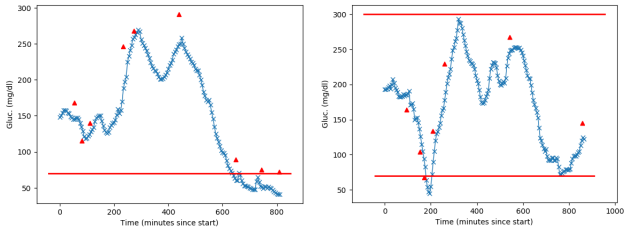


Figure 6 Examples of patterns with small robustness thresholds for properties P1 (left) with robustness value of 0.7, and P3 (right) with robustness 0.02. The red triangles show calibration events.

Note that related work in the area of monitoring cyber-physical systems (CPS) mentioned earlier [16, 14, 12, 1] can be used to perform approximate pattern matching using robustness of temporal properties over hybrid traces. However, we note important differences that are achieved due to the theory developed in this paper. For one, the use of a transducer can provide a nuanced model of how errors transform a trace, wherein the transformation itself changes based on the transducer state. A detailed transducer model of CGM errors remains beyond the scope of this study but will likely be desirable for applications to the analysis of patterns in type-1 diabetes data.

6 Conclusion

In conclusion, we have shown how notions of robustness can be defined through weighted transducers along with approaches for solving the threshold and kernel synthesis problems for various cost aggregators such as `Sum`, `DSum` and `Mean`. In the future, we will investigate these notions for richer classes of systems including timed and hybrid systems. We also plan to investigate connections to robust learning of automata from examples.

References

1 Takumi Akazaki and Ichiro Hasuo. Time robustness in MTL and expressivity in hybrid system falsification. In *Computer Aided Verification (CAV)*, volume 9207 of *Lecture Notes in Computer Science*, pages 356–374. Springer, 2015.

2 Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? In Tevfik Bultan and Pao-Ann Hsiung, editors, *Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings*, volume 6996 of *Lecture Notes in Computer Science*, pages 482–491. Springer, 2011. doi:10.1007/978-3-642-24372-1_37.

3 Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A Henzinger, P-H Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 138(1):3–34, 1995.

4 Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

5 Rajeev Alur, Sampath Kannan, Kevin Tian, and Yifei Yuan. On the complexity of shortest path problems on discounted cost graphs. In *Language and Automata Theory and Applications - 7th International Conference, LATA 2013, Bilbao, Spain, April 2-5, 2013. Proceedings*, pages 44–55, 2013.

6 Rajeev Alur and Parthasarathy Madhusudan. Visibly pushdown languages. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 202–211. ACM, 2004.

7 Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donze, Georgios Fainekos, Oded Maler, Dejan Nickovic, and Sriram Sankaranarayanan. Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications. In *Lectures on Runtime Verification*, volume 10457 of *LNCS*, pages 135–175, 2018.

8 Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In *International Conference on Concurrency Theory*, pages 135–150. Springer, 1997.

9 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Expressiveness and closure properties for quantitative languages. *Logical Methods in Computer Science*, 6(3), 2010. arXiv:1007.4018.

10 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Logic*, 11(4):23:1–23:38, July 2010. doi:10.1145/1805950.1805953.

11 Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018.

12 Jyotirmoy V. Deshmukh, Rupak Majumdar, and Vinayak S. Prabhu. Quantifying conformance using the skorokhod metric. *Formal Methods Syst. Des.*, 50(2-3):168–206, 2017.

13 Michel Marie Deza and Elena Deza. *Encyclopedia of Distances*. Springer, 2009.

14 Alexandre Donze and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *Formal Modeling and Analysis of Timed Systems*, volume 6246 of *LNCS*, pages 92–106. Springer, 2010.

15 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of weighted automata*. Springer Science & Business Media, 2009.

16 Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

17 Thomas A. Henzinger, Jan Otop, and Roopsha Samanta. Lipschitz robustness of finite-state transducers. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 431–443. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.431.

18 Richard M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23(3):309–311, 1978. doi:10.1016/0012-365X(78)90011-0.

19 Dexter Kozen. Lower bounds for natural proof systems. In *Foundations of Computer Science*, pages 254–266, 1977.

20 David M. Maahs, Peter Calhoun, Bruce A. Buckingham, H. Peter Chase, Irene Hramiak, John Lum, Fraser Cameron, B. Wayne Bequette, Tandy Aye, Terri Paul, Robert Slover, R. Paul Wadwa, Darrell M. Wilson, Craig Kollman, and Roy W. Beck. A randomized trial of a home system to reduce nocturnal hypoglycemia in type 1 diabetes. *Diabetes Care*, 37(7):1885–1891, 2014. doi:10.2337/dc13-2159.

21 Aurelien Rizk, Gregory Batt, Francois Fages, and Sylvain Soliman. Continuous valuations of temporal logic specifications with applications to parameter optimization and robustness measures. *Theor. Comput. Sci.*, 412(26):2827–2839, 2011.

22 Roopsha Samanta, Jyotirmoy V. Deshmukh, and Swarat Chaudhuri. Robustness analysis of string transducers. In Dang Van Hung and Mizuhito Ogawa, editors, *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*, volume 8172 of *Lecture Notes in Computer Science*, pages 427–441. Springer, 2013. doi:10.1007/978-3-319-02444-8_30.

23 Stefan Schwoon. *Model-checking pushdown systems*. PhD thesis, Technische Universität München, 2002.

24 Masaki Waga, Étienne André, and Ichiro Hasuo. Symbolic monitoring against specifications parametric in time and data. In *Computer Aided Verification*, pages 520–539, Cham, 2019. Springer International Publishing.

Appendix

Proof of Lemma 10

Proof. We first trim the graph G by removing all the vertices that cannot be reached from V_I and that cannot reach V_F as those vertices cannot participate to paths from V_I to V_F. The set of paths from V_I to V_F is empty iff the trimmed graph is empty and then the infimum is equal to $+\infty$. Now, we assume the trimmed graph to be non-empty, i.e. there is at least one path from V_I to V_F. In that case, the infimum value is guaranteed to be a non-negative rational number.

We now consider the three measures in turn. For `Sum`, computing the infimum amounts to computing a shortest path in a finite graph with non-negative weights. Any PTIME algorithm that solves this problem can be used, e.g. Dijkstra shortest path algorithm. In the case of sum, the infimum is always realized by a (simple) shortest path.

For `Mean`, we first note that the infimum is either realized by a simple path from V_I to V_F of minimal `Mean` value, or it is equal to the minimal `Mean` value among the simple cycles in the graph. Indeed, if c is a cycle of `Mean` m which is smaller than the `Mean` value of any path from V_I to V_F then the family of paths $\rho_k = p \cdot c^k \cdot s$, where p is simple path from V_I to c and s is a simple path from c to V_F (such simple paths exist as the graph is trimmed), is such that $\lim_{k \mapsto +\infty} \texttt{Mean}(\rho_k) = \texttt{Mean}(c)$ and $\texttt{Mean}(c)$ is the infimum. Now if all the simple cycles have a value larger than the infimum, they cannot participate to a path or a family of paths that realize the infimum as those cycles can be systematically removed and give paths with smaller values. Now, we note that the minimum value of simple paths from V_I to V_F can be computed in PTIME by a simple dynamic program that considers the minimal values of paths of lengths at most equal to the number of states in the trimmed graph. Moreover, the minimum mean value of simple cycles in the trimmed graph can be computed in PTIME using the Karp algorithm [18]. It is easy to see that the infimum is feasible iff it equals the minimum `Mean` value of simple paths.

We now turn to the `DSum` measure. Remember that the graph is trimmed according to V_I and V_F. Theorem 1 of [5] tells us that we can compute for all $v \in V$, the infimum of `DSum` values x_v of paths reaching the target V_F from v, in PTIME. According to Lemma 1 of [5], and similarly to the case of `Mean`, for all $v_I \in V_I$, the infimum `DSum` value x_{v_I} of paths from v_I to some $v_F \in V_F$ is either realized by a simple path or by a family of paths of the form $p \cdot c^k \cdot s$. This is because if it is beneficial to include a cycle c to reduce the cost of a path from v_I to v_F then it is beneficial to repeat the cycle arbitrarily many times. In particular, the infimum value is feasible only when there exists a simple path with this value. In order to decide the feasibility of the values x_{v_I} for all $v_I \in V_I$, we consider a subgraph where we keep only those edges $e = (v, v')$ such that the optimal value x_v of v can be realised through the vertex v'. Formally, we construct $G' = (V, E')$ with $E' \subseteq E$ and such that $(v, v') \in E'$ if $x_v = \lambda x_{v'} + \texttt{W}(v, v')$. We claim that, V_F is reachable from v in G' iff x_v is feasible in G from v, hence testing feasibility boils down to checking the existence of a path in G'.

The left-to-right implication comes by induction on the length of the path π to reach some $v_F \in V_F$ from v. If $v \in V_F$ then $|\pi| = 0$, $x_v = 0$ and this value is feasible. Assume $v \notin V_F$ and $\pi = (v, v')\pi'$. By induction hypothesis, $x_{v'}$ is feasible by some path π'' from v' to V_F. By construction of G' we have $x_v = \lambda x_{v'} + \texttt{W}(v, v')$. Hence x_v is feasible by $(v, v')\pi''$. For the right-to-left implication, if $v \in V_F$ it is trivial, so assume that $v \notin V_F$ and let $\pi = (v, v')\pi'$ a path that realises x_v. Assume $x_v > \lambda x_{v'} + \texttt{W}(v, v')$. This contradicts the optimality of x_v, as π witnesses a better discounted value from v to V_F. Assume $x_v < \lambda x_{v'} + \texttt{W}(v, v')$, then since π realises x_v, we have $x_v = \texttt{W}(v, v') + \lambda \texttt{DSum}(\pi')$. It implies $\texttt{DSum}(\pi') < x_{v'}$. This contradicts

the minimality of $x_{v'}$, as then π' witnesses a better value for paths from v' to V_F. Hence $x_v = \lambda x_{v'} + \mathtt{W}(v, v')$ and (v, v') is an edge of G'. By induction on the length of π, we can also conclude that π' is a path of G' and then π is a path of G' from v to V_F. ◀

Proof of Lemma 14

Proof. First, we show that the complement of $\mathtt{Rob}_T(\nu, L)$, defined as

$$\overline{\mathtt{Rob}_T(\nu, L)} = \{w_1 \mid \exists w_2 \cdot \mathsf{Sum}_T(w_1, w_2) < \nu \wedge w_2 \notin L\}$$

is regular. First, let us assume that L is given by some NFA A, let $\overline{A}$ be a DFA recognizing the complement of L. We first transform T into $T \otimes \overline{A}$, which simulates T and controls that the output words belong to $\overline{L}$. In particular, it rejects whenever the rewriting by T is in L. It is obtained as a product of T with $\overline{A}$ run on the output, with set of states $Q_T \times Q_{\overline{A}}$. It accepts whenever the final pair of states (p, q) is a pair of accepting states both for T and $\overline{A}$. Then, we have the following:

$$\overline{\mathtt{Rob}_T(\nu, L)} = \{w_1 \mid \exists w_2 \cdot \mathsf{Sum}_{T \otimes \overline{A}}(w_1, w_2) < \nu\}$$

Now, by definition of $\mathsf{Sum}_{T \otimes \overline{A}}(w_1, w_2)$ we have $w_1 \in \overline{\mathtt{Rob}_T(\nu, L)}$ iff there exists a word w_2 and an accepting run r over (w_1, w_2) such that $\mathsf{Sum}(r) < \nu$. Therefore, we can project $T \otimes \overline{A}$ on its input dimension (thus, we just ignore the outputs) and obtain a $\mathtt{Sum}$-automaton that we call U such that $\overline{\mathtt{Rob}_T(\nu, L)} = \{w_1 \mid U(w_1) < \nu\}$, where $U(w_1)$ is defined as $+\infty$ if there is no accepting run of U on w_1, and as the minimal sum of the accepting runs on w_1 otherwise. Complementing again, we get: $\mathtt{Rob}_T(\nu, L) = \{w_1 \mid U(w_1) \geq \nu\}$. Now, we apply directly Lemma 13 on U to conclude for regularity. The state-complexity is again given by Lemma 13 and the fact that U has $n_T \times n_L$ states. ◀

Proof of Lemma 18

Proof. Let r be a run of length n of T. Since T is trim, there exists a continuation r' of r, and moreover we have $\mathtt{DSum}(rr') = \mathtt{DSum}(r) + \lambda^n \mathtt{DSum}(r')$. We have $\mathtt{DSum}(r') \leq \sum_{i=0}^{+\infty} \lambda^i \mu = \mu(1-\lambda)^{-1}$ where μ is the largest absolute weight of T. We let $B_n = \lambda^n \mu (1-\lambda)^{-1}$. Let n^* be the smallest non-negative integer such that $B_{n^*} \leq \epsilon/2$ (it exists since B_n is strictly decreasing of limit 0). Assume that the length of r is greater than n^* i.e. $n \geq n^*$. As a consequence $B_n \leq B_{n^*}$. Since ν is ϵ-isolated, we have two cases:

- **i.** If $\mathtt{DSum}(rr') \leq \nu - \epsilon$ then $\mathtt{DSum}(r) \leq \nu - \epsilon$ since $\mathtt{DSum}(r) \leq \mathtt{DSum}(rr')$ by non-negativity of the weights of T
- **ii.** If $\mathtt{DSum}(rr') \geq \nu + \epsilon$ then $\mathtt{DSum}(r) \geq \nu + \epsilon - \lambda^n \mathtt{DSum}(r')$. Moreover $\lambda^n \mathtt{DSum}(r') \leq B_n \leq B_{n^*} \leq \epsilon/2$ by construction. So $-\lambda^n \mathtt{DSum}(r') \geq -\epsilon/2$ which implies $\mathtt{DSum}(r) \geq \nu + \epsilon/2$.

We have just shown that either $\mathtt{DSum}(r) \leq \nu - \epsilon$ by (i) or $\mathtt{DSum}(r) \geq \nu + \epsilon/2$ by (ii). We prove now that, for all continuation r' of r we have (i) implies $\mathtt{DSum}(rr') \leq \nu - \epsilon$ and (ii) implies $\mathtt{DSum}(rr') \geq \nu + \epsilon$. In the first case, assume by contradiction that (i) holds and some continuation r' of r satisfies $\mathtt{DSum}(rr') \geq \nu + \epsilon$. As a consequence $\lambda^n \mathtt{DSum}(r') \geq 2\epsilon$, which is impossible since $\lambda^n \mathtt{DSum}(r') \leq B_n \leq B_{n^*} \leq \epsilon/2$. In the second case, if $\mathtt{DSum}(r) \geq \nu + \epsilon/2$ then any continuation r' of r satisfies $\mathtt{DSum}(rr') \geq \mathtt{DSum}(r) > \nu + \epsilon/2$. Since ν is ϵ-isolated, we get $\mathtt{DSum}(rr') \geq \nu + \epsilon$. ◀

Proof of Lemma 19

Proof. For all n, we let $B_n = \lambda^n W (1-\lambda)^{-1}$, as in the proof of Lemma 18. A run r on a pair (w_1, w_2) is called *bad* if $\mathtt{DSum}(r) \leq \nu$, $w_2 \notin L$ and r is accepting. Not that necessarily,

$w_1 \notin \mathtt{Rob}_T(\nu, L)$. The run r is called *dangerous* if $|r| \geq n$ and $\mathtt{DSum}(r) \leq \nu - B_n$. A dangerous run r can possibly be extended to a bad run rr'. It is possible iff there exists a continuation r' of r such that the output of rr' is not in L. Note that the cost of rr' does not matter because the largest value r' can achieve is B_n, keeping $\mathtt{DSum}(rr')$ smaller than ν. Hence, when a dangerous run is met, only a regular property has to be tested to extend it to a bad run. We exploit this idea in the automata construction. Namely, A_n will accept words for which there exists a bad run of length n at most, or a dangerous run of length n which can be extended to a bad run.

- *Automata construction.* Let $\mathsf{Runs}_T^{\leq n}$ be the runs of T of length at most n, and Q its set of states. We assume that for all $(w_1, w_2) \in R_T$, $w_2 \notin L$ holds. This can be ensured by taking the synchronised product of T (on its outputs) with an automaton recognizing the complement of L. Let us now build the NFA A_n. Its set of states is $\mathsf{Runs}_T^{\leq n} \cup Q$. Its transitions are defined as follows: for all T-runs r of length $n-1$ at most ending in some state q, for all $\sigma \in \Sigma_\varepsilon$, if there exists a transition t of T from state q on reading σ, then we create the transition $r \xrightarrow{\sigma} rt$ in A_n. From any run r of length n, we consider two cases: if r is not dangerous, then r has no outgoing transitions in A_n. If r is a dangerous run, then we add some ε-transition to its last state: $r \xrightarrow{\varepsilon} p$ where p is the last state of r. Finally, we add a transition from any state q to any state q' on σ in A_n whenever there is a transition from q to q' on input σ in T. Accepting states are bad runs of $\mathsf{Runs}_T^{\leq n}$ and accepting states of T.
- *Correctness.* Let us show that the family A_n satisfies the requirements of the lemma. First, we show that $L(A_n) \subseteq L(A_{n+1})$. Let $w \in L(A_n)$ and ρ some accepting run of A_n on w. To simplify the notations, we assume here in this proof that runs of A_n, A_{n+1} and T are just sequences of states rather than sequences of transitions. By definition of A_n, ρ can be decomposed into two parts $\rho_1\rho_2$ such that $\rho_1 \in (\mathsf{Runs}_T^{\leq n})^*$ and $\rho_2 \in Q^*$ with an ϵ-transition from the last state of ρ_1 to the first of ρ_2. We consider two cases. If $|\rho_2| = 0$, then $\rho = \rho_1$ and by definition of A_{n+1}, ρ is still an accepting run of A_{n+1}. In the other case, there is a dangerous run r of T such that ρ_1 can be written $\rho_1 = r[:1]r[:2]\dots r[:n]$ where $r[:i]$ is the prefix of r up to position i, and $\rho_2 = q_1q_2\dots q_k$ is a proper run of T. Note that q_1 is the last state of r by construction of A_n. Moreover, $r\rho_2$ is bad. Since r was dangerous at step n, we also get that rq_2 is dangerous at step $n+1$, in the sense that $|rq_2| = n+1$ and $\mathtt{DSum}(rq_2) \leq \nu - B_{n+1}$, by definition of B_{n+1} and the fact that $\mathtt{DSum}(r) \leq \nu - B_n$. So, we get that the sequence of states $\rho_1.(rq_2).q_2\dots q_k$ is a run of A_{n+1} on w is accepting in A_{n+1} (note that rq_2 here is a state of A_{n+1} and there is an ϵ-transition from (rq_2) to q_2), concluding the first part of the proof.

Now, suppose that ν is ϵ-isolated for some ϵ. Then, take n^* as given by Lemma 18 and let us show that $\overline{\mathtt{Rob}_T(\nu, L)} \cap \mathrm{dom}(T) \subseteq L(A_{n^*})$ (the other inclusion has just been proved for all n). Let $w \in \mathrm{dom}(T)$ such that $w \notin \mathtt{Rob}_T(\nu, L)$. There exists $(w_1, w_2) \in R_T$ and an accepting run r of T on it such that $\mathtt{DSum}(r) \leq \nu$ and $w_2 \notin L$. In other words, r is bad. If $|r| \leq n^*$, then $r[:1]r[:2]\dots r[:|r|]$ is an accepting run of A_{n^*} on w, and we are done. Now suppose that $|r| > n^*$. Since ν is ϵ-isolated, we have $\mathtt{DSum}(r) \leq \nu - \epsilon$. By Lemma 18, we also get that $\mathtt{DSum}(r[:n^*]) \leq \nu - \epsilon$. By definition of n^* being the smallest integer such that $B_{n^*} < \epsilon/2$, we get $\mathtt{DSum}(r[:n^*]) \leq \nu - B_{n^*}$, hence $r[:n^*]$ is dangerous. We can conclude since then $r[:1]r[:2]\dots r[:n^*]r[n^*]r[n^*+1]\dots r[|r|]$ is an accepting run of A_{n^*} on w. ◀

On the Axiomatisability of Parallel Composition: A Journey in the Spectrum

Luca Aceto
Reykjavik University, Iceland
Gran Sasso Science Institute, L'Aquila, Italy

Valentina Castiglioni
Reykjavik University, Iceland

Anna Ingólfsdóttir
Reykjavik University, Iceland

Bas Luttik
Eindhoven University of Technology, The Netherlands

Mathias Ruggaard Pedersen
Reykjavik University, Iceland

Abstract

This paper studies the existence of finite equational axiomatisations of the *interleaving parallel composition operator* modulo the behavioural equivalences in van Glabbeek's *linear time-branching time spectrum*. In the setting of the process algebra BCCSP over a finite set of actions, we provide *finite*, ground-complete axiomatisations for various simulation and (decorated) trace semantics. On the other hand, we show that no congruence over that language that includes bisimilarity and is included in possible futures equivalence has a finite, ground-complete axiomatisation. This *negative result* applies to all the nested trace and nested simulation semantics.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting

Keywords and phrases Axiomatisation, Parallel composition, Linear time-branching time spectrum

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.18

Funding This work has been supported by the project "*Open Problems in the Equational Logic of Processes*" (OPEL) of the Icelandic Research Fund (grant No. 196050-051).

Acknowledgements We thank the anonymous reviewers for their valuable comments, and Rob van Glabbeek for a fruitful discussion on the axiomatisability of failures equivalence.

1 Introduction

Process algebras [4, 6] are prototype specification languages allowing for the description and analysis of concurrent and distributed systems, or simply *processes*. Briefly, the *operational semantics* [26] of a process is modelled via a *labelled transition system* (LTS) [20] in which the computational steps are abstracted into state-to-state transitions having actions as labels. Notably, in order to model the concurrent interaction between processes, the majority of process algebras include some form of *parallel composition operator*, also known as *merge*.

Behavioural equivalences have then been introduced as simple and elegant tools for comparing the behaviour of processes. These are equivalence relations defined on the states of LTSs allowing one to establish whether two processes have the same *observable behaviour*. Different notions of observability correspond to different levels of abstraction from the information carried by the LTS, which can either be considered irrelevant in a given application context, or be unavailable to an external observer.

In [16], van Glabbeek presented the *linear time-branching time spectrum*, namely a taxonomy of behavioural equivalences based on their distinguishing power. He carried out his study in the setting of the process algebra BCCSP, which consists of the basic operators from CCS [21] and CSP [19], and he proposed *ground-complete axiomatisations* for most of the congruences in the spectrum over this language. (An axiomatisation is ground-complete if it can prove all the valid equations relating terms that do not contain process variables.) The presented ground-complete axiomatisations are *finite* if so is the set of actions. For ready simulation, ready trace and failure trace equivalences, the axiomatisation in [16] made use of conditional equations. Blom, Fokkink and Nain gave purely equational, finite axiomatisations of those equivalences in [7]. Then, the works in [1], on nested semantics, and in [8], on impossible futures semantics, completed the studies of the axiomatisability of behavioural congruences over BCCSP by providing *negative* results: neither impossible futures nor any of the nested semantics have a finite, ground-complete axiomatisation over BCCSP.

Obtaining a complete axiomatisation of a behavioural congruence is a classic, key problem in concurrency theory, as it allows for characterising the semantics of a process algebra in a purely syntactic fashion. Hence, this characterisation becomes independent of the details of the definition of the process semantics of interest.

All the results mentioned so far were obtained over the algebra BCCSP that does not include any operator for the parallel composition of processes. Considering the crucial role of such an operator, it is natural to ask which of those results would still hold over a process algebra including it.

In the literature, we can find a wealth of studies on the axiomatisability of parallel composition modulo *bisimulation semantics* [25]. Briefly, in the seminal work [18], Hennessy and Milner proposed a ground-complete axiomatisation of (a part of) CCS modulo bisimilarity. That axiomatisation, however, included infinitely many axioms, which corresponded to instances of the *expansion law* used to express equationally the semantics of the merge operator. Then, Bergstra and Klop showed in [5] that a finite ground-complete axiomatisation modulo bisimilarity can be obtained by enriching CCS with two auxiliary operators, i.e., the *left merge* $\between$ and the *communication merge* $|$. Later, Moller proved that the use of auxiliary operators is indeed necessary to obtain a finite equational axiomatisation of bisimilarity in [22–24].

To the best of our knowledge, no systematic study of the axiomatisability of the parallel composition operator modulo the other semantics in the spectrum has been presented so far.

Our contribution. We consider the process algebra $\mathrm{BCCSP}_{\parallel}$, namely BCCSP enriched with the interleaving parallel composition operator, and we study the existence of finite equational axiomatisations of the behavioural congruences in the linear time-branching time spectrum over it. Our results delineate the boundary between finite and non-finite axiomatisability of the congruences in the spectrum over the language $\mathrm{BCCSP}_{\parallel}$. (See Figure 1.)

We start by providing a *finite*, *ground-complete* axiomatisations for *ready simulation* semantics. The axiomatisation is obtained by extending the one for BCCSP with a few axioms expressing equationally the behaviour of interleaving modulo the considered congruence. The added axioms allow us to eliminate all occurrences of the interleaving operator from $\mathrm{BCCSP}_{\parallel}$ processes, thus reducing ground-completeness over $\mathrm{BCCSP}_{\parallel}$ to ground-completeness over BCCSP [7, 16]. Since the axioms for the elimination of parallel composition modulo ready simulation equivalence are of course sound with respect to the coarser equivalences, the reduction works for all behavioural equivalences below ready simulation equivalence. Nevertheless, we shall find more elegant ways to do the reduction for the coarser equivalences

in the spectrum. We shall then observe a sort of parallelism between the axiomatisations for the notions of simulation and the corresponding decorated trace semantics: the axioms used to express equationally the interleaving operator in a decorated trace semantics can be seen as the *linear counterpart* of those used in the corresponding notion of simulation semantics. For instance, while the axioms for ready simulation impose constraints on the form of both arguments of the interleaving operator to trigger the reductions, those for ready trace equivalence impose similar constraints but only on one argument.

Then, we complete our journey in the spectrum by showing that *nested simulation* and *nested trace* semantics do not have a finite axiomatisation over $\mathrm{BCCSP}_{\|}$. To this end, firstly we adapt Moller's arguments to the effect that bisimilarity is not finitely based over CCS to obtain the *negative result* for *possible futures equivalence*, also known as 2-*nested trace equivalence*. Informally, the negative result is obtained by providing an infinite family of equations that are all sound modulo possible futures equivalence but that cannot all be derived from any finite sound axiom system. Then, we exploit the soundness of the equations in the family modulo bisimilarity to extend the negative result to all the congruences that are finer than possible futures and coarser than bisimilarity, thus including all nested trace and nested simulation semantics.

Organisation of contents. After reviewing some basic notions on behavioural equivalences and equational logic in Section 2, we start our journey in the spectrum by providing a finite, ground-complete axiomatisation for ready simulation equivalence over $\mathrm{BCCSP}_{\|}$ in Section 3. In Section 4 we discuss how it is possible to refine the axioms for ready simulation to obtain finite, ground-complete axiomatisations for completed simulation and simulation equivalences. Then, in Section 5 similar refinements are provided for the (decorate) trace equivalences, thus completing the presentation of our positive results. We end our journey in Section 6 with the presentation of the negative results, namely that the nested simulation and nested trace equivalences do not have a finite axiomatiosation over $\mathrm{BCCSP}_{\|}$. Finally, in Section 7 we draw some conclusions and discuss avenues for future work.

2 Background

The language $\mathbf{BCCSP}_{\|}$. The language $\mathrm{BCCSP}_{\|}$ extends BCCSP with parallel composition. Formally, $\mathrm{BCCSP}_{\|}$ consists of basic operators from CCS [21] and CSP [19], with the purely *interleaving* parallel composition operator $\|$, and is given by the following grammar:

$$t ::= \mathbf{0} \mid x \mid a.t \mid t + t \mid t \,\|\, t$$

where a ranges over a set of actions $\mathcal{A}$ and x ranges over a countably infinite set of variables $\mathcal{V}$. In what follows, we assume that the set of actions $\mathcal{A}$ is *finite*.

We shall use the meta-variables $t, u, \ldots$ to range over $\mathrm{BCCSP}_{\|}$ terms, and write $\mathrm{var}(t)$ for the collection of variables occurring in the term t. We also adopt the standard convention that prefixing binds strongest and $+$ binds weakest. Moreover, trailing $\mathbf{0}$'s will often be omitted from terms. We use a *summation* $\sum_{i\in\{1,\ldots,k\}} t_i$ to denote the term $t = t_1 + \cdots + t_k$, where the empty sum represents $\mathbf{0}$. We can also assume that the terms t_i, for $i \in \{1, \ldots, k\}$, do not have $+$ as head operator, and refer to them as the *summands* of t. The *size* of a term t, denoted by $\mathrm{size}(t)$, is the number of operator symbols in it.

A $\mathrm{BCCSP}_{\|}$ term is *closed* if it does not contain any variables. We shall, sometimes, refer to closed terms simply as *processes*. We let $\mathcal{P}$ denote the set of $\mathrm{BCCSP}_{\|}$ processes and let $p, q, \ldots$ range over it. We use the *Structural Operational Semantics* (SOS) framework [26]

Table 1 Operational semantics of $\mathrm{BCCSP}_{\|}$.

$$\frac{}{a.x \xrightarrow{a} x} \qquad \frac{x \xrightarrow{a} x'}{x + y \xrightarrow{a} x'} \qquad \frac{y \xrightarrow{a} y'}{x + y \xrightarrow{a} y'} \qquad \frac{x \xrightarrow{a} x'}{x \,\|\, y \xrightarrow{a} x' \,\|\, y} \qquad \frac{y \xrightarrow{a} y'}{x \,\|\, y \xrightarrow{a} x \,\|\, y'}$$

to equip processes with an operational semantics. A *literal* is an expression of the form $t \xrightarrow{a} t'$ for some process terms t, t' and action $a \in \mathcal{A}$. It is *closed* if both t, t' are closed terms. The inference rules for *prefixing* $a.\_$, *nondeterministic choice* $+$ and *interleaving parallel composition* $\|$ are reported in Table 1. A *substitution* σ is a mapping from variables to terms. It extends to terms, literals and rules in the usual way and it is *closed* if it maps every variable to a process.

The inference rules in Table 1 induce the $\mathcal{A}$-*labelled transition system* [20] $(\mathcal{P}, \mathcal{A}, \rightarrow)$ whose transition relation $\rightarrow \subseteq \mathcal{P} \times \mathcal{A} \times \mathcal{P}$ contains exactly the closed literals that can be derived using the rules in Table 1. As usual, we write $p \xrightarrow{a} p'$ in lieu of $(p, a, p') \in \rightarrow$. For each $p \in \mathcal{P}$ and $a \in \mathcal{A}$, we write $p \xrightarrow{a}$ if $p \xrightarrow{a} p'$ holds for some p', and $p \not\xrightarrow{a}$ otherwise. The *initials* of p are the actions that label the outgoing transitions of p, that is, $\mathtt{I}(p) = \{a \mid p \xrightarrow{a}\}$. For a sequence of actions $\alpha = a_1 \cdots a_k$ $(k \geq 0)$, and processes p, p', we write $p \xrightarrow{\alpha} p'$ if and only if there exists a sequence of transitions $p = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} p_k = p'$. If $p \xrightarrow{\alpha} p'$ holds for some process p', then α is a *trace* of p, and p' is a *derivative* of p. Moreover, we say that α is a *completed trace* of p if $\mathtt{I}(p') = \emptyset$. We let $\mathtt{T}(p)$ denote the set of traces of p, and we let $\mathtt{CT}(p) \subseteq \mathtt{T}(p)$ denote the set of completed traces of p. We let ε denote the empty trace, and $|\alpha|$ denote the length of trace α. It is well known, and easy to show, that $\mathtt{T}(p)$ is finite for each $\mathrm{BCCSP}_{\|}$ process p. It follows that we can define the *depth* of a process p, denoted by $\mathrm{depth}(p)$, as the length of a *longest* completed trace of p. Formally, $\mathrm{depth}(p) = \max\{|\alpha| \mid \alpha \in \mathtt{CT}(p)\}$. Similarly, the *norm* of a process p, denoted by $\mathrm{norm}(p)$, is the length of a *shortest* completed trace of p, i.e. $\mathrm{norm}(p) = \min\{|\alpha| \mid \alpha \in \mathtt{CT}(p)\}$.

Behavioural equivalences. *Behavioural equivalences* have been introduced to establish whether the behaviours of two processes are *indistinguishable for their observers*. Roughly, they allow us to check whether the *observable* semantics of two processes is *the same*. In the literature we can find several notions of behavioural equivalence based on the observations that an external observer can make on the process. In his seminal article [16], van Glabbeek gave a taxonomy of the behavioural equivalences discussed in the literature on concurrency theory, which is now called the *linear time-branching time spectrum* (see Figure 1).

One of the main concerns in the development of a meta-theory of process languages is to guarantee their *compositionality*, i.e., that the *replacement* of a component of a system with an $\mathcal{R}$-equivalent one, for a chosen behavioural equivalence $\mathcal{R}$, does not affect the behaviour of that system. In algebraic terms, this is known as the *congruence property* of $\mathcal{R}$ with respect to all language operators, which consists in verifying whether

$$f(t_1, \ldots, t_n)\, \mathcal{R}\, f(t'_1, \ldots, t'_n) \text{ for any } n\text{-ary operator } f \text{ whenever } t_i\, \mathcal{R}\, t'_i \text{ for all } i = 1, \ldots, n \ .$$

Since $\mathrm{BCCSP}_{\|}$ operators are defined by inference rules in the de Simone format [12], by [14, Theorem 4] we have that all the equivalences in the spectrum in Figure 1 are congruences with respect to them. Our aim in this paper is to investigate the existence of a finite equational axiomatisation of $\mathrm{BCCSP}_{\|}$ modulo all those congruences.

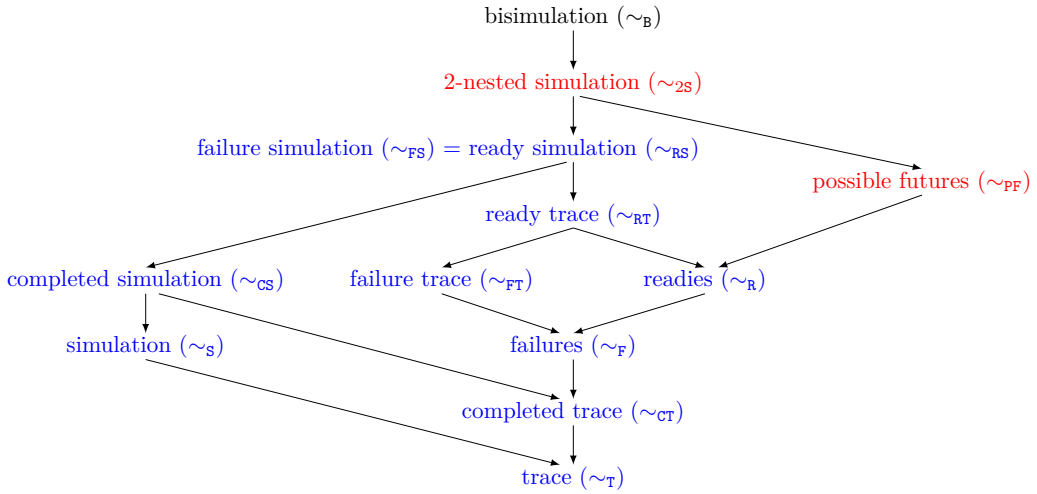


Figure 1 The linear time-branching time spectrum [16]. For the equivalence relations in blue we provide a finite, ground-complete axiomatization. For the ones in red, we provide a negative result. The case of bisimulation is known from the literature.

Table 2 The rules of equational logic.

$$(e_1)\ t \approx t \qquad (e_2)\ \frac{t \approx u}{u \approx t} \qquad (e_3)\ \frac{t \approx u \quad u \approx v}{t \approx v} \qquad (e_4)\ \frac{t \approx u}{\sigma(t) \approx \sigma(u)}$$

$$(e_5)\ \frac{t \approx u}{a.t \approx a.u} \qquad (e_6)\ \frac{t \approx u \quad t' \approx u'}{t + t' \approx u + u'} \qquad (e_8)\ \frac{t \approx u \quad t' \approx u'}{t \parallel t' \approx u \parallel u'}\ .$$

Equational Logic. An *axiom system* $\mathcal{E}$ is a collection of *equations* $t \approx u$ over $\mathrm{BCCSP}_{\parallel}$. An equation $t \approx u$ is *derivable* from an axiom system $\mathcal{E}$, notation $\mathcal{E} \vdash t \approx u$, if there is an *equational proof* for it from $\mathcal{E}$, namely if $t \approx u$ can be inferred from the axioms in $\mathcal{E}$ using the *rules* of *equational logic*, which express reflexivity, symmetry, transitivity, substitution and closure under $\mathrm{BCCSP}_{\parallel}$ contexts and are reported in Table 2.

We are interested in equations that are valid modulo some congruence relation $\mathcal{R}$ over closed terms. The equation $t \approx u$ is said to be *sound* modulo $\mathcal{R}$ if $\sigma(t)\ \mathcal{R}\ \sigma(u)$ for all closed substitutions σ. For simplicity, if $t \approx u$ is sound modulo $\mathcal{R}$, then we write $t\ \mathcal{R}\ u$. An axiom system is *sound* modulo $\mathcal{R}$ if, and only if, all of its equations are sound modulo $\mathcal{R}$. Conversely, we say that $\mathcal{E}$ is *ground-complete* modulo $\mathcal{R}$ if $p\ \mathcal{R}\ q$ implies $\mathcal{E} \vdash p \approx q$ for all closed terms p, q. We say that $\mathcal{R}$ has a *finite* ground-complete axiomatisation, if there is a *finite* axiom system $\mathcal{E}$ that is sound and ground-complete for $\mathcal{R}$.

In Table 3 we present some basic axioms for $\mathrm{BCCSP}_{\parallel}$ that are sound with respect to all the behavioural equivalences in Figure 1. Henceforth, we will let $\mathcal{E}_0 = \{\mathrm{A0}, \mathrm{A1}, \mathrm{A2}, \mathrm{A3}\}$, and we will denote by $\mathcal{E}_1$ the axiom system consisting of all the axioms in Table 3, namely $\mathcal{E}_1 = \mathcal{E}_0 \cup \{\mathrm{P0}, \mathrm{P1}\}$.

To be able to eliminate the interleaving parallel composition operator from closed terms we will make use of two refinements EL1 and EL2 of EL3, which is the classic expansion law [18] (see Table 4). We remark that the actions occurring in the three axioms in Table 4 are not action variables. Hence, when we write that an axiom system $\mathcal{E}$ includes one of these axioms, we mean that it includes all possible instances of that axiom with respect to the

Table 3 Basic axioms for $\mathrm{BCCSP}_{\parallel}$. We define $\mathcal{E}_0 = \{\mathrm{A0}, \mathrm{A1}, \mathrm{A2}, \mathrm{A3}\}$ and $\mathcal{E}_1 = \mathcal{E}_0 \cup \{\mathrm{P0}, \mathrm{P1}\}$.

(A0)	$x + \mathbf{0} \approx x$	(P0)	$x \parallel \mathbf{0} \approx x$
(A1)	$x + y \approx y + x$	(P1)	$x \parallel y \approx y \parallel x$
(A2)	$(x + y) + z \approx x + (y + z)$		
(A3)	$x + x \approx x$		

Table 4 The different instantiations of the expansion law.

(EL1)	$ax \parallel by \approx a(x \parallel by) + b(ax \parallel y)$
(EL2)	$\sum_{i\in I} a_i x_i \parallel \sum_{j\in J} b_j y_j \approx \sum_{i\in I} a_i(x_i \parallel \sum_{j\in J} b_j y_j) + \sum_{j\in J} b_j(\sum_{i\in I} a_i x_i \parallel y_j)$ with $a_i \neq a_k$ whenever $i \neq k$ and $b_j \neq b_h$ whenever $j \neq h$, $\forall\, i, k \in I, \forall\, j, h \in J$
(EL3)	$\sum_{i\in I} a_i x_i \parallel \sum_{j\in J} b_j y_j \approx \sum_{i\in I} a_i(x_i \parallel \sum_{j\in J} b_j y_j) + \sum_{j\in J} b_j(\sum_{i\in I} a_i x_i \parallel y_j)$

actions in $\mathcal{A}$. In particular, EL3 is a schema that generates infinitely many axioms, regardless of the cardinality of the set of actions. This is due to the fact that we can have arbitrary summations in the two arguments of the parallel composition in the left hand side of EL3. Conversely, when the set of actions is assumed to be finite, we are guaranteed that there are only finitely many instances of EL1 and EL2. Indeed, EL1 is a particular instance of EL2, i.e., the one in which both summations are over singletons. The reason for considering both is that, as we will see, EL1 is enough to obtain the elimination result when combined with axioms allowing us to reduce any process of the form $(\sum_{i\in I} a_i p_i) \parallel (\sum_{j\in J} b_j q_j)$ to $\sum_{i\in I, j\in J}(a_i p_i \parallel b_j q_j)$. Conversely, EL2 is needed when this reduction is not sound modulo the considered semantics.

3 The first stage: ready simulation

In this section we study the equational theory of *ready simulation*, whose formal definition is recalled below together with those of *completed simulation* and *simulation* equivalence.

▶ **Definition 1** (Simulation equivalences).

- *A* simulation *is a binary relation $\mathcal{R} \subseteq \mathcal{P} \times \mathcal{P}$ such that, whenever $p\,\mathcal{R}\,q$ and $p \xrightarrow{a} p'$, then there is some q' such that $q \xrightarrow{a} q'$ and $p'\,\mathcal{R}\,q'$. We write $p \sqsubseteq_{\mathtt{S}} q$ if there is a simulation $\mathcal{R}$ such that $p\,\mathcal{R}\,q$. We say that p is* simulation equivalent *to q, notation $p \sim_{\mathtt{S}} q$, if $p \sqsubseteq_{\mathtt{S}} q$ and $q \sqsubseteq_{\mathtt{S}} p$.*
- *A* completed simulation *is a simulation $\mathcal{R}$ such that, whenever $p\,\mathcal{R}\,q$ and $\mathtt{I}(p) = \emptyset$, then $\mathtt{I}(q) = \emptyset$. We write $p \sqsubseteq_{\mathtt{CS}} q$ if there is a completed simulation $\mathcal{R}$ such that $p\,\mathcal{R}\,q$. We say that p is* completed simulation equivalent *to q, notation $p \sim_{\mathtt{CS}} q$, if $p \sqsubseteq_{\mathtt{CS}} q$ and $q \sqsubseteq_{\mathtt{CS}} p$.*
- *A* ready simulation *is a simulation $\mathcal{R}$ such that, whenever $p\,\mathcal{R}\,q$ then $\mathtt{I}(p) = \mathtt{I}(q)$. We write $p \sqsubseteq_{\mathtt{RS}} q$ if there is a ready simulation $\mathcal{R}$ such that $p\,\mathcal{R}\,q$. We say that p is* ready simulation equivalent *to q, notation $p \sim_{\mathtt{RS}} q$, if $p \sqsubseteq_{\mathtt{RS}} q$ and $q \sqsubseteq_{\mathtt{RS}} p$.*

In [15] the notion of *failure simulation* was also introduced as a simulation $\mathcal{R}$ such that, whenever $p\,\mathcal{R}\,q$ and $\mathtt{I}(p) \cap X = \emptyset$, for some $X \subseteq \mathcal{A}$, then $\mathtt{I}(q) \cap X = \emptyset$. Then, in [14] it was proved that the notion of failure simulation coincides with that of ready simulation.

Our aim is to provide a *finite*, *ground-complete* axiomatisation of $\mathrm{BCCSP}_{\parallel}$ modulo ready simulation equivalence. To this end, we recall that in [16] it was proved that the axiom system consisting of $\mathcal{E}_0$ together with axiom RS in Table 5 is a ground-complete axiomatisation of BCCSP, namely $\mathrm{BCCSP}_{\parallel}$ without any occurrence of $\parallel$, modulo ready simulation equivalence.

Table 5 Additional axioms for (ready, completed) simulation equivalence.

(RS)	$a(bx+by+z) \approx a(bx+by+z)+a(bx+z)$
(RSP1)	$(ax+ay+u) \parallel (bz+bw+v) \approx (ax+u) \parallel (bz+bw+v)+(ay+u) \parallel (bz+bw+v)+$ $+(ax+ay+u) \parallel (bz+v)+(ax+ay+u) \parallel (bw+v)$
(RSP2)	$\left(\sum_{i\in I} a_i x_i\right) \parallel (by+bz+w) \approx \sum_{i\in I} a_i (x_i \parallel (by+bz+w))+$ $+\left(\sum_{i\in I} a_i x_i\right) \parallel (by+w)+\left(\sum_{i\in I} a_i x_i\right) \parallel (bz+w)$ where $a_j \neq a_k$ whenever $j \neq k$ for $j,k \in I$
$\mathcal{E}_{\mathtt{RS}} = \mathcal{E}_1 \cup \{\text{RS,RSP1,RSP2,EL2}\}$	
(CS)	$a(bx+y+z) \approx a(bx+y+z)+a(bx+z)$
(CSP1)	$(ax+by+u) \parallel (cz+dw+v) \approx (ax+u) \parallel (cz+dw+v)+(by+u) \parallel (cz+dw+v)+$ $+(ax+by+u) \parallel (cz+v)+(ax+by+u) \parallel (dw+v)$
(CSP2)	$ax \parallel (by+cz+w) \approx a(x \parallel (by+cz+w))+ax \parallel (by+w)+ax \parallel (cz+w)$
$\mathcal{E}_{\mathtt{CS}} = \mathcal{E}_1 \cup \{\text{CS,CSP1,CSP2,EL1}\}$	
(S)	$a(x+y) \approx a(x+y)+ax$
(SP1)	$(x+y) \parallel (z+w) \approx x \parallel (z+w)+y \parallel (z+w)+(x+y) \parallel z+(x+y) \parallel w$
(SP2)	$ax \parallel (y+z) \approx a(x \parallel (y+z))+ax \parallel y+ax \parallel z$
$\mathcal{E}_{\mathtt{S}} = \mathcal{E}_1 \cup \{\text{S,SP1,SP2,EL1}\}$	

Hence, to obtain a finite, ground-complete axiomatisation of $\mathrm{BCCSP}_{\parallel}$ modulo $\sim_{\mathtt{RS}}$ it suffices to enrich the axiom system $\mathcal{E}_1 \cup \{\mathrm{RS}\}$ with finitely many axioms allowing one to eliminate all occurrences of $\parallel$ from closed $\mathrm{BCCSP}_{\parallel}$ terms. In fact, by letting $\mathcal{E}_{\mathtt{RS}}$ denote the axiom system $\mathcal{E}_1 \cup \{\mathrm{RS}\}$ enriched with the necessary axioms, the elimination result consists in proving that for every closed $\mathrm{BCCSP}_{\parallel}$ term p there is a closed BCCSP term q (i.e., without any occurrence of $\parallel$ in it) such that $\mathcal{E}_{\mathtt{RS}} \vdash p \approx q$. Therefore, the completeness of the proposed axiom system over $\mathrm{BCCSP}_{\parallel}$ is a direct consequence of that over BCCSP proved in [16].

Clearly, EL3 would allow us to obtain the desired elimination, but, as previously outlined, it is a schema that finitely presents as infinite collection of equations, and thus an axiom system including it is not finite. In order to obtain the elimination result using only finitely many axioms we will characterise the distributivity properties of $\parallel$ over $+$ modulo ready simulation equivalence. This is done by axioms RSP1 and RSP2 in Table 5.

First of all, we notice that the axiom system $\mathcal{E}_{\mathtt{RS}} = \mathcal{E}_1 \cup \{\mathrm{RS}, \mathrm{RSP1}, \mathrm{RSP2}, \mathrm{EL2}\}$ is sound modulo ready simulation equivalence.

▶ **Theorem 2** ($\mathcal{E}_{\mathtt{RS}}$ soundness). *The axiom system $\mathcal{E}_{\mathtt{RS}}$ is sound for $\mathrm{BCCSP}_{\parallel}$ modulo ready simulation equivalence, namely whenever $\mathcal{E}_{\mathtt{RS}} \vdash p \approx q$ then $p \sim_{\mathtt{RS}} q$.*

Let us focus now on ground-completeness. Intuitively, RSP1 and RSP2 have been constructed in such a way that the set of initial actions of the two arguments of $\parallel$ is preserved, while the initial term is reduced to a sum of terms of smaller size. Briefly, according to the main features of ready simulation semantics, axiom RSP1 allows us to distribute $\parallel$ over $+$ when both arguments of $\parallel$ have nondeterministic choices among summands having the same initial action. Conversely, axiom RSP2 deals with the case in which only one argument of $\parallel$ has summands with the same initial action. In order to preserve the branching structure of the process, which is fundamental to guarantee the soundness of the axioms modulo $\sim_{\mathtt{RS}}$, both RSP1 and RSP2 take into account the behaviour of both arguments

of $\|$: the terms in the right-hand side of both axioms are such that whenever the initial nondeterministic choice of one argument of $\|$ is resolved, the entire behaviour of the other argument is preserved. In fact, we stress that a simplified version of, e.g., RSP1 in which only one argument of $\|$ distributes over $+$ would not be sound modulo $\sim_{\mathtt{RS}}$. Consider, for instance, the process $p = (ap_1 + ap_2 + b) \| c$, with $p_1 \not\sim_{\mathtt{RS}} p_2$. It is immediate to verify that $p \not\sim_{\mathtt{RS}} (ap_1 + b) \| c + (ap_2 + b) \| c$.

The idea is that by (repeatedly) applying axioms RSP1 and RSP2, from left to right, we are able to reduce a process of the form $(\sum_{i\in I} p_i) \| (\sum_{j\in J} p_j)$ to a process of the form $\sum_{k\in K} p_k$ such that whenever p_k has $\|$ as head operator then $p_k = \sum_{h\in H} a_h p_h \| \sum_{l\in L} b_l p_l$, with $a_h \neq a_{h'}$ for $h \neq h'$, and $b_l \neq b_{l'}$ for $l \neq l'$, for some closed $\mathrm{BCCSP}_{\|}$ terms p_h, p_l. The elimination of $\|$ from these terms can then proceed by means of the finitary refinement EL2 of the expansion law presented in Table 4. In particular, we notice that RSP2 is needed because RSP1 alone does not allow us to reduce all processes of the form $(\sum_{i\in I} p_i) \| (\sum_{j\in J} p_j)$ into a sum of processes to which EL2 can be applied. This is mainly due to the fact that, in order to be sound modulo $\sim_{\mathtt{RS}}$, RSP1 imposes constraints on the form of both arguments of a process $(\sum_{i\in I} p_i) \| (\sum_{j\in J} p_j)$.

We can then proceed to prove the elimination result.

▶ **Proposition 3** ($\mathcal{E}_{\mathtt{RS}}$ elimination). *For every closed* $\mathrm{BCCSP}_{\|}$ *term* p *there exists a* BCCSP *term* q *such that* $\mathcal{E}_{\mathtt{RS}} \vdash p \approx q$.

The ground-completeness of $\mathcal{E}_{\mathtt{RS}}$ then follows from the ground-completeness of $\mathcal{E}_0 \cup \{\mathrm{RS}\}$ over BCCSP [16].

▶ **Theorem 4** ($\mathcal{E}_{\mathtt{RS}}$ completeness). *The axiom system* $\mathcal{E}_{\mathtt{RS}}$ *is a ground-complete axiomatisation of* $\mathrm{BCCSP}_{\|}$ *modulo ready simulation equivalence, i.e., whenever* $p \sim_{\mathtt{RS}} q$ *then* $\mathcal{E}_{\mathtt{RS}} \vdash p \approx q$.

We remark that since axioms RSP1, RSP2, and EL2 are sound modulo ready simulation equivalence, they are automatically sound modulo all the equivalences in the spectrum that are coarser than $\sim_{\mathtt{RS}}$, namely the completed simulation, simulation, and (decorated) trace equivalences. Hence, we can easily obtain finite, ground-complete axiomatisations of $\mathrm{BCCSP}_{\|}$ modulo each of those equivalences by adding RSP1, RSP2 and EL2 to the respective ground-complete axiomatisations of BCCSP that have been proposed in the literature [7, 16]. However, for each of those equivalences we can provide stronger axioms that give a more elegant characterisation of the distributivity properties of $\|$ over $+$. In particular, the axiom schemata RSP2 and EL2 both generate $2^{|\mathcal{A}|}$ equational axioms. By exploiting the various forms of distributivity of parallel composition over choice, we can obtain more concise ground-complete axiomatisations of $\mathrm{BCCSP}_{\|}$ modulo the coarser equivalences. We dedicate the next two sections to the presentation of these results.

4 Completed simulation and simulation

In this section we refine the axiom system $\mathcal{E}_{\mathtt{RS}}$ to obtain finite, ground-complete axiomatisations of $\mathrm{BCCSP}_{\|}$ modulo completed simulation and simulation equivalences. To this end, we replace RSP1 and RSP2 with new axioms, tailored for the considered semantics, that allow us to obtain the elimination of $\|$ from closed $\mathrm{BCCSP}_{\|}$ terms, while imposing less restrictive constraints on the distributivity of $\|$ over $+$.

Let us focus first on completed simulation equivalence. We can use axioms CSP1 and CSP2 in Table 5 to characterise the distributivity of $\|$ over $+$ modulo $\sim_{\mathtt{CS}}$. Intuitively, CSP1 is the *completed simulation counterpart* of RSP1, and CSP2 is that of RSP2. Notice that both

CSP1 and CSP2 are such that when distributing $\|$ over $+$ we never get $\mathbf{0}$ as an argument of $\|$, thus guaranteeing the soundness of the reduction modulo $\sim_{\mathtt{CS}}$. Moreover, we stress that CSP1 and CSP2 are not sound modulo ready simulation equivalence. This is due to the fact that both axioms allow for distributing $\|$ over $+$ regardless of the initial actions of the summands. It is then immediate to check that, for instance, $a \,\|\, (b+c) \not\sim_{\mathtt{RS}} a \,\|\, b + a \,\|\, c + a \,\|\, (b+c)$, whereas $a \,\|\, (b+c) \sim_{\mathtt{CS}} a \,\|\, b + a \,\|\, c + a \,\|\, (b+c)$. Interestingly, due to the relaxed constraints on distributivity, by (repeatedly) applying CSP1 and CSP2, from left to right, we are able to reduce a $\mathrm{BCCSP}_{\|}$ process of the form $(\sum_{i\in I} p_i) \,\|\, (\sum_{j\in J} p_j)$ to a $\mathrm{BCCSP}_{\|}$ process of the form $\sum_{k\in K} p_k$ such that whenever p_k has $\|$ as head operator then $p_k = a_k q_k \,\|\, b_k q'_k$ for some q_k, q'_k. We can then use the refinement EL1 of the expansion law to proceed with the elimination of $\|$ from these terms.

Consider the axiom system $\mathcal{E}_{\mathtt{CS}} = \mathcal{E}_1 \cup \{\mathrm{CS}, \mathrm{CSP1}, \mathrm{CSP2}, \mathrm{EL1}\}$. We can formalise the elimination result for $\sim_{\mathtt{CS}}$ in the following proposition.

▶ **Proposition 5** ($\mathcal{E}_{\mathtt{CS}}$ elimination). *For every closed* $\mathrm{BCCSP}_{\|}$ *term* p *there exists a* BCCSP *term* q *such that* $\mathcal{E}_{\mathtt{CS}} \vdash p \approx q$.

A similar reasoning could be applied to obtain the elimination result for simulation equivalence. Although this result could be directly derived by the soundness of CSP1 and CSP2 modulo simulation equivalence, we can provide stronger axioms for the distributivity of $\|$ over summation modulo $\sim_{\mathtt{S}}$. Hence, we replace CSP1 and CSP2 by axioms SP1 and SP2 in Table 5 and we combine them with EL1 to eliminate all occurrences of $\|$ from the closed $\mathrm{BCCSP}_{\|}$ terms. However, it is also possible to obtain the elimination result for simulation equivalence as a corollary of that for completed simulation. Consider the axiom system $\mathcal{E}_{\mathtt{S}} = \mathcal{E}_1 \cup \{\mathrm{S}, \mathrm{SP1}, \mathrm{SP2}, \mathrm{EL1}\}$. We can show that the axioms in $\mathcal{E}_{\mathtt{CS}}$ are all provable from the axiom system $\mathcal{E}_{\mathtt{S}}$.

▶ **Lemma 6.** *The axioms of the system* $\mathcal{E}_{\mathtt{CS}}$ *are derivable from the axiom system* $\mathcal{E}_{\mathtt{S}}$*, namely:*

1. $\mathcal{E}_{\mathtt{S}} \vdash \mathrm{CS}$,
2. $\mathcal{E}_{\mathtt{S}} \vdash \mathrm{CSP1}$, *and*
3. $\mathcal{E}_{\mathtt{S}} \vdash \mathrm{CSP2}$.

▶ **Proposition 7** ($\mathcal{E}_{\mathtt{S}}$ elimination). *For every closed* $\mathrm{BCCSP}_{\|}$ *term* p *there exists a closed* BCCSP *term* q *such that* $\mathcal{E}_{\mathtt{S}} \vdash p \approx q$.

▶ Remark 8. A natural question that may arise is whether a similar derivation is possible for $\mathcal{E}_{\mathtt{RS}}$ from $\mathcal{E}_{\mathtt{CS}}$. We conjecture that the answer is negative. In particular, axiom RSP2 cannot be derived from the axioms in $\mathcal{E}_{\mathtt{CS}}$.

In light of the results above, and those in [16] showing that $\mathcal{E}_0 \cup \{\mathrm{CS}\}$ and $\mathcal{E}_0 \cup \{\mathrm{S}\}$ are sound and ground-complete axiomatisations of BCCSP modulo $\sim_{\mathtt{CS}}$ and $\sim_{\mathtt{S}}$, respectively, we can infer that $\mathcal{E}_{\mathtt{CS}}$ and $\mathcal{E}_{\mathtt{S}}$ are ground-complete axiomatisations of $\mathrm{BCCSP}_{\|}$ modulo completed simulation equivalence and simulation equivalence, respectively.

▶ **Theorem 9** (Soundness and completeness of $\mathcal{E}_{\mathtt{CS}}$ and $\mathcal{E}_{\mathtt{S}}$). *Let* $\mathtt{X} \in \{\mathtt{CS}, \mathtt{S}\}$. *The axiom system* $\mathcal{E}_{\mathtt{X}}$ *is a sound, ground-complete axiomatisation of* $\mathrm{BCCSP}_{\|}$ *modulo* $\sim_{\mathtt{X}}$*, i.e.,* $p \sim_{\mathtt{X}} q$ *if and only if* $\mathcal{E}_{\mathtt{X}} \vdash p \approx q$.

Table 6 Additional axioms for trace and decorated trace equivalences.

(RT)	$a\left(\sum_{i=1}^{\lvert\mathcal{A}\rvert}(b_ix_i+b_iy_i)+z\right) \approx a\left(\sum_{i=1}^{\lvert\mathcal{A}\rvert} b_ix_i+z\right)+a\left(\sum_{i=1}^{\lvert\mathcal{A}\rvert} b_iy_i+z\right)$
(FP)	$(ax+ay+w) \parallel z \approx (ax+w) \parallel z + (ay+w) \parallel z$
	$\mathcal{E}_{\mathtt{RT}} = \mathcal{E}_1 \cup \{\text{RT,FP,EL2}\}$
(FT)	$ax+ay \approx ax+ay+a(x+y)$
	$\mathcal{E}_{\mathtt{FT}} = \mathcal{E}_1 \cup \{\text{FT,RS,FP,EL2}\}$
(R)	$a(bx+z)+a(by+w) \approx a(bx+by+z)+a(by+w)$
	$\mathcal{E}_{\mathtt{R}} = \mathcal{E}_1 \cup \{\text{R,FP,EL2}\}$
(F)	$ax+a(y+z) \approx ax+a(x+y)+a(y+z)$
	$\mathcal{E}_{\mathtt{F}} = \mathcal{E}_1 \cup \{\text{F,R,FP,EL2}\}$
(CT)	$a(bx+z)+a(cy+w) \approx a(bx+cy+z+w)$
(CTP)	$(ax+by+w) \parallel z \approx (ax+w) \parallel z + (by+w) \parallel z$
	$\mathcal{E}_{\mathtt{CT}} = \mathcal{E}_1 \cup \{\text{CT,CTP,EL1}\}$
(T)	$ax+ay \approx a(x+y)$
(TP)	$(x+y) \parallel z \approx x \parallel z + y \parallel z$
	$\mathcal{E}_{\mathtt{T}} = \mathcal{E}_1 \cup \{\text{T,TP,EL1}\}$

5 Linear semantics: from ready traces to traces

We continue our journey in the spectrum by moving to the linear-time semantics. In this section we consider trace semantics and all of its decorated versions, and we provide a finite, ground-complete axiomatisation for each of them (see Table 6).

From a technical point of view, we can split the results of this section into two parts:

1. those for ready trace, failure trace, ready, and failures equivalence, and
2. those for completed trace, and trace equivalence.

In both parts we prove the elimination result only for the finest semantics, namely ready trace (Proposition 11) and completed trace (Proposition 17) respectively. We then obtain the remaining elimination results by showing that all the axioms in $\mathcal{E}_{\mathtt{X}}$ are provable from $\mathcal{E}_{\mathtt{Y}}$, where $\mathtt{X}$ is finer than $\mathtt{Y}$ in the considered part.

5.1 From ready traces to failures

First we deal with the decorated trace semantics based on the comparison of the failure and ready sets of processes.

▶ **Definition 10** (Readiness and failures equivalences).

- *A* failure pair *of a process p is a pair (α, X), with $\alpha \in \mathcal{A}^*$ and $X \subseteq \mathcal{A}$, such that $p \xrightarrow{\alpha} q$ for some process q with $\mathtt{I}(q) \cap X = \emptyset$. We denote by $\mathtt{F}(p)$ the set of failure pairs of p. Two processes p and q are* failures equivalent*, denoted $p \sim_{\mathtt{F}} q$, if $\mathtt{F}(p) = \mathtt{F}(q)$.*
- *A* ready pair *of a process p is a pair (α, X), with $\alpha \in \mathcal{A}^*$ and $X \subseteq \mathcal{A}$, such that $p \xrightarrow{\alpha} q$ for some process q with $\mathtt{I}(q) = X$. We let $\mathtt{R}(p)$ denote the set of ready pairs of p. Two processes p and q are* ready equivalent*, written $p \sim_{\mathtt{R}} q$, if $\mathtt{R}(p) = \mathtt{R}(q)$.*

- *A* failure trace *of a process p is a sequence $X_0a_1X_1 \dots a_nX_n$, with $X_i \subseteq \mathcal{A}$ and $a_i \in \mathcal{A}$, such that there are $p_1, \dots, p_n \in \mathcal{P}$ with $p = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p_n$ and $\mathtt{I}(p_i) \cap X_i = \emptyset$ for all $0 \leq i \leq n$. We write $\mathtt{FT}(p)$ for the set of failure traces of p. Two processes p and q are* failure trace equivalent*, denoted $p \sim_{\mathtt{FT}} q$, if $\mathtt{FT}(p) = \mathtt{FT}(q)$.*
- *A* ready trace *of a process p is a sequence $X_0a_1X_1 \dots a_nX_n$, for $X_i \subseteq \mathcal{A}$ and $a_i \in \mathcal{A}$, such that there are $p_1, \dots p_n \in \mathcal{P}$ with $p = p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} p_n$ and $\mathtt{I}(p_i) = X_i$ for all $0 \leq i \leq n$. We write $\mathtt{RT}(p)$ for the set of ready traces of p. Two processes p and q are* ready trace equivalent*, denoted $p \sim_{\mathtt{RT}} q$, if $\mathtt{RT}(p) = \mathtt{RT}(q)$.*

We consider first the finest equivalence among those in Definition 10, namely ready trace equivalence. This can be considered as the linear counterpart of ready simulation: we focus on the current execution of the process and we require that each step is mimicked by reaching processes having the same sets of initial actions. Interestingly, we can find a similar correlation between the axioms characterising the distributivity of $\|$ over $+$ modulo the two semantics. Consider axiom FP in Table 6. We can see this axiom as the *linear* counterpart of RSP1: since in the linear semantics we are interested only in the current execution of a process, we can characterise the distributivity of $\|$ over $+$ by treating the two arguments of $\|$ independently from one another. To obtain the elimination result for $\sim_{\mathtt{RT}}$ we do not need to introduce the linear counterpart of axiom RSP2. In fact, FP imposes constraints on the form of only one argument of $\|$. Hence, it is possible to use it to reduce any process of the form $(\sum_{i\in I} p_i) \,\|\, (\sum_{j\in J} p_j)$ into a sum of processes to which EL2 can be applied. We can in fact prove that the axioms in the system $\mathcal{E}_{\mathtt{RT}} = \mathcal{E}_1 \cup \{\mathrm{RT}, \mathrm{FP}, \mathrm{EL2}\}$ are sufficient to eliminate all occurrences of $\|$ from closed $\mathrm{BCCSP}_{\|}$ terms.

▶ **Proposition 11** ($\mathcal{E}_{\mathtt{RT}}$ elimination). *For every closed* $\mathrm{BCCSP}_{\|}$ *term p there is a closed* BCCSP *term q such that $\mathcal{E}_{\mathtt{RT}} \vdash p \approx q$.*

▶ Remark 12. Similarly to the case of completed simulation (cf. Remark 8), the reason why we propose to prove directly the elimination result for ready trace equivalence is that we did not manage to derive the axioms in $\mathcal{E}_{\mathtt{RS}}$ from those in $\mathcal{E}_{\mathtt{RT}}$. Once again, the main issue is that axiom RSP2 cannot be derived from those in $\mathcal{E}_{\mathtt{RT}}$, even though all its closed instantiations can. We leave a formal analysis of this issue as future work.

Interestingly, axiom FP also characterises the distributivity of $\|$ over $+$ modulo $\sim_{\mathtt{FT}}, \sim_{\mathtt{R}}$ and $\sim_{\mathtt{F}}$, in the sense that the constraints that it imposes on the form of the arguments of $\|$ to trigger the reduction cannot be relaxed when considering the above-mentioned coarser semantics. Consider the axiom systems $\mathcal{E}_{\mathtt{FT}} = \mathcal{E}_1 \cup \{\mathrm{FT,RS,FP,EL2}\}$, $\mathcal{E}_{\mathtt{R}} = \mathcal{E}_1 \cup \{\mathrm{R,FP,EL2}\}$ and $\mathcal{E}_{\mathtt{F}} = \mathcal{E}_1 \cup \{\mathrm{F,R,FP,EL2}\}$. The following derivability relations among them and $\mathcal{E}_{\mathtt{RT}}$ are then easy to check.

▶ **Lemma 13.**

1. *The axioms in the system $\mathcal{E}_{\mathtt{RT}}$ are derivable from $\mathcal{E}_{\mathtt{FT}}$, namely $\mathcal{E}_{\mathtt{FT}} \vdash \mathrm{RT}$.*
2. *The axioms in the system $\mathcal{E}_{\mathtt{RT}}$ are derivable from $\mathcal{E}_{\mathtt{R}}$, namely $\mathcal{E}_{\mathtt{R}} \vdash \mathrm{RT}$.*
3. *The axioms in the system $\mathcal{E}_{\mathtt{FT}}$ are derivable from $\mathcal{E}_{\mathtt{F}}$, namely,*
   a. *$\mathcal{E}_{\mathtt{F}} \vdash \mathrm{FT}$, and*
   b. *$\mathcal{E}_{\mathtt{F}} \vdash \mathrm{RS}$.*

   Moreover, also the axioms in the system $E_{\mathtt{R}}$ are derivable from $\mathcal{E}_{\mathtt{F}}$.

The next proposition is then a corollary of Proposition 11 and Lemma 13.

▶ **Proposition 14** ($\mathcal{E}_{\mathtt{FT}}$, $\mathcal{E}_{\mathtt{R}}$, $\mathcal{E}_{\mathtt{F}}$ elimination). *Let $\mathtt{X} \in \{\mathtt{FT}, \mathtt{R}, \mathtt{F}\}$. For every* $\mathrm{BCCSP}_{\|}$ *term p there is a closed* BCCSP *term q such that $\mathcal{E}_{\mathtt{X}} \vdash p \approx q$.*

In [7] it was proved that, under the assumption that $\mathcal{A}$ is finite, the axiom system $\mathcal{E}_0 \cup \{\mathrm{RT}\}$ is a ground-complete axiomatisation of BCCSP modulo $\sim_{\mathtt{RT}}$. Moreover, it was also proved that $\mathcal{E}_0 \cup \{\mathrm{FT,RS}\}$ is a ground-complete axiomatisation of BCCSP modulo $\sim_{\mathtt{FT}}$. The ground-completeness of $\mathcal{E}_0 \cup \{\mathrm{R}\}$, modulo $\sim_{\mathtt{R}}$, and that of $\mathcal{E}_0 \cup \{\mathrm{F}, \mathrm{R}\}$, modulo $\sim_{\mathtt{F}}$, over BCCSP were proved in [16]. Consequently, the soundness and ground-completeness of the proposed axioms systems can then be derived from the elimination results above and the completeness results given in [7, 16].

▶ **Theorem 15** (Soundness and completeness of $\mathcal{E}_{\mathtt{RT}}$, $\mathcal{E}_{\mathtt{FT}}$, $\mathcal{E}_{\mathtt{R}}$ and $\mathcal{E}_{\mathtt{F}}$)**.** *Let* $\mathtt{X} \in \{\mathtt{RT}, \mathtt{FT}, \mathtt{R}, \mathtt{F}\}$. *The axiom system* $\mathcal{E}_{\mathtt{X}}$ *is a sound, ground-complete axiomatisation of* $\mathrm{BCCSP}_{\parallel}$ *modulo* $\sim_{\mathtt{X}}$*, i.e.,* $p \sim_{\mathtt{X}} q$ *if and only if* $\mathcal{E}_{\mathtt{X}} \vdash p \approx q$*.*

5.2 Completed traces and traces

It remains to consider completed trace equivalence and trace equivalence.

▶ **Definition 16** (Trace and completed trace equivalences)**.** *Two processes* p *and* q *are* trace equivalent*, denoted* $p \sim_{\mathtt{T}} q$*, if* $\mathtt{T}(p) = \mathtt{T}(q)$*. If, in addition, it holds that* $\mathtt{CT}(p) = \mathtt{CT}(q)$*, then* p *and* q *are* completed trace equivalent*, denoted* $p \sim_{\mathtt{CT}} q$*.*

Consider the axiom systems $\mathcal{E}_{\mathtt{CT}} = \mathcal{E}_1 \cup \{\mathrm{CT,CTP,EL1}\}$ and $\mathcal{E}_{\mathtt{T}} = \mathcal{E}_1 \cup \{\mathrm{T,TP,EL1}\}$, presented in Table 6. In the same way that axiom FP is the linear counterpart of RSP1 and RSP2, we have that CTP is the linear counterpart of CSP1 and CSP2, and TP is that of SP1 and SP2. It is then easy to check that we can use the axioms in $\mathcal{E}_{\mathtt{CT}}$ to obtain the elimination result for $\sim_{\mathtt{CT}}$.

▶ **Proposition 17** ($\mathcal{E}_{\mathtt{CT}}$ elimination)**.** *For every closed* $\mathrm{BCCSP}_{\parallel}$ *term* p *there is a closed BCCSP term* q *such that* $\mathcal{E}_{\mathtt{CT}} \vdash p \approx q$*.*

Moreover, the elimination for $\sim_{\mathtt{T}}$ follows from the fact that the axioms in $\mathcal{E}_{\mathtt{CT}}$ are derivable from those in $\mathcal{E}_{\mathtt{T}}$.

▶ **Lemma 18.** *The axioms in the system* $\mathcal{E}_{\mathtt{CT}}$ *are derivable from* $\mathcal{E}_{\mathtt{T}}$*, namely,*

1. $\mathcal{E}_{\mathtt{T}} \vdash \mathrm{CT}$*, and*
2. $\mathcal{E}_{\mathtt{T}} \vdash \mathrm{CTP}$*.*

▶ **Proposition 19** ($\mathcal{E}_{\mathtt{T}}$ elimination)**.** *For every closed* $\mathrm{BCCSP}_{\parallel}$ *term* p *there exists a closed* BCCSP *term* q *such that* $\mathcal{E}_{\mathtt{T}} \vdash p \approx q$*.*

▶ Remark 20. The precise relationship between $\mathcal{E}_{\mathtt{CT}}$ on the one hand, and $\mathcal{E}_{\mathtt{RT}}$ and $\mathcal{E}_{\mathtt{CS}}$ on the other hand still needs to be investigated further. We conjecture that the axioms of $\mathcal{E}_{\mathtt{RT}}$ are derivable from $\mathcal{E}_{\mathtt{CT}}$ and that those of $\mathcal{E}_{\mathtt{CS}}$ are not.

In light of Proposition 17, the ground-completeness of $\mathcal{E}_{\mathtt{CT}}$ over $\mathrm{BCCSP}_{\parallel}$ modulo $\sim_{\mathtt{CT}}$ follows from that of $\mathcal{E}_0 \cup \{\mathrm{CT}\}$ over BCCSP provided in [16]. Similarly, the ground-completeness of $\mathcal{E}_0 \cup \{\mathrm{T}\}$ over BCCSP proved in [16] and Proposition 19 give us the ground-completeness of $\mathcal{E}_{\mathtt{T}}$ over $\mathrm{BCCSP}_{\parallel}$.

▶ **Theorem 21** (Soundness and completeness of $\mathcal{E}_{\mathtt{CT}}$ and $\mathcal{E}_{\mathtt{T}}$)**.** *Let* $\mathtt{X} \in \{\mathtt{CT}, \mathtt{T}\}$*. The axiom system* $\mathcal{E}_{\mathtt{X}}$ *is a ground-complete axiomatisation of* $\mathrm{BCCSP}_{\parallel}$ *modulo* $\sim_{\mathtt{X}}$*, i.e.,* $p \sim_{\mathtt{X}} q$ *if and only if* $\mathcal{E}_{\mathtt{X}} \vdash p \approx q$*.*

6 The negative results

We dedicate this section to the negative results: we prove that all the congruences between possible futures equivalence ($\sim_{\mathtt{PF}}$) and bisimilarity ($\sim_{\mathtt{B}}$) do not admit a finite, ground-complete axiomatisation over $\mathrm{BCCSP}_{\parallel}$. This includes all the nested trace and nested simulation equivalences. In [1] it was shown that, even if the set of actions is a singleton, the nested semantics admit no finite axiomatisation over BCCSP. Indeed, the presence of the additional operator $\parallel$ might, at least in principle, allow us to finitely axiomatise the equations over closed BCCSP terms that are valid modulo the considered equivalences. Hence, we prove these results directly.

In detail, firstly we focus on the negative result for possible futures semantics, corresponding to the 2-nested trace semantics [18]. To obtain it, we apply the general technique used by Moller to prove that interleaving is not finitely axiomatisable modulo bisimilarity [22–24]. Briefly, the main idea is to identify a *witness property*. This is a specific property of $\mathrm{BCCSP}_{\parallel}$ terms, say W_N for $N \geq 0$, that, when N is *large enough*, is an invariant that is preserved by provability from finite, sound axiom systems. Roughly, this means that if $\mathcal{E}$ is a finite set of axioms that are sound modulo possible futures equivalence, the equation $p \approx q$ can be derived from $\mathcal{E}$, and N is larger than the size of all the terms in the equations in $\mathcal{E}$, then either both p and q satisfy W_N, or none of them does. Then, we exhibit an infinite family of valid equations, called the *witness family of equations*, in which W_N is not preserved, namely it is satisfied only by one side of each equation.

Afterwards, we exploit the soundness modulo bisimilarity of the equations in the witness family to lift the negative result for $\sim_{\mathtt{PF}}$ to all congruences between $\sim_{\mathtt{B}}$ and $\sim_{\mathtt{PF}}$.

Differently from the aforementioned negative results over BCCSP, ours are obtained assuming that the set of actions contains at least two distinct elements. In fact, when the action set is a singleton, and *only* in that case, the axiom

$$ax \parallel (ay + az) \approx ax \parallel (ay + a(y + z)) + ax \parallel (az + a(y + z))$$

is sound modulo $\sim_{\mathtt{PF}}$. Due to this axiom we were not able to prove the negative result for $\sim_{\mathtt{PF}}$ in the case that $|\mathcal{A}| = 1$, which we leave as an open problem for future work.

6.1 Possible futures equivalence

According to possible futures equivalence [27] two processes are deemed equivalent if, by performing the same traces, they reach processes that are trace equivalent. For this reason, possible futures equivalence is also known as the 2-*nested trace equivalence* [18].

▶ **Definition 22** (Possible futures equivalence). *A* possible future *of a process p is a pair (α, X) where $\alpha \in \mathcal{A}^*$ and $X \subseteq \mathcal{A}^*$ such that $p \xrightarrow{\alpha} p'$ for some p' with $X = \mathtt{T}(p')$. We write $\mathtt{PF}(p)$ for the set of possible futures of p. Two processes p and q are said to be* possible futures equivalent*, denoted $p \sim_{\mathtt{PF}} q$, if $\mathtt{PF}(p) = \mathtt{PF}(q)$.*

Our order of business is to prove the following result.

▶ **Theorem 23.** *Assume that $|\mathcal{A}| \geq 2$. Possible futures equivalence has no finite, ground-complete, equational axiomatisation over the language* $\mathrm{BCCSP}_{\parallel}$.

In what follows, for actions $a, b \in \mathcal{A}$ and $i \geq 0$, we let $b^0 a$ denote $a.0$ and $b^{i+1}a$ stand for $b(b^i a)$. Consider now the infinite family of equations $\{e_N \mid N \geq 1\}$ given, for $a \neq b$, by:

$$p_N = \sum_{i=1}^{N} b^i a \qquad (N \geq 1)$$

$$e_N : a \parallel p_N \approx ap_N + \sum_{i=1}^{N} b(a \parallel b^{i-1}a) \qquad (N \geq 1) \ .$$

Notice that the equations e_N are sound modulo $\sim_{\mathtt{PF}}$ for all $N \geq 1$.

We also notice that none of the summands in the right-hand side of equation e_N is, alone, possible futures equivalent to $a \parallel p_N$. However, we now proceed to show that, when N is large enough, having a summand possible futures equivalent to $a \parallel p_N$ is an invariant under provability from finite sound axiom systems, and it will thus play the role of witness property for our negative result.

To this end, we introduce first some basic notions and results on $\sim_{\mathtt{PF}}$.

▶ **Definition 24.** *We say that a* $\mathrm{BCCSP}_{\parallel}$ *term t has a* $\mathbf{0}$ *factor if it contains a subterm of the form $t_1 \parallel t_2$, where either t_1 or t_2 is possible futures equivalent to* $\mathbf{0}$.

Next, we characterise closed $\mathrm{BCCSP}_{\parallel}$ terms that are possible futures equivalent to p_N.

▶ **Lemma 25.** *Let q be a* $\mathrm{BCCSP}_{\parallel}$ *term that does not have* $\mathbf{0}$ *summands or factors and such that $\mathtt{CT}(q) = \mathtt{CT}(p_N)$ for some $N \geq 1$. Then q does not contain any occurrence of $\parallel$. Moreover $q \sim_{\mathtt{PF}} p_N$ if and only if $q = \sum_{j \in J} q_j$ for some terms q_j such that none of them has $+$ as head operator and:*

- *for each $i \in \{1, \ldots, N\}$ there is some $j \in J$ such that $b^i a \sim_{\mathtt{PF}} q_j$;*
- *for each $j \in J$ there is some $i \in \{1, \ldots, N\}$ such that $q_j \sim_{\mathtt{PF}} b^i a$.*

In light of Lemma 25, we can also provide a decomposition-like characterisation of closed $\mathrm{BCCSP}_{\parallel}$ terms that are possible futures equivalent to $a \parallel p_N$.

▶ **Proposition 26.** *Assume that p, q are two* $\mathrm{BCCSP}_{\parallel}$ *processes such that $p, q \not\sim_{\mathtt{PF}} \mathbf{0}$, p, q do not have* $\mathbf{0}$ *summands or factors, and $p \parallel q \sim_{\mathtt{PF}} a \parallel p_N$, for some $N > 1$. Then either $p \sim_{\mathtt{PF}} a$ and $q \sim_{\mathtt{PF}} p_N$, or $p \sim_{\mathtt{PF}} p_N$ and $q \sim_{\mathtt{PF}} a$.*

The following lemma characterises the open $\mathrm{BCCSP}_{\parallel}$ terms whose substitution instances can be equivalent in possible futures semantics to terms having at least two summands of p_N $(N > 1)$ as their summands.

▶ **Lemma 27.** *Let t be a* $\mathrm{BCCSP}_{\parallel}$ *term that does not have $+$ as head operator. Let $m > 1$ and σ be a closed substitution such that $\sigma(t)$ has no* $\mathbf{0}$ *summands or factors. If $\sigma(t) \sim_{\mathtt{PF}} \sum_{k=1}^{m} b^{i_k} a$, for some $1 \leq i_1 < \cdots < i_m$, then $t = x$ for some variable x.*

We now have all the ingredients necessary to prove Theorem 23. To streamline our presentation, we split the proof of into two main parts: Proposition 28 deals with the preservation of the witness property under provability from the substitution rule of equational logic. Theorem 29 builds on Proposition 28 and proves the witness property to be an invariant under provability from finite sound axiom systems. The full proofs of these two results are provided in the Appendix.

▶ **Proposition 28.** *Let $t \approx u$ be an equation over* $\mathrm{BCCSP}_{\parallel}$ *that is sound modulo $\sim_{\mathtt{PF}}$. Let σ be a closed substitution with $p = \sigma(t)$ and $q = \sigma(u)$. Suppose that p and q have neither* $\mathbf{0}$ *summands nor* $\mathbf{0}$ *factors, and that $p, q \sim_{\mathtt{PF}} a \parallel p_N$ for some N larger than the sizes of t and u. If p has a summand possible futures equivalent to $a \parallel p_N$, then so does q.*

▶ **Theorem 29.** *Let $\mathcal{E}$ be a finite axiom system over* $\mathrm{BCCSP}_{\parallel}$ *that is sound modulo $\sim_{\mathtt{PF}}$. Let N be larger than the size of each term in the equations in $\mathcal{E}$. Assume that p and q are closed terms that contain no occurrences of* $\mathbf{0}$ *as a summand or factor, and that $p, q \sim_{\mathtt{PF}} a \parallel p_N$. If $\mathcal{E} \vdash p \approx q$ and p has a summand possible futures equivalent to $a \parallel p_N$, then so does q.*

As the left-hand side of equation e_N, i.e., the term $a \parallel p_N$, has a summand possible futures equivalent to $a \parallel p_N$, whilst the right-hand side, i.e., the term $ap_N + \sum_{i=1}^{N} b(a \parallel b^{i-1}a)$, does not, we can conclude that the collection of infinitely many equations e_N $(N \geq 1)$ is the desired witness family. This concludes the proof of Theorem 23.

6.2 Extending the negative result

It is easy to check that the equations e_N $(N \geq 1)$ in the witness family of the negative result for $\sim_{\mathtt{PF}}$ are all sound modulo bisimilarity, i.e., the largest symmetric simulation. Consequently, they are also sound modulo any congruence $\mathcal{R}$ such that $\sim_{\mathtt{B}} \subseteq \mathcal{R} \subseteq \sim_{\mathtt{PF}}$. Hence, the negative result for all these equivalences can be derived from that for $\sim_{\mathtt{PF}}$, by exploiting this fact and that any finite axiom system that is sound modulo $\mathcal{R}$ is also sound modulo $\sim_{\mathtt{PF}}$.

▶ **Theorem 30.** *Assume that $|\mathcal{A}| \geq 2$. Let $\mathcal{R}$ be a congruence such that $\sim_{\mathtt{B}} \subseteq \mathcal{R} \subseteq \sim_{\mathtt{PF}}$. Then $\mathcal{R}$ has no finite, ground-complete, equational axiomatisation over the language* $\mathrm{BCCSP}_{\parallel}$.

Theorem 30 can be applied to establish for $n \geq 2$ that the n-nested trace and simulation semantics have no finite, ground-complete equational axiomatisation over $\mathrm{BCCSP}_{\parallel}$. The n-nested trace equivalences were introduced in [18] as an alternative tool to define bisimilarity. The hierarchy of n-nested simulations, namely simulation relations contained in a (nested) simulation equivalence, was introduced in [17].

▶ **Definition 31** (n-nested semantics). *For $n \geq 0$, the relation $\sim_{\mathtt{T}}^{n}$ over $\mathcal{P}$, called the* n-nested trace equivalence*, is defined inductively as follows:*

- $p \sim_{\mathtt{T}}^{0} q$ *for all* $p, q \in \mathcal{P}$,
- $p \sim_{\mathtt{T}}^{n+1} q$ *if and only if for all traces* $\alpha \in \mathcal{A}^*$*:*
  - *if* $p \xrightarrow{\alpha} p'$ *then there is a* q' *such that* $q \xrightarrow{\alpha} q'$ *and* $p' \sim_{\mathtt{T}}^{n} q'$*, and*
  - *if* $q \xrightarrow{\alpha} q'$ *then there is a* p' *such that* $p \xrightarrow{\alpha} p'$ *and* $p' \sim_{\mathtt{T}}^{n} q'$*.*

For $n \geq 0$, the relation $\sqsubseteq_{\mathtt{S}}^{n}$ over $\mathcal{P}$ is defined inductively as follows:

- $p \sqsubseteq_{\mathtt{S}}^{0} q$ *for all* $p, q \in \mathcal{P}$,
- $p \sqsubseteq_{\mathtt{S}}^{n+1} q$ *if and only if* $p \,\mathcal{R}\, q$ *for some simulation* $\mathcal{R}$*, with* $\mathcal{R}^{-1}$ *included in* $\sqsubseteq_{\mathtt{S}}^{n}$.

n-nested simulation equivalence *is the kernel of* $\sqsubseteq_{\mathtt{S}}^{n}$*, i.e., the equivalence* $\sim_{\mathtt{S}}^{n} = \sqsubseteq_{\mathtt{S}}^{n} \cap (\sqsubseteq_{\mathtt{S}}^{n})^{-1}$.

Notably, $\sim_{\mathtt{T}}^{1}$ corresponds to trace equivalence, $\sim_{\mathtt{T}}^{2}$ is possible futures equivalence, and $\sim_{\mathtt{S}}^{1}$ is simulation equivalence. The following theorem is a corollary of Theorems 23 and 30.

▶ **Theorem 32.** *Assume that $|\mathcal{A}| \geq 2$. Let $n \geq 2$. Then, n-nested trace equivalence and n-nested simulation equivalence admit no finite, ground-complete, equational axiomatisation over the language* $\mathrm{BCCSP}_{\parallel}$.

7 Concluding remarks

We have studied the finite axiomatisability of the language $\mathrm{BCCSP}_{\parallel}$ modulo the behavioural equivalences in the linear time-branching time spectrum. On the one hand we have obtained finite, ground-complete axiomatisations modulo the (decorated) trace and simulation semantics in the spectrum. On the other hand we have proved that for all equivalences that are finer than possible futures equivalence and coarser than bisimilarity a finite ground-complete axiomatisation does not exist.

Since our ground-completeness proof for ready simulation equivalence proceeds via elimination of $\parallel$ from closed terms (Proposition 3), and all behavioural equivalences in the linear time-branching time spectrum that include ready simulation have a finite ground-complete axiomatisation over BCCSP, it immediately follows from the elimination result

that all these behavioural equivalences have a finite ground-complete axiomatisation over $\mathrm{BCCSP}_{\|}$. Exploiting various forms of distributivity of parallel composition over choice, we were able to present more concise and elegant axiomatisations for the coarser behavioural equivalences. We did not succeed to equationally derive the axioms of ready simulation equivalence from the axiomatisations of the coarser equivalences. In fact, we conjecture that this is not possible, and leave it for future research to find a proof.

The parallel composition operator we have considered in this paper implements interleaving without synchronisation between parallel components. It is natural to consider extensions of our result to parallel composition operators with some form synchronisation. We expect that extension with CCS-style synchronisation is straightforward, both for the positive and the negative results. Whether this is also the case for extension with ACP-style or CSP-style synchronisation we leave as a topic for future investigations.

As previously outlined, in [1] it was proved that the nested semantics admit no finite axiomatisation over BCCSP. However, our negative results cannot be reduced to a mere lifting of those in [1], as the presence of the additional operator $\|$ might, at least in principle, allow us to finitely axiomatise the equations over BCCSP processes that are valid modulo the considered nested semantics. Indeed, auxiliary operators can be added to some language in order to obtain a finite axiomatisation of some congruence relation (see, e.g. the classic example given in [5]). Understanding whether it is possible to lift non-finite axiomatisability results among different algebras, and under which constraints this can be done, is an interesting research avenue and we aim to investigate it in future work. A methodology for transferring non-finite-axiomatisability results across languages was presented in [3], where a reduction-based approach was proposed. However, that method has some limitations and thus further studies are needed.

A behavioural equivalence is *finitely based* if it has a finite equational axiomatisation from which all valid equations between open terms are derivable. In [13] and [2] finite bases for bisimilarity with respect to PA and $\mathrm{BCCSP}_{\|}$ extended with the auxiliary operators left merge and communication merge were presented. Furthermore, in [9] an overview was given of which behavioural equivalences in the linear time-branching time spectrum are finitely based with respect to BCCSP. The negative results in Section 6 imply that none of the behavioural equivalences between possible futures equivalence and bisimilarity is finitely based with respect to $\mathrm{BCCSP}_{\|}$. An interesting question is which of the behavioural equivalences including ready simulation semantics is finitely based with respect to $\mathrm{BCCSP}_{\|}$.

In [11] an alternative classification of the equivalences in the spectrum with respect to [16] was proposed. In order to obtain a general, unified, view of process semantics, the spectrum was divided into layers, each corresponding to a different notion of constrained simulation [10]. There are pleasing connections between the different layers and the partition they induce over on the congruences in the spectrum, as given in [11], and the relationships between the axioms for the interleaving operator we have presented in this study.

References

1 Luca Aceto, Wan Fokkink, Rob J. van Glabbeek, and Anna Ingólfsdóttir. Nested semantics over finite trees are equationally hard. *Inf. Comput.*, 191(2):203–232, 2004. doi:10.1016/j.ic.2004.02.001.

2 Luca Aceto, Wan Fokkink, Anna Ingólfsdóttir, and Bas Luttik. A finite equational base for CCS with left merge and communication merge. *ACM Trans. Comput. Log.*, 10(1):6:1–6:26, 2009. doi:10.1145/1459010.1459016.

3 Luca Aceto, Wan Fokkink, Anna Ingólfsdóttir, and Mohammad Reza Mousavi. Lifting non-finite axiomatizability results to extensions of process algebras. *Acta Inf.*, 47(3):147–177, 2010. doi:10.1007/s00236-010-0114-7.

4 Jos C.M. Baeten, T. Basten, and M.A. Reniers. *Process algebra: Equational Theories of Communicating Processes*. Cambridge tracts in theoretical computer science. Cambridge University Press, United Kingdom, 2010. doi:10.1017/CBO9781139195003.

5 Jan A. Bergstra and Jan Willem Klop. Process algebra for synchronous communication. *Information and Control*, 60(1-3):109–137, 1984. doi:10.1016/S0019-9958(84)80025-X.

6 Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors. *Handbook of Process Algebra*. North-Holland / Elsevier, 2001. doi:10.1016/b978-0-444-82830-9.x5017-6.

7 Stefan Blom, Wan Fokkink, and Sumit Nain. On the axiomatizability of ready traces, ready simulation, and failure traces. In *Proceedings of ICALP 2003*, volume 2719 of *Lecture Notes in Computer Science*, pages 109–118, 2003. doi:10.1007/3-540-45061-0_10.

8 Taolue Chen and Wan Fokkink. On the axiomatizability of impossible futures: Preorder versus equivalence. In *Proceedings of LICS 2008*, pages 156–165. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.13.

9 Taolue Chen, Wan Fokkink, Bas Luttik, and Sumit Nain. On finite alphabets and infinite bases. *Inf. Comput.*, 206(5):492–519, 2008. doi:10.1016/j.ic.2007.09.003.

10 David de Frutos-Escrig and Carlos Gregorio-Rodríguez. Universal coinductive characterisations of process semantics. In *IFIP International Conference On Theoretical Computer Science 2008*, volume 273 of *IFIP*, pages 397–412, 2008. doi:10.1007/978-0-387-09680-3_27.

11 David de Frutos-Escrig, Carlos Gregorio-Rodríguez, Miguel Palomino, and David Romero-Hernández. Unifying the linear time-branching time spectrum of process semantics. *Logical Methods in Computer Science*, 9(2), 2013. doi:10.2168/LMCS-9(2:11)2013.

12 Robert de Simone. Higher-level synchronising devices in Meije-SCCS. *Theor. Comput. Sci.*, 37:245–267, 1985. doi:10.1016/0304-3975(85)90093-3.

13 Wan Fokkink and Bas Luttik. An *omega*-complete equational specification of interleaving. In *Proceedings of ICALP 2000*, volume 1853 of *Lecture Notes in Computer Science*, pages 729–743, 2000. doi:10.1007/3-540-45022-X_61.

14 Rob J. van Glabbeek. Full abstraction in structural operational semantics (extended abstract). In *Proceedings of AMAST '93*, Workshops in Computing, pages 75–82, 1993.

15 Rob J. van Glabbeek. The linear time – branching time spectrum II. In *Proceedings of CONCUR '93*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81, 1993. doi:10.1007/3-540-57208-2_6.

16 Rob J. van Glabbeek. The linear time – branching time spectrum I. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. North-Holland / Elsevier, 2001. doi:10.1016/b978-044482830-9/50019-9.

17 Jan Friso Groote and Frits W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Inf. Comput.*, 100(2):202–260, 1992. doi:10.1016/0890-5401(92)90013-6.

18 Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *J. Assoc. Comput. Mach.*, 32:137–161, 1985. doi:10.1145/2455.2460.

19 Charles A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

20 Robert M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, 1976. doi:10.1145/360248.360251.

21 Robin Milner. *Communication and Concurrency*. PHI Series in computer science. Prentice Hall, 1989.

22 Faron Moller. *Axioms for Concurrency*. PhD thesis, Department of Computer Science, University of Edinburgh, 1989. Report CST-59-89. Also published as ECS-LFCS-89-84.

23 Faron Moller. The importance of the left merge operator in process algebras. In *Proceedings of ICALP '90*, volume 443 of *Lecture Notes in Computer Science*, pages 752–764, 1990. doi:10.1007/BFb0032072.

24 Faron Moller. The nonexistence of finite axiomatisations for CCS congruences. In *Proceedings of LICS '90*, pages 142–153. IEEE Computer Society, 1990. doi:10.1109/LICS.1990.113741.

25 David M.R. Park. Concurrency and automata of infinite sequences. In 5^{th} *GI Conference*, volume 104 of *LNCS*, pages 167–183. Springer, 1981.

26 Gordon D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Aarhus University, 1981.

27 William C. Rounds and Stephen D. Brookes. Possible futures, acceptances, refusals, and communicating processes. In *Proceedings of Annual Symposium on Foundations of Computer Science*, pages 140–149, 1981. doi:10.1109/SFCS.1981.36.

A Proof of Theorem 23

Before proceeding to the proof we introduce some auxiliary results.

For $k \geq 0$, we denote by $\mathrm{var}_k(t)$ the set of variables occurring in the k-derivatives of t, namely $\mathrm{var}_k(t) = \{x \in \mathrm{var}(t') \mid t \xrightarrow{\alpha} t', |\alpha| = k\}$.

▶ **Lemma 33.** *Let t, u be two* $\mathrm{BCCSP}_{\parallel}$ *terms. If $t \sim_{\mathtt{PF}} u$ then:*

1. *For each $k \geq 0$ it holds that $\mathrm{var}_k(t) = \mathrm{var}_k(u)$.*
2. *t has a summand x, for some variable x, if and only if u does.*
3. $\mathrm{norm}(t) = \mathrm{norm}(u)$ *and* $\mathrm{depth}(t) = \mathrm{depth}(u)$.

The following result is immediate.

▶ **Lemma 34.** *Let t be a* $\mathrm{BCCSP}_{\parallel}$ *term, and let σ be a closed substitution. If $x \in \mathrm{var}(t)$ then* $\mathrm{depth}(\sigma(t)) \geq \mathrm{depth}(\sigma(x))$.

▶ **Proposition 28.** *Let $t \approx u$ be an equation over* $\mathrm{BCCSP}_{\parallel}$ *that is sound modulo $\sim_{\mathtt{PF}}$. Let σ be a closed substitution with $p = \sigma(t)$ and $q = \sigma(u)$. Suppose that p and q have neither $\mathbf{0}$ summands nor $\mathbf{0}$ factors, and that $p, q \sim_{\mathtt{PF}} a \parallel p_N$ for some N larger than the sizes of t and u. If p has a summand possible futures equivalent to $a \parallel p_N$, then so does q.*

Proof. Observe, first of all, that since $\sigma(t) = p$ and $\sigma(u) = q$ have no $\mathbf{0}$ summands or factors, then neither do t and u. We can therefore assume that, for some finite index sets $I, J \neq \emptyset$,

$$t = \sum_{i \in I} t_i \quad \text{and} \quad u = \sum_{j \in J} u_j \ , \tag{1}$$

where none of the t_i $(i \in I)$ and u_j $(j \in J)$ is $\mathbf{0}$ or has $+$ as its head operator. Note that, as t and u have no $\mathbf{0}$ summands or factors, then none of the t_i $(i \in I)$ and u_j $(j \in J)$ does either.

Since $p = \sigma(t)$ has a summand that is possible futures equivalent to $a \parallel p_N$, there is an index $i \in I$ such that $\sigma(t_i) \sim_{\mathtt{PF}} a \parallel p_N$. Our aim is now to show that there is an index $j \in J$ such that $\sigma(u_j) \sim_{\mathtt{PF}} a \parallel p_N$, proving that $q = \sigma(u)$ has the required summand. This we proceed to do by a case analysis on the form t_i may have.

1. Case $t_i = x$ for some variable x. In this case, we have that $\sigma(x) \sim_{\mathtt{PF}} a \parallel p_N$ and t has x as a summand. As $t \approx u$ is sound with respect to possible futures equivalence, from $t \sim_{\mathtt{PF}} u$ we get $t \sim_{\mathtt{CT}} u$. Hence, by Lemma 33.2, we obtain that u has a summand x as well, namely there is an index $j \in J$ such that $u_j = x$. It is then immediate to conclude that $q = \sigma(u)$ has a summand which is possible futures equivalent to $a \parallel p_N$.
2. Case $t_i = ct'$ for some action $c \in \{a, b\}$ and term t'. This case is vacuous because, since $\sigma(t_i) = c\sigma(t') \xrightarrow{c} \sigma(t')$ is the only transition afforded by $\sigma(t_i)$, this term cannot be possible futures equivalent to $a \parallel p_N$.

3. Case $t_i = t' \| t''$ for some terms t', t''. We have that $\sigma(t_i) = \sigma(t') \| \sigma(t'') \sim_{\mathtt{PF}} a \| p_N$. As $\sigma(t_i)$ has no $\mathbf{0}$ factors, it follows that $\sigma(t') \not\sim_{\mathtt{PF}} \mathbf{0}$ and $\sigma(t'') \not\sim_{\mathtt{PF}} \mathbf{0}$. Thus, by Proposition 26, we can infer that, without loss of generality, $\sigma(t') \sim_{\mathtt{PF}} a$ and $\sigma(t'') \sim_{\mathtt{PF}} p_N$. Notice that $\sigma(t'') \sim_{\mathtt{PF}} p_N$ implies $\mathtt{CT}(\sigma(t'')) = \mathtt{CT}(p_N)$. Now, t'' can be written in the general form $t'' = v_1 + \cdots + v_l$ for some $l > 0$, where none of the summands v_h is $\mathbf{0}$ or a sum. By Lemma 25, $\sigma(t'') \sim_{\mathtt{PF}} p_N$ implies that for each $i \in \{1, \ldots, N\}$ there is a summand r_i of $\sigma(t'')$ such that $b^i a \sim_{\mathtt{PF}} r_i$, and for each summand r of $\sigma(t'')$ there is an $i_r \in \{1, \ldots, N\}$ such that $r \sim_{\mathtt{PF}} b^i a$. Observe that, since N is larger than the size of t, we have that $l < N$. Hence, there must be some $h \in \{1, \ldots, l\}$ such that $\sigma(v_h) \sim_{\mathtt{S}} \sum_{k=1}^{m} b^{i_k} a$ for some $m > 1$ and $1 \leq i_1 < \ldots < i_m \leq N$. The term $\sigma(v_h)$ has no $\mathbf{0}$ summands or factors, or else, so would $\sigma(t'')$ and $\sigma(t)$. By Lemma 27, it follows that v_h can only be a variable x and thus that

$$\sigma(x) \sim_{\mathtt{PF}} \sum_{k=1}^{m} b^{i_k} a \ . \tag{2}$$

Observe, for later use, that the above equation yields that $x \notin \mathrm{var}(t')$, or else $\sigma(t') \not\sim_{\mathtt{PF}} a$ due to Lemma 34. So, modulo possible futures equivalence, t_i has the form $t' \parallel (x + t''')$, for some term t''', with $x \notin \mathrm{var}(t')$, $\sigma(t') \sim_{\mathtt{PF}} a$ and $\sigma(x + t'') \sim_{\mathtt{PF}} p_N$.

Our order of business will now be to show that u has a summand u_j such that $\sigma(u_j)$ is possible futures equivalent to $a \parallel p_N$. We recall that $t \sim_{\mathtt{PF}} u$ implies $t \sim_{\mathtt{CT}} u$. Thus, by Lemma 33.1 we obtain that $\mathrm{var}_k(t) = \mathrm{var}_k(u)$ for all $k \geq 0$. Hence, from $x \in \mathrm{var}_0(t_i) = \mathrm{var}(t_i)$ we get that there is at least one $j \in J$ such that $x \in \mathrm{var}_0(u_j) = \mathrm{var}(u_j)$.

So, firstly, we show that x cannot occur in the scope of prefixing in u_j, namely u_j cannot be of the form $c.u'$ or $(c.u' + u'') \parallel u'''$ for some $c \in \{a, b\}$ and u' with $x \in \mathrm{var}(u')$. We proceed by a case analysis:

   a. $c = b$ and $u_j = (b.u' + u'') \parallel u'''$ for some $u', u'', u''' \in \mathrm{BCCSP}_{\parallel}$ with $x \in \mathrm{var}(u')$. As $\sigma(u)$ does not have $\mathbf{0}$ summands or factors we have that $\sigma(u''') \not\sim_{\mathtt{PF}} \mathbf{0}$. Let $D = \max\{d \mid x \in \mathrm{var}_d(u')\}$. From $\sigma(x) \sim_{\mathtt{PF}} \sum_{k=1}^{m} b^{i_k} a$ and $\mathtt{CT}(\sigma(u)) = \mathtt{CT}(a \parallel p_N)$ we can infer that the completed traces of $\sigma(u''')$ are of the form $b^i a$, for some $i \in \{0, \ldots, N - i_m - D - 1\}$. Let $\alpha \in \mathtt{T}(\sigma(u'))$ be such that $|\alpha| = D$ and $u' \xrightarrow{\alpha} w$ with $x \in \mathrm{var}(w)$. By the choice of D, we can infer that x does not occur in the scope of prefixing in w, and thus $\mathtt{T}(\sigma(x)) \subseteq \mathtt{T}(\sigma(w))$. Then we get that $(b^i ab\alpha, \mathtt{T}(\sigma(w))) \in \mathtt{PF}(\sigma(u))$, where $b^i a \in \mathtt{CT}(\sigma(u'''))$. However, as $m \geq 2$, there is no p' such that $a \parallel p_N \xrightarrow{b^i ab\alpha} p'$ and $\mathtt{T}(\sigma(x)) \subseteq \mathtt{T}(p')$, thus giving $(b^i ab\alpha, \mathtt{T}(\sigma(w))) \notin \mathtt{PF}(a \parallel p_N)$. This contradicts $\sigma(u) \sim_{\mathtt{PF}} a \parallel p_N$.
   b. $c = b$ and $u_j = b.u'$ for some $\mathrm{BCCSP}_{\parallel}$ term u' with $x \in \mathrm{var}(u')$. The proof is similar to the one of the previous case and it is therefore omitted.
   c. $c = a$ and $u_j = (a.u' + u'') \parallel u'''$ for some $u', u'', u''' \in \mathrm{BCCSP}_{\parallel}$ with $x \in \mathrm{var}(u')$. As $\sigma(u)$ does not have $\mathbf{0}$ summands or factors we have that $\sigma(u''') \not\sim_{\mathtt{PF}} \mathbf{0}$. From $\sigma(x) \sim_{\mathtt{PF}} \sum_{k=1}^{m} b^{i_k} a$ we infer that $\mathtt{T}(a.\sigma(u'))$ includes traces having two occurrences of action a. Since $\sigma(u) \sim_{\mathtt{PF}} a \parallel p_N$, this implies that there is no $\alpha \in \mathtt{T}(\sigma(u'''))$ such that α contains an occurrence of action a, for otherwise $\sigma(u)$ could perform a trace having 3 occurrences of that action. In particular, this implies that the last symbol in each trace of $\sigma(u''')$ must be action b. This gives that there is at least one completed trace of $\sigma(u_j)$, and thus of $\sigma(u)$, whose last symbol is action b. Hence we get $\mathtt{CT}(\sigma(u)) \neq \mathtt{CT}(a \parallel p_N)$, which contradicts $\sigma(u) \sim_{\mathtt{PF}} a \parallel p_N$.
   d. $c = a$ and $u_j = a.u'$ for some $\mathrm{BCCSP}_{\parallel}$ term u' with $x \in \mathrm{var}(u')$. In this case we are going to prove a slightly weaker property, namely that not all summands u_j with $x \in \mathrm{var}(u_j)$ can be of this form. Consider the closed substitution σ' defined by

$$\sigma'(y) = \begin{cases} ap_N & \text{if } y = x \\ \sigma(y) & \text{otherwise.} \end{cases}$$

Then we have that $\sigma'(t_i) = \sigma'(t') \parallel \sigma'(x) + \sigma'(t''') \xrightarrow{a} \sigma(t') \parallel p_N \sim_{\mathtt{PF}} a \parallel p_N$. Since $\sigma'(t) \sim_{\mathtt{PF}} \sigma'(u)$ then there is a process r such that $\sigma'(u) \xrightarrow{a} r$ and $\mathtt{T}(r) = \mathtt{T}(a \parallel p_N)$. In particular, this means that $\mathrm{depth}(r) = N + 2$. Hence, from the choices of N, σ and σ', we can infer that such an a-move by $\sigma'(u)$ can only stem from a summand u_j such that $x \in \mathrm{var}(u_j)$. Assume, towards a contradiction, that all such summands u_j are of the form $a.u'_j$ for some $\mathrm{BCCSP}_\parallel$ term u'_j with $x \in \mathrm{var}(u'_j)$ and $r = \sigma'(u'_j)$. As $\mathrm{depth}(\sigma'(u'_j)) = N + 2 = \mathrm{depth}(\sigma'(x))$, by Lemma 34 we get that u'_j can only be of the form $u'_j = x + w_j$ for some $\mathrm{BCCSP}_\parallel$ term w_j with $\mathrm{depth}(\sigma'(w_j)) \leq N + 2$. Notice that $\mathtt{T}(\sigma'(x)) \subset \mathtt{T}(a \parallel p_N)$. Hence $\sigma'(w_j) \neq \mathbf{0}$. More precisely, $\sigma'(x) = ap_N$ implies that $\{b\alpha \mid b\alpha \in \mathtt{T}(a \parallel p_N)\} \subseteq \mathtt{T}(\sigma'(w_j)) \subseteq \mathtt{T}(a \parallel p_N)$. Clearly, no trace starting with action b can stem from $\sigma'(x)$ and we can then infer, in light of Lemma 34, that $x \notin \mathrm{var}(w_j)$, as $\mathrm{depth}(\sigma'(w_j)) \leq N + 2$. This implies that $\sigma'(w_j) = \sigma(w_j)$ and thus $\{b\alpha \mid b\alpha \in \mathtt{T}(a \parallel p_N)\} \subseteq \mathtt{T}(\sigma(w_j)) \subseteq \mathtt{T}(a \parallel p_N)$. In particular, $\sigma(w_j)$ can perform at least one (completed) trace of the form $b\alpha$ where α contains two occurrences of action a. From $\sigma(u_j) = a.(\sigma(x) + \sigma(w_j))$, then get that $(ab\alpha, \emptyset) \in \mathtt{PF}(\sigma(u))$, namely $\sigma(u)$ can perform at least one (completed) trace containing 3 occurrences of action a. This gives a contradiction with $\sigma(u) \sim_{\mathtt{PF}} a \parallel p_N$.

We have therefore obtained that x does not occur in the scope of prefixing in (at least one) u_j. We proceed now by a case analysis on the possible forms of this summand.

a. $u_j = x$. Then, modulo possible futures equivalence, $\sigma(u)$ has the form $r' + \sum_{k=1}^{m} b^{i_k} a$ for some r'. We show that this contradicts $\sigma(u) \sim_{\mathtt{PF}} a \parallel p_N$. This follows directly by noticing that, due to the summand $b^{i_1} a$, we have that $(b^{i_1} a, \emptyset) \in \mathtt{PF}(\sigma(u))$. However, $(b^{i_1} a, \emptyset) \notin \mathtt{PF}(a \parallel p_N)$, since $a \parallel p_N$ by performing the trace $b^{i_1} a$ can reach either a process that can perform an a (in case the first b-move is performed by the summand $b^{i_1} a$ of p_N) or a b (in case the first b-move is performed by a summand $b^i a$ of p_N such that $i > i_1$).

b. $u_j = (x + w) \parallel w'$, for some terms w, w' with $w' \not\sim_{\mathtt{PF}} \mathbf{0}$. From $\sigma(u) \sim_{\mathtt{PF}} a \parallel p_N$, we infer that $\mathtt{CT}(\sigma(u_j)) \subseteq \mathtt{CT}(a \parallel p_N)$. We recall that no completed trace of $a \parallel p_N$ has b as last symbol and, moreover, in all the completed traces of $a \parallel p_N$ there are exactly two occurrences of a. Hence, all (nonempty) completed traces of $\sigma(x), \sigma(w)$ and $\sigma(w')$ must have exactly one occurrence of a and this occurrence must be as the last symbol in the completed trace.

We now proceed to show that $\sigma(w')$ has a summand a and $a \notin \mathtt{I}(\sigma(x) + \sigma(w))$. We start by noticing that it cannot be the case that $a \in \mathtt{I}(\sigma(x) + \sigma(w)) \cap \mathtt{I}(\sigma(w'))$, for otherwise we would have $a^2 \in \mathtt{T}(\sigma(u_j)) \subseteq \mathtt{T}(\sigma(u))$, thus contradicting $\sigma(u) \sim_{\mathtt{PF}} a \parallel p_N$. Assume now, towards a contradiction, that $\mathtt{I}(\sigma(w')) = \{b\}$. Then, due to summand $b^{i_m} a$ of $\sigma(x)$, we have that $\sigma(u_j) \xrightarrow{b^{i_m - 1}} ba \parallel \sigma(w')$ and $a\alpha \notin \mathtt{T}(ba \parallel \sigma(w'))$ for any trace $\alpha \in \mathcal{A}^*$. Clearly, $(b^{i_m - 1}, \mathtt{T}(ba \parallel \sigma(w'))) \in \mathtt{PF}(\sigma(u_j))$, and thus it is also a possible future of $\sigma(u)$. However, $(b^{i_m - 1}, \mathtt{T}(ba \parallel \sigma(w'))) \notin \mathtt{PF}(a \parallel p_N)$, as the interleaving of p_N with a guarantees that after an initial trace of an arbitrary number of b-transitions it is always possible to perform a trace starting with a. This gives a contradiction with $\sigma(u) \sim_{\mathtt{PF}} a \parallel p_N$. We have obtained that $a \in \mathtt{I}(\sigma(w'))$. More precisely, from the constraints on the completed traces of $\sigma(w')$, we infer that $\sigma(w')$ has a summand a.

Our order of business will now be to show that $\sigma(w') \sim_{\mathtt{PF}} a$. Since $\sigma(w') \xrightarrow{a} \mathbf{0}$, we have that $\sigma(u_j) \xrightarrow{a} (\sigma(x)+\sigma(w)) \parallel \mathbf{0} \sim_{\mathtt{PF}} \sigma(x)+\sigma(w)$. Thus, $\sigma(u) \sim_{\mathtt{PF}} a \parallel p_N$ implies that $a \parallel p_N \xrightarrow{a} r$ for some r with $\mathtt{T}(r) = \mathtt{T}(\sigma(x)+\sigma(w))$. Since $a \parallel p_N$ has only one possible initial a-transition, namely $a \parallel p_N \xrightarrow{a} \mathbf{0} \parallel p_N$, we get that $r \sim_{\mathtt{PF}} p_N$ and thus $\mathtt{T}(\sigma(x)+\sigma(w)) = \mathtt{T}(p_N)$. In particular, this implies that $\mathrm{depth}(\sigma(x)+\sigma(w)) = N+1$. Therefore, we have

$$\begin{aligned} 1 \leq \mathrm{depth}(\sigma(w')) &= \mathrm{depth}(\sigma(u_j)) - \mathrm{depth}(\sigma(x)+\sigma(w)) \\ &= \mathrm{depth}(\sigma(u_j)) - (N+1) \\ &\leq \mathrm{depth}(\sigma(u)) - (N+1) \\ &= \mathrm{depth}(a \parallel p_N) - (N+1) && \text{(by Lem. 33.3)} \\ &= N+2-(N+1) \\ &= 1 \end{aligned}$$

and we can therefore conclude that $\sigma(w') \sim_{\mathtt{PF}} a$. Furthermore, it is not difficult to prove that $\mathtt{CT}(\sigma(x)+\sigma(w)) = \mathtt{CT}(p_N)$, for otherwise we get a contradiction with $\sigma(u) \sim_{\mathtt{PF}} a \parallel p_N$.

So far we have obtained that, modulo possible futures equivalence,

$$\sigma(u_j) \sim_{\mathtt{PF}} \left(\sum_{k=1}^{m} b^{i_k}a + \sigma(w)\right) \parallel a \text{ and } \mathtt{CT}(\sum_{k=1}^{m} b^{i_k}a + \sigma(w)) = \{b^i a \mid i \in \{1,\ldots,N\}\} \ .$$

To conclude the proof, we need to show that $\sum_{k=1}^{m} b^{i_k}a + \sigma(w) \sim_{\mathtt{PF}} p_N$. Let $I_m = \{i_1,\ldots,i_m\}$ and $I_N = \{1,\ldots,N\}$. Assume, towards a contradiction, that $\sum_{k=1}^{m} b^{i_k}a + \sigma(w) \not\sim_{\mathtt{PF}} p_N$. Notice that $\sigma(w)$ can be written in the general form $\sigma(w) = \sum_{l\in L} q_l$ for some terms q_l that do not have $+$ as head operator nor contain any occurrence of $\parallel$. By Lemma 25, this means that either there is an $i \in I_N \setminus I_m$ such that $b^i a \not\sim_{\mathtt{PF}} q_l$ for any $l \in L$, or that there is a summand q_l of $\sigma(w)$ such that $q_l \not\sim_{\mathtt{PF}} b^i a$ for any $i \in I_N$. In both cases, we obtain that there is (at least) a summand q_l of $\sigma(w)$ such that $b^k a, b^h a \in \mathtt{CT}(q_l)$ for some $k \neq h, h, k \in I_N$. We can then proceed as in the proof of Lemma 25 to prove that this gives the desired contradiction. We have therefore obtained that $\sum_{k=1}^{m} b^{i_k}a + \sigma(w) \sim_{\mathtt{PF}} p_N$. Hence, by congruence closure, we get that $\sigma(u_j) \sim_{\mathtt{PF}} a \parallel p_N$ and we can therefore conclude that $\sigma(u)$ has the desired summand. This concludes the proof. ◀

Finally, we can formally prove Theorem 29.

▶ **Theorem 29.** *Let $\mathcal{E}$ be a finite axiom system over* $\mathrm{BCCSP}_{\parallel}$ *that is sound modulo $\sim_{\mathtt{PF}}$. Let N be larger than the size of each term in the equations in $\mathcal{E}$. Assume that p and q are closed terms that contain no occurrences of $\mathbf{0}$ as a summand or factor, and that $p, q \sim_{\mathtt{PF}} a \parallel p_N$. If $\mathcal{E} \vdash p \approx q$ and p has a summand possible futures equivalent to $a \parallel p_N$, then so does q.*

Proof. Assume that $\mathcal{E}$ is a finite axiom system over the language $\mathrm{BCCSP}_{\parallel}$ that is sound modulo possible futures equivalence, and that the following hold, for some closed terms p and q and positive integer N larger than the size of each term in the equations in $\mathcal{E}$:

1. $E \vdash p \approx q$,
2. $p \sim_{\mathtt{PF}} q \sim_{\mathtt{PF}} a \parallel p_N$,
3. p and q contain no occurrences of $\mathbf{0}$ as a summand or factor, and
4. p has a summand possible futures equivalent to $a \parallel p_N$.

We prove that q also has a summand possible futures equivalent to $a \parallel p_N$ by induction on the depth of the closed proof of the equation $p \approx q$ from $\mathcal{E}$. Without loss of generality, we may assume that the closed terms involved in the proof of the equation $p \approx q$ have no $\mathbf{0}$ summands or factors, and that applications of symmetry happen first in equational proofs (that is, $\mathcal{E}$ is closed with respect to symmetry).

We proceed by a case analysis on the last rule used in the proof of $p \approx q$ from $\mathcal{E}$. The case of reflexivity is trivial, and that of transitivity follows immediately by using the inductive hypothesis twice. Below we only consider the other possibilities.

- Case $E \vdash p \approx q$, because $\sigma(t) = p$ and $\sigma(u) = q$ for some equation $(t \approx u) \in E$ and closed substitution σ. Since $\sigma(t) = p$ and $\sigma(u) = q$ have no $\mathbf{0}$ summands or factors, and N is larger than the size of each term mentioned in equations in $\mathcal{E}$, the claim follows by Proposition 28.
- Case $E \vdash p \approx q$, because $p = cp'$ and $q = cq'$ for some p', q' such that $E \vdash p' \approx q'$, and for some action c. This case is vacuous because $p = cp' \not\sim_{\mathtt{PF}} a \parallel p_N$, and thus p does not have a summand possible futures equivalent to $a \parallel p_N$.
- Case $E \vdash p \approx q$, because $p = p' + p''$ and $q = q' + q''$ for some p', q', p'', q'' such that $E \vdash p' \approx q'$ and $E \vdash p'' \approx q''$. Since p has a summand possible futures equivalent to $a \parallel p_N$, we have that so does either p' or p''. Assume, without loss of generality, that p' has a summand possible futures equivalent to $a \parallel p_N$. Since p is possible futures equivalent to $a \parallel p_N$, so is p'. Using the soundness of $\mathcal{E}$ modulo possible futures equivalence, it follows that $q' \sim_{\mathtt{PF}} a \parallel p_N$. The inductive hypothesis now yields that q' has a summand possible futures equivalent to $a \parallel p_N$. Hence, q has a summand possible futures equivalent to $a \parallel p_N$, which was to be shown.
- Case $E \vdash p \approx q$, because $p = p' \parallel p''$ and $q = q' \parallel q''$ for some p', q', p'', q'' such that $E \vdash p' \approx q'$ and $E \vdash p'' \approx q''$. Since the proof involves no uses of $\mathbf{0}$ as a summand or a factor, we have that $p', p'' \not\sim_{\mathtt{PF}} \mathbf{0}$ and $q', q'' \not\sim_{\mathtt{PF}} \mathbf{0}$. It follows that q is a summand of itself. By our assumptions, $q' \parallel q'' \sim_{\mathtt{PF}} a \parallel p_N$ which, by Proposition 26 gives that either $q' \sim_{\mathtt{S}} a$ and $q'' \sim_{\mathtt{S}} p_N$, or $q' \sim_{\mathtt{S}} p_N$ and $q'' \sim_{\mathtt{S}} a$. In both cases, we can conclude that q has itself as summand of the required form.

This completes the proof of Theorem 29 and thus of Theorem 23. ◀

Wreath/Cascade Products and Related Decomposition Results for the Concurrent Setting of Mazurkiewicz Traces

Bharat Adsul
IIT Bombay, India
adsul@cse.iitb.ac.in

Paul Gastin
LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France
paul.gastin@ens-paris-saclay.fr

Saptarshi Sarkar
IIT Bombay, India
sapta@cse.iitb.ac.in

Pascal Weil
Université Bordeaux, LaBRI, CNRS UMR 5800, Talence, France
CNRS, ReLaX, IRL 2000, Siruseri, India
pascal.weil@labri.fr

Abstract

We develop a new algebraic framework to reason about languages of Mazurkiewicz traces. This framework supports true concurrency and provides a non-trivial generalization of the wreath product operation to the trace setting. A novel local wreath product principle has been established. The new framework is crucially used to propose a decomposition result for recognizable trace languages, which is an analogue of the Krohn-Rhodes theorem. We prove this decomposition result in the special case of acyclic architectures and apply it to extend Kamp's theorem to this setting. We also introduce and analyze distributed automata-theoretic operations called local and global cascade products. Finally, we show that aperiodic trace languages can be characterized using global cascade products of localized and distributed two-state reset automata.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Algebraic language theory

Keywords and phrases Mazurkiewicz traces, asynchronous automata, wreath product, cascade product, Krohn Rhodes decomposition theorem, local temporal logic over traces

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.19

Related Version An extended version is available at `https://arxiv.org/abs/2007.07940`.

Funding *Paul Gastin*: Supported by IRL ReLaX.
Pascal Weil: Partially supported by the DeLTA project (ANR-16-CE40- 0007)

1 Introduction

Transformation monoids provide an abstraction of transition systems. One of the key tools in their analysis is the notion of wreath product [7, 20, 18] which, when translated to the language of finite state automata, corresponds to the cascade product. In the cascade product of automata A and B, with A "followed by" B, the automaton A runs on the input sequence, while the automaton B runs on the input sequence as well as the state sequence produced by the automaton A. The wreath product principle (see [20, 18, 17]) is a key result which relates a language accepted by a cascade/wreath product to languages accepted by the individual automata.

In this work, we are interested in generalizing the wreath product operation from the sequential setting to the concurrent setting involving multiple processes. Towards this, we work with Mazurkiewicz traces (or simply traces) [12, 6] which are well established as models of true concurrency, and asynchronous automata [21] which are natural distributed finite state devices working on traces. A trace represents a concurrent behaviour as a labelled partial order which faithfully captures the distribution of events across processes, and causality and concurrency between them. An asynchronous automaton runs on the input trace in a distributed fashion and respects the underlying causality and concurrency between events. During the run, when working on an event, only the local states of the processes participating in that event are updated; the rest of the processes remain oblivious to the occurrence of the event at this point.

A natural generalization of the above mentioned sequential cascade product to asynchronous automata A and B is as follows: the asynchronous automaton A runs on the input trace, thus assigning, for each event, a local state for every process participating in that event. Now the asynchronous automaton B runs on the input trace with the *same* set of events which are *additionally* labelled by the previous local states of the participating processes in A. It is easy to capture this operational semantics by another asynchronous automaton which we call the *local* cascade product of A and B. Such a construction is used in [2] to provide an asynchronous automata-theoretic characterization of aperiodic trace languages.

Here we propose a new algebraic framework to deal with the issues posed by the concurrent setting. More precisely, we introduce a new class of transformation monoids called *asynchronous transformation monoids* (in short, *atm's*). These monoids make a clear distinction between local and global "states" and allow us to reason about whether a global transformation is essentially induced by a particular subset of processes. Recall that, from a purely algebraic viewpoint, the set of all traces forms a free partially commutative monoid in which *independent* actions commute [6]. In order to recognize a trace language via an atm, we introduce the notion of an *asynchronous morphism* which exploits the locality of the underlying atm. It is rather easy to see that asynchronous morphisms are the algebraic counterparts of asynchronous automata.

One of the central results here is a wreath product principle in the new algebraic framework. It turns out that the *standard* wreath product operation yields an operation on asynchronous transformation monoids. Let T_1 and T_2 be atm's and $T_1 \wr T_2$ be the wreath product atm. Our *local* wreath product principle describes a trace language recognized by $T_1 \wr T_2$ in terms of a *local asynchronous transducer* which is a natural *causality and concurrency preserving* map from traces to traces (over an appropriately extended alphabet), and trace languages recognized by T_1 and T_2. It is a novel generalization of the standard wreath product principle. The work [8] presents a wreath product principle for traces in the setting of transformation monoids but it seems less significant since it uses *non-trace* structures.

The importance of the standard wreath product operation is clearly highlighted by the fundamental Krohn-Rhodes decomposition theorem [11] which, broadly speaking, says that any finite transformation monoid can be simulated by wreath products of "simple" transformation monoids. The wreath product principle along with the Krohn-Rhodes theorem can be used to provide alternate and conceptually simpler proofs (see [14, 3]) of several interesting classical results about formal languages of words such as the theorems due to Schützenberger [19], McNaughton-Papert [13] and Kamp [9] which together show the equivalence between star-free, aperiodic, first-order-definable and linear-temporal-logic definable word languages. Motivated by these applications, we investigate an analogue of the fundamental Krohn-Rhodes decomposition theorem over traces. We use the new algebraic

framework to propose a simultaneous generalization of the Krohn-Rhodes theorem (for word languages) and Zielonka's theorem (for trace languages). The proof of this generalization for the special case of acyclic architectures is another significant result. As an application, we extend Kamp's theorem: we show a natural local temporal logic to be expressively complete.

It turns out that asynchronous morphisms into wreath products correspond to the aforementioned distributed automata-theoretic local cascade products. We also introduce the *global* cascade product operation and show that it can be realized as the local cascade product with the help of the ubiquitous gossip automaton from [16].

Our final major contribution concerns aperiodic trace languages and is in the spirit of the Krohn-Rhodes theorem for the aperiodic case. We establish that aperiodic trace languages can be characterized using global cascade products of localized and distributed two-state reset automata. The proof of this characterization crucially uses an expressively complete process-based local temporal logic over traces from [4].

The rest of the paper is organized as follows. After setting up the preliminaries in Section 2, we develop the new algebraic framework in Section 3 and establish the local wreath product principle. In Section 4, we postulate a new decomposition result, and we prove it for acyclic architectures. We introduce and analyze local and global cascade products in Section 5. The global cascade product based characterization of aperiodic trace languages appears in Section 6. Finally, we conclude in Section 7. Due to space constraints some proofs and examples are omitted and can be found in the extended version [1].

2 Preliminaries

2.1 Basic Notions in Trace Theory

Let $\mathcal{P}$ be a finite set of agents/processes. A *distributed alphabet* over $\mathcal{P}$ is a family $\widetilde{\Sigma} = \{\Sigma_i\}_{i\in\mathcal{P}}$. Let $\Sigma = \bigcup_{i\in\mathcal{P}} \Sigma_i$. For $a \in \Sigma$, we set $\mathrm{loc}(a) = \{i \in \mathcal{P} \mid a \in \Sigma_i\}$. By (Σ, I) we denote the corresponding trace alphabet, i.e., I is the *independence relation* $\{(a,b) \in \Sigma^2 \mid \mathrm{loc}(a) \cap \mathrm{loc}(b) = \emptyset\}$ induced by $\widetilde{\Sigma}$. The corresponding *dependence relation* $\Sigma^2 \setminus I$ is denoted by D.

A Σ-labelled poset is a structure $t = (E, \leq, \lambda)$ where E is a set, $\leq$ is a partial order on E and $\lambda\colon E \to \Sigma$ is a labelling function. For $e, e' \in E$, define $e \lessdot e'$ if and only if $e < e'$ and for each e'' with $e \leq e'' \leq e'$ either $e = e''$ or $e' = e''$. For $X \subseteq E$, let $\downarrow X = \{y \in E \mid y \leq x \text{ for some } x \in X\}$. For $e \in E$, we abbreviate $\downarrow\{e\}$ by simply $\downarrow e$.

A *trace* over $\widetilde{\Sigma}$ is a finite Σ-labelled poset $t = (E, \leq, \lambda)$ such that

- If $e, e' \in E$ with $e \lessdot e'$ then $(\lambda(e), \lambda(e')) \in D$
- If $e, e' \in E$ with $(\lambda(e), \lambda(e')) \in D$, then $e \leq e'$ or $e' \leq e$

Let $TR(\widetilde{\Sigma})$ denote the set of all traces over $\widetilde{\Sigma}$. Henceforth a trace means a trace over $\widetilde{\Sigma}$ unless specified otherwise. Let $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$. The elements of E are referred to as *events* in t and for an event e in t, $\mathrm{loc}(e)$ abbreviates $\mathrm{loc}(\lambda(e))$. Further, let $i \in \mathcal{P}$. The set of i-events in t is $E_i = \{e \in E \mid i \in \mathrm{loc}(e)\}$. This is the set of events in which process i participates. It is clear that E_i is totally ordered by $\leq$.

A subset $c \subseteq E$ is a *configuration* of t if and only if $\downarrow c = c$. We let $\mathcal{C}_t$ denote the set of all configurations of t. Notice that $\emptyset$, the empty set, and E are configurations. More importantly, $\downarrow e$ is a configuration for every $e \in E$. There are two natural transition relations that one may associate with the configurations of t. The event based transition relation $\Rightarrow_t \subseteq \mathcal{C}_t \times E \times \mathcal{C}_t$ is defined by $c \overset{e}{\Rightarrow}_t c'$ if and only if $e \notin c$ and $c \cup \{e\} = c'$. The action based transition relation $\to_t \subseteq \mathcal{C}_t \times \Sigma \times \mathcal{C}_t$ is defined by $c \xrightarrow{a}_t c'$ if and only if there exists $e \in E$ such that $\lambda(e) = a$ and $c \overset{e}{\Rightarrow}_t c'$.

Now we turn our attention to the important operation of concatenation of traces. Let $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$ and $t' = (E', \leq', \lambda') \in TR(\widetilde{\Sigma})$. Without loss of generality, we can assume E and E' to be disjoint. We define $tt' \in TR(\widetilde{\Sigma})$ to be the trace $(E'', \leq'', \lambda'')$ where

- $E'' = E \cup E'$,
- $\leq''$ is the transitive closure of $\leq \cup \leq' \cup \{(e, e') \in E \times E' \mid (\lambda(e), \lambda'(e')) \in D\}$,
- $\lambda'' \colon E'' \to \Sigma$ where $\lambda''(e) = \lambda(e)$ if $e \in E$; otherwise, $\lambda''(e) = \lambda'(e)$.

This operation, henceforth referred to as *trace concatenation*, gives $TR(\widetilde{\Sigma})$ a monoid structure. Observe that, with a (resp. b) denoting the singleton trace with the only event labelled a (resp. b), if $(a, b) \in I$ then $ab = ba$ in $TR(\widetilde{\Sigma})$.

A basic result in trace theory gives a presentation of the trace monoid as a quotient of the *free* word monoid Σ^*. See [6] for more details. Let $\sim_I \subseteq \Sigma^* \times \Sigma^*$ be the congruence generated by $ab \sim_I ba$ for $(a, b) \in I$.

▶ **Proposition 1.** *The canonical morphism from $\Sigma^* \to TR(\widetilde{\Sigma})$, sending a letter $a \in \Sigma$ to the trace a, factors through the quotient monoid $\Sigma^*/\sim_I$ and induces an isomorphism from $\Sigma^*/\sim_I$ to the trace monoid $TR(\widetilde{\Sigma})$.*

2.2 Transformation Monoids and Trace Languages

A map from a set X to itself is called a *transformation* of X. Under function composition, the set of all such transformations forms a monoid; let us denote this monoid by $\mathcal{F}(X)$. The function composition $f_1 f_2$ applies from left-to-right, that is, $(f_1 f_2)(\cdot) = f_2(f_1(\cdot))$.

A *transformation monoid* (or simply *tm*) is a pair $T = (X, M)$ where M is a submonoid of $\mathcal{F}(X)$. The tm (X, M) is called *finite* if X is finite.

▶ **Example 2.** Consider $X = \{1, 2\}$ with the monoid $M = \{\mathrm{id}_X, r_1, r_2\}$ where id_X is the identity transformation and r_i maps every element in X to element i. Note that $r_1 r_2 = r_2$ and $r_2 r_1 = r_1$. Then (X, M) is a tm. We will refer to it as U_2. ⌟

Let $T = (X, M)$ be a tm. By a morphism φ from $TR(\widetilde{\Sigma})$ to T, we mean a (monoid) morphism $\varphi \colon TR(\widetilde{\Sigma}) \to M$. We abuse the notation and also write this as $\varphi \colon TR(\widetilde{\Sigma}) \to T$. Observe that, if $(a, b) \in I$, then as $ab = ba$ in $TR(\widetilde{\Sigma})$, $\varphi(a)$ and $\varphi(b)$ must commute in M. In fact, in view of Proposition 1, any function $\varphi \colon \Sigma \to M$ which has the property that $\varphi(a)$ and $\varphi(b)$ commute for every $(a, b) \in I$, can be uniquely extended to a morphism from $TR(\widetilde{\Sigma})$ to M.

Transformation monoids can be naturally used to recognize trace languages. Let $L \subseteq TR(\widetilde{\Sigma})$ be a trace language. We say that L is *recognized by* a tm $T = (X, M)$ if there exists a morphism $\varphi \colon TR(\widetilde{\Sigma}) \to T$, an *initial* element $x_{\mathrm{in}} \in X$ and a *final* subset $X_{\mathrm{fin}} \subseteq X$ such that $L = \{t \in TR(\widetilde{\Sigma}) \mid \varphi(t)(x_{\mathrm{in}}) \in X_{\mathrm{fin}}\}$. A trace language is said to be *recognizable* if it is recognized by a finite tm.

3 New Algebraic Framework

3.1 Asynchronous Transformation Monoids

Recall that we have a fixed finite set $\mathcal{P}$ of processes. If $\mathcal{P}$ is clear from the context, we use the simpler notation $\{X_i\}$ to denote the $\mathcal{P}$-indexed family $\{X_i\}_{i \in \mathcal{P}}$. The elements of the sets in a $\mathcal{P}$-indexed family will be typically called *states*.

We begin with some notation involving local and global states. Suppose that each process $i \in \mathcal{P}$ is equipped with a finite non-empty set of *local i-states*, denoted S_i. We set $S = \bigcup_{i\in\mathcal{P}} S_i$ and call S the set of *local states*. We let P range over non-empty subsets of $\mathcal{P}$ and let i, j range over $\mathcal{P}$. A *P-state* is a map $s\colon P \to S$ such that $s(j) \in S_j$ for every $j \in P$. We let S_P denote the set of all P-states. We call $S_{\mathcal{P}}$ the set of all *global states*.

If $P' \subseteq P$ and $s \in S_P$ then $s_{P'}$ is s restricted to P'. We use the shorthand $-P$ to indicate the complement of P in $\mathcal{P}$. We sometimes split a global state $s \in S_{\mathcal{P}}$ as $(s_P, s_{-P}) \in S_P \times S_{-P}$. We let S_a denote the set of all $\mathrm{loc}(a)$-states which we also call a-states for simplicity. Thus if $\mathrm{loc}(a) \subseteq P$ and s is a P-state we shall write s_a to mean $s_{\mathrm{loc}(a)}$.

Now we are ready to introduce a new class of transformation monoids.

▶ **Definition 3.** *An* asynchronous transformation monoid *(in short, atm) T (over $\mathcal{P}$) is a pair $(\{S_i\}, M)$ where*

- *S_i is a finite non-empty set for each process $i \in \mathcal{P}$.*
- *M is a submonoid of $\mathcal{F}(S_{\mathcal{P}})$, the monoid of all transformations from $S_{\mathcal{P}}$ to itself.*

Note that this definition is dependent on $\mathcal{P}$ and an atm $T = (\{S_i\}, M)$ naturally induces the tm $(S_{\mathcal{P}}, M)$. We abuse the notation and write T also for this tm.

A crucial feature of the definition of an atm is that it makes a clear distinction between local and global states. Observe that the underlying transformations operate on global states. It will be useful to know whether a global transformation is essentially induced by a particular subset of processes. We develop some notions to make this precise.

Fix an atm $(\{S_i\}, M)$ and $P \subseteq \mathcal{P}$. Let $f\colon S_P \to S_P$ be a map. We define $g\colon S_{\mathcal{P}} \to S_{\mathcal{P}}$ as: for $s \in S_{\mathcal{P}}$,

$$g(s) = s' \text{ iff } f(s_P) = s'_P \text{ and } s_{-P} = s'_{-P}$$

We refer to g as the extension of f. More generally, $h\colon S_{\mathcal{P}} \to S_{\mathcal{P}}$ is said to be a P-map if it is the extension of some $f\colon S_P \to S_P$. Note that, in this case, for all $s = (s_P, s_{-P}) \in S_{\mathcal{P}}$, $h((s_P, s_{-P})) = (f(s_P), s_{-P})$ and f is uniquely determined by h. It is worth pointing out that a map $h\colon S_{\mathcal{P}} \to S_{\mathcal{P}}$ with the property that for every $s \in S_{\mathcal{P}}$ there exists $s'_P \in S_P$ such that $h((s_P, s_{-P})) = (s'_P, s_{-P})$ is *not* necessarily a P-map. This condition merely says that the $(-P)$-component of a global state is not updated by h. The update of the P-component may still depend on the $(-P)$-component.

The following lemma provides a characterization of P-maps. We skip the easy proof.

▶ **Lemma 4.** *Let $h\colon S_{\mathcal{P}} \to S_{\mathcal{P}}$. Then h is a P-map if and only if for every s in $S_{\mathcal{P}}$, $[h(s)]_{-P} = s_{-P}$ and for every s, s' in $S_{\mathcal{P}}$, $s_P = s'_P$ implies that $[h(s)]_P = [h(s')]_P$.*

A simple but crucial observation regarding P-maps is recorded in the following lemma.

▶ **Lemma 5.** *Let $f, g\colon S_{\mathcal{P}} \to S_{\mathcal{P}}$ be such that f is a P-map and g is a P'-map. If $P \cap P' = \emptyset$, then $fg = gf$.*

▶ **Example 6.** Fix a process $\ell \in \mathcal{P}$. We define the atm $U_2[\ell] = (\{S_i\}, M)$ where, $S_\ell = \{1, 2\}$ and for each $i \neq \ell$, S_i has exactly one element. Observe that $S_{\mathcal{P}}$ has only two global states which are completely determined by their ℓ-components. We will identify a global state with its ℓ-component. The monoid M is $\{\mathrm{id}_{S_{\mathcal{P}}}, r_1, r_2\}$ where $\mathrm{id}_{S_{\mathcal{P}}}$ is the identity transformation and r_i maps every global state to the global state i. Note that r_1 and r_2 are $\{\ell\}$-maps. ⌟

3.2 Asynchronous Morphisms

Now we fix a distributed alphabet $\widetilde{\Sigma} = \{\Sigma_i\}_{i\in\mathcal{P}}$ over $\mathcal{P}$ and introduce special morphisms from the trace monoid $TR(\widetilde{\Sigma})$ to atm's.

▶ **Definition 7.** *Let $T = (\{S_i\}, M)$ be an atm. An* asynchronous morphism *from $TR(\widetilde{\Sigma})$ to T is a (monoid) morphism $\varphi\colon TR(\widetilde{\Sigma}) \to M$ such that, for $a \in \Sigma$, $\varphi(a)$ is a $\mathrm{loc}(a)$-map (or simply, an a-map). We also write this as $\varphi\colon TR(\widetilde{\Sigma}) \to T$.*

It is important to observe that, contrary to the sequential case, a morphism from $TR(\widetilde{\Sigma})$ to M is not necessarily an asynchronous morphism from $TR(\widetilde{\Sigma})$ to the atm $T = (\{S_i\}, M)$. In a morphism $\psi\colon TR(\widetilde{\Sigma}) \to M$, for $(a, b) \in I$, $\psi(a)$ and $\psi(b)$ must commute; however for some $a \in \Sigma$, $\psi(a)$ may not be an a-map.

▶ **Example 8.** Consider $\mathcal{P} = \{p_1, p_2, p_3\}$, $\widetilde{\Sigma} = \{\Sigma_{p_1} = \{a, b\}, \Sigma_{p_2} = \{b, c\}, \Sigma_{p_3} = \{c\}\}$ and the atm $U_2[p_1] = (\{S_i\}, M)$ where $M = \{\mathrm{id}, r_1, r_2\}$. Recall that r_1 and r_2 are $\{p_1\}$-maps. The function $\psi(a) = \mathrm{id}$, $\psi(b) = r_2$ and $\psi(c) = r_1$ extends to a morphism from $TR(\widetilde{\Sigma})$ to the monoid M. However, it is not an asynchronous morphism to the atm $U_2[p_1]$ as $\psi(c)$ (being r_1) is not a $\{p_2, p_3\}$-map. ⌟

A fundamental result about asynchronous morphisms is stated in the following lemma. Its proof follows from Proposition 1 and Lemma 5, and can be found in [1].

▶ **Lemma 9.** *Let $T = (\{S_i\}, M)$ be an atm. Further, let $\varphi\colon \Sigma \to M$ be such that, for $a \in \Sigma$, $\varphi(a)$ is an a-map. Then φ can be uniquely extended to an asynchronous morphism from $TR(\widetilde{\Sigma})$ to T.*

▶ **Example 10.** Consider the setup of Example 8. The function $\varphi(a) = r_1$, $\varphi(b) = r_2$ and $\varphi(c) = \mathrm{id}$ extends to an asynchronous morphism from $TR(\widetilde{\Sigma})$ to $U_2[p_1]$. ⌟

Now we extend the notion of trace-language recognition from tm's to atm's via asynchronous morphisms. Let $L \subseteq TR(\widetilde{\Sigma})$ be a trace language. We say that L is recognized by an atm $T = (\{S_i\}, M)$ if there exists an *asynchronous* morphism $\varphi\colon TR(\widetilde{\Sigma}) \to T$, an *initial* element $s_{\mathrm{in}} \in S_{\mathcal{P}}$ and a *final* subset $S_{\mathrm{fin}} \subseteq S_{\mathcal{P}}$ such that

$$L = \{t \in TR(\widetilde{\Sigma}) \mid \varphi(t)(s_{\mathrm{in}}) \in S_{\mathrm{fin}}\}.$$

In the rest of this subsection, we bring out the intimate relationship between asynchronous morphisms and asynchronous automata. We begin with the description of an asynchronous automaton – a model introduced by Zielonka for concurrent computation on traces.

An *asynchronous automaton* A over $\widetilde{\Sigma}$ is a structure $(\{S_i\}_{i\in\mathcal{P}}, \{\delta_a\}_{a\in\Sigma}, s_{\mathrm{in}})$ where

- S_i is a finite non-empty set of local i-states for each process i
- For $a \in \Sigma$, $\delta_a\colon S_a \to S_a$ is a transition function on a-states
- $s_{\mathrm{in}} \in S_{\mathcal{P}}$ is an initial global state

Observe that an a-transition of A reads and updates only the local states of the agents which participate in a. As a result, actions which involve disjoint sets of agents are processed concurrently by A. For $a \in \Sigma$, let $\Delta_a\colon S_{\mathcal{P}} \to S_{\mathcal{P}}$ be the extension of $\delta_a\colon S_a \to S_a$. Clearly, if $(a, b) \in I$ then Δ_a and Δ_b commute. Similar to $\mathcal{P}$-indexed families, we will follow the convention of writing $\{Y_a\}$ to denote the Σ-indexed family $\{Y_a\}_{a\in\Sigma}$.

Now we describe the notion of a run of A on an input trace. A trace run is easiest to define using configurations. Towards this, fix a trace $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$. Recall that (see Section 2.1) $\mathcal{C}_t$ is the set of all configurations of t and $\rightarrow_t \subseteq \mathcal{C}_t \times \Sigma \times \mathcal{C}_t$ is the natural

action based transition relation on configurations. A *trace run* of A over $t \in TR(\widetilde{\Sigma})$ is a map $\rho\colon \mathcal{C}_t \to S_{\mathcal{P}}$ such that $\rho(\emptyset) = s_{\text{in}}$, and for every (c, a, c') in $\to_t$, we have $\Delta_a(\rho(c)) = \rho(c')$. As A is deterministic, t admits a unique trace run; it will be denoted by ρ_t.

Let $L \subseteq TR(\widetilde{\Sigma})$ be a trace language. We say that L is *accepted* by A if there exists a subset $S_{\text{fin}} \subseteq S_{\mathcal{P}}$ of final global states such that $L = \{t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma}) \mid \rho_t(E) \in S_{\text{fin}}\}$.

Our aim is to associate with A, a natural atm T_A and an asynchronous morphism φ_A such that languages accepted by A are precisely the languages recognized via φ_A.

We first describe the *transition* monoid M_A associated to A. It is possible to extend the global transition functions $\{\Delta_a\}$ to arbitrary traces using Proposition 1. For $t \in TR(\widetilde{\Sigma})$, we denote this extended global transition function by $\Delta_t\colon S_{\mathcal{P}} \to S_{\mathcal{P}}$. It is easy to check that, for $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$, $\Delta_t(s_{\text{in}}) = \rho_t(E)$. Further, as expected, for $t, t' \in TR(\widetilde{\Sigma})$, the function composition $\Delta_t\Delta_{t'}$ is identical to $\Delta_{tt'}$. We let M_A be the finite set of functions $\{\Delta_t \mid t \in TR(\widetilde{\Sigma})\}$. Clearly, it is a monoid under the usual composition of functions.

Next, we define the *transition* atm of A to be $T_A = (\{S_i\}, M_A)$ and the natural map $\varphi_A\colon TR(\widetilde{\Sigma}) \to M_A$ sending t to Δ_t. It is clear that φ_A is a morphism of monoids. Furthermore, it is an asynchronous morphism from $TR(\widetilde{\Sigma})$ to T_A; this is because, for $a \in \Sigma$, $\varphi_A(a) = \Delta_a$ is in fact an a-map of the atm T_A. The map φ_A is called the *transition* (asynchronous) morphism of A. Note that, for $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$,

$$\varphi_A(t)(s_{\text{in}}) = \Delta_t(s_{\text{in}}) = \rho_t(E)$$

We refer to the above statement as the *duality* between a run of A and an evaluation of φ_A.

The following lemma summarizes the above discussion for later reference and its proof is immediate.

▶ **Lemma 11.** *Given an asynchronous automaton $A = (\{S_i\}, \{\delta_a\}, s_{in})$ over $\widetilde{\Sigma}$, the transition atm $T_A = (\{S_i\}, M_A)$ and the transition asynchronous morphism $\varphi_A\colon TR(\widetilde{\Sigma}) \to T_A$ are effectively constructible. Moreover, if L is a trace language, then L is accepted by A if and only if it is recognized by T_A via φ_A with s_{in} as the initial state.*

We now provide a form of converse to Lemma 11. Towards this, we fix an atm $T = (\{S_i\}, M)$, a state $s_{\text{in}} \in S_{\mathcal{P}}$ and an asynchronous morphism $\varphi\colon TR(\widetilde{\Sigma}) \to T$. Since φ is an asynchronous morphism, $\varphi(a)$ is an a-map, and is an extension of some $\delta_a\colon S_a \to S_a$ over a-states. We set $A_\varphi = (\{S_i\}, \{\delta_a\}, s_{\text{in}})$ over $\widetilde{\Sigma}$. It turns out that the transition monoid of A_φ is the image of φ, a submonoid of M and the transition morphism of A_φ is the appropriate restriction of φ to this submonoid. The next lemma is easy to prove and we skip its proof.

▶ **Lemma 12.** *Given $T = (\{S_i\}, M)$, $\varphi\colon TR(\widetilde{\Sigma}) \to T$ and $s_{in} \in S_{\mathcal{P}}$, the asynchronous automaton A_φ over $\widetilde{\Sigma}$ is effectively constructible. Moreover, a trace language $L \subseteq TR(\widetilde{\Sigma})$ is recognized by T via φ (with initial state s_{in}) if and only if it is accepted by A_φ.*

3.3 Asynchronous Wreath Product

We begin with the crucial definition of a wreath product of transformation monoids. For sets U and V, we denote the set of all functions from U to V by $\mathcal{F}(U, V)$.

▶ **Definition 13** (Wreath Product). *Let $T_1 = (X, M)$ and $T_2 = (Y, N)$ be two tm's. We define $T = T_1 \wr T_2$ to be the tm $(X \times Y, M \times \mathcal{F}(X, N))$ where, for $m \in M$ and $f \in \mathcal{F}(X, N)$, (m, f) represents the following transformation on $X \times Y$:*

$$\text{for } (x, y) \in X \times Y, \quad (m, f)((x, y)) = (m(x), f(x)(y))$$

The tm T is called the wreath product *of T_1 and T_2. It turns out that, for $(m_1, f_1), (m_2, f_2)$ in $M \times \mathcal{F}(X, N)$, the composition law $(m_1, f_1)(m_2, f_2) = (m, f)$ is such that $m = m_1 m_2$ and for $x \in X$, $f(x) = f_1(x) + f_2(m_1(x))$. Here $+$ denotes the composition operation of N.*

It is a standard fact that the wreath product operation is associative [7]. We now adapt this operation to asynchronous transformation monoids.

▶ **Definition 14.** *Let $T_1 = (\{S_i\}, M)$ and $T_2 = (\{Q_i\}, N)$ be two atm's. We define their asynchronous wreath product, also denoted by $T_1 \wr T_2$, to be the atm $(\{S_i \times Q_i\}, M \times \mathcal{F}(S_\mathcal{P}, N))$. An element $(m, f) \in M \times \mathcal{F}(S_\mathcal{P}, N)$ represents the following global*[1] *transformation on $S_\mathcal{P} \times Q_\mathcal{P}$:*

$$\text{for } (s, q) \in S_\mathcal{P} \times Q_\mathcal{P}, \quad (m, f)((s, q)) = (m(s), f(s)(q))$$

The composition law on $M \times \mathcal{F}(S_\mathcal{P}, N)$ is the same as in Definition 13.

An important observation is that the tm associated with $T_1 \wr T_2$ is the wreath product of the tm's $(S_\mathcal{P}, M)$ and $(Q_\mathcal{P}, N)$ associated with T_1 and T_2 respectively. Sometimes, we will refer to the asynchronous wreath product simply as wreath product. The associativity of the asynchronous wreath product operation follows immediately.

We now present an important combinatorial lemma regarding the "support" of a global transformation in the wreath product. It plays a crucial role later.

▶ **Lemma 15.** *Fix atm's $T_1 = (\{S_i\}, M)$ and $T_2 = (\{Q_i\}, N)$. Let $(m, f) \in M \times \mathcal{F}(S_\mathcal{P}, N)$ represent a P-map in $T_1 \wr T_2$ for some subset $P \subseteq \mathcal{P}$. Then*

- *m is a P-map in T_1.*
- *For every $s \in S_\mathcal{P}$, $f(s)$ is a P-map in T_2. Further, if $s, s' \in S_\mathcal{P}$ are such that $s_P = s'_P$, then $f(s) = f(s')$.*

Proof. Fix $x_0 \in S_{-P}$ and $y_0 \in Q_{-P}$. We define $g_1 \colon S_P \to S_P$ and $g_2 \colon S_P \times Q_P \to Q_P$ by $g_1(x) = [m((x, x_0))]_P$ and $g_2(x, y) = [f((x, x_0))(y, y_0)]_P$. We first show that for all $s \in S_\mathcal{P}, q \in Q_\mathcal{P}$, $(m, f)((s, q)) = ((g_1(s_P), s_{-P}), (g_2(s_P, q_P), q_{-P}))$. Take an arbitrary $(s, q) \in S_\mathcal{P} \times Q_\mathcal{P}$. Then consider the global state $((s_P, x_0), (q_P, y_0))$ sharing the same P-component as (s, q) and the fixed $-P$-component (x_0, y_0). By the wreath product action (see Definition 13), $(m, f)\,(((s_P, x_0), (q_P, y_0))) = (m((s_P, x_0)), f((s_P, x_0))((q_P, y_0)))$. Being a P-map, (m, f) does not change the $-P$-component of any global state. So we have $m((s_P, x_0)) = ([m((s_P, x_0))]_P, x_0)$ and $f((s_P, x_0))((q_P, y_0)) = ([f((s_P, x_0))((q_P, y_0))]_P, y_0)$.

Let $(m, f)((s, q)) = (s', q')$. Since (m, f) is a P-map and the two global states (s, q) and $((s_P, x_0), (q_P, y_0))$ share the same P-component, by Lemma 4, $s'_P = [m((s_P, x_0))]_P$ and $q'_P = [f((s_P, x_0))((q_P, y_0))]_P$. Further, $s'_{-P} = s_{-P}$ and $q'_{-P} = q_{-P}$. Using the definitions of g_1 and g_2, we immediately see that $(m, f)((s, q)) = ((g_1(s_P), s_{-P}), (g_2(s_P, q_P), q_{-P}))$. However, by the wreath product action, $(m, f)((s, q)) = (m(s), f(s)(q))$. Comparing this with the previous expression, we have $m(s) = (g_1(s_P), s_{-P})$ and $f(s)(q) = (g_2(s_P, q_P), q_{-P})$. The result now follows from Lemma 4. ◀

[1] a global state (resp. P-state) of $T_1 \wr T_2$ is canonically identified with an element of $S_\mathcal{P} \times Q_\mathcal{P}$ (resp. $S_P \times Q_P$)

3.4 Local Wreath Product Principle

Let $A = (\{S_i\}, \{\delta_a\}, s_{in})$ be an asynchronous automaton over $\widetilde{\Sigma}$. Based on A and $\widetilde{\Sigma}$, we define the alphabet $\Sigma^{\|_S} = \{(a, s_a) \mid a \in \Sigma, s \in S_{\mathcal{P}}\}$ where a letter a in Σ is extended with local a-state information of A. This can naturally be viewed as a distributed alphabet $\widetilde{\Sigma^{\|_S}}$ where $\forall a \in \Sigma, \forall s \in S_{\mathcal{P}}$, $(a, s_a) \in \Sigma_i^{\|_S}$ if and only if $a \in \Sigma_i$. Then A induces the following transducer over traces.

▶ **Definition 16** (Local Asynchronous Transducer). *Let $\chi_A \colon TR(\widetilde{\Sigma}) \to TR(\widetilde{\Sigma^{\|_S}})$ be defined as follows. If $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$, then $\chi_A(t) = t'$ where $t' = (E, \leq, \mu) \in TR(\widetilde{\Sigma^{\|_S}})$ with the labelling $\mu \colon E \to \Sigma^{\|_S}$ defined by:*

$$\forall e \in E, \mu(e) = (a, s_a) \text{ where } a = \lambda(e) \text{ and } s = \rho_t(\downarrow e \setminus \{e\})$$

(recall that ρ_t is the unique trace run of A over t). We call χ_A the local asynchronous transducer of A.

▶ **Example 17.** Let χ be the local asynchronous transducer associated to A_φ where φ is as in Example 10. Figure 1 shows the run of A_φ on a trace $t \in TR(\widetilde{\Sigma})$ (by showing local process states before and after each event), and the resulting trace $\chi(t) \in TR(\widetilde{\Sigma^{\|_S}})$. ⌟

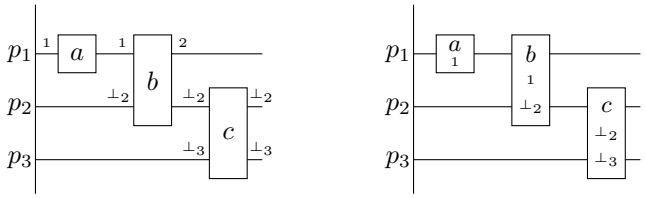


Figure 1 Local asynchronous transducer output on a trace; $S_{p_2} = \{\perp_2\}, S_{p_3} = \{\perp_3\}$.

Note that, in general, χ_A is not a morphism of monoids. The following lemma is a straightforward consequence of the definition of χ_A and the duality between trace runs of A and evaluations of the asynchronous asynchronous morphism φ_A.

▶ **Lemma 18.** *Let $t \in TR(\widetilde{\Sigma})$ with factorization $t = t'a$ (where $a \in \widetilde{\Sigma}$), and $s = \varphi_A(t')(s_{in})$. Then the trace $\chi_A(t) \in TR(\widetilde{\Sigma^{\|_S}})$ factors as $\chi_A(t) = \chi_A(t')(a, s_a)$.*

▶ **Theorem 19.** *Let A be an asynchronous automaton over $\widetilde{\Sigma}$ and χ_A be the corresponding local asynchronous transducer. If $L \subseteq TR(\widetilde{\Sigma^{\|_S}})$ is recognized by an atm T, then $\chi_A^{-1}(L)$ is recognized by the atm $T_A \wr T$.*

Proof. Let $\psi \colon TR(\widetilde{\Sigma^{\|_S}}) \to T = (\{Q_i\}, N)$ be an asynchronous morphism, which recognizes L with $q_{\text{in}} \in Q_{\mathcal{P}}$ as the initial global state, and $Q_{\text{fin}} \subseteq Q_{\mathcal{P}}$ as the set of final global states. Then $L = \{t \in TR(\widetilde{\Sigma^{\|_S}}) \mid \psi(t)(q_{\text{in}}) \in Q_{\text{fin}}\}$. Note that, for $(a, s_a) \in \Sigma^{\|_S}$, $\psi((a, s_a))$ is an a-map (that is, an extension of a map from Q_a to Q_a; recall that $\text{loc}((a, s_a)) = \text{loc}(a)$).

For $a \in \Sigma$, we set $\eta(a) = (\varphi_A(a), f_a)$ where $f_a \colon S_{\mathcal{P}} \to N$ is defined by $f_a(s) = \psi((a, s_a))$. It is easy to check that $\eta(a)$ is an a-map (that is, an extension of a map from $S_a \times Q_a$ to $S_a \times Q_a$). By Lemma 9, this uniquely defines an asynchronous morphism $\eta \colon TR(\widetilde{\Sigma}) \to T_A \wr T$.

Let $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$. We write $\eta(t) = (\pi_1(t), \pi_2(t))$. It follows from the definition of wreath product that $\pi_1(t) = \varphi_A(t)$. Now we claim that $\pi_2(t)(s_{\text{in}}) = \psi(\chi_A(t))$. We prove this by induction on the cardinality of E. Suppose $t = t'a$. Then $\eta(t) = \eta(t')\eta(a)$.

As a result, we have $(\pi_1(t), \pi_2(t)) = (\pi_1(t'), \pi_2(t'))(\pi_1(a), \pi_2(a))$. Therefore, for $s \in S_{\mathcal{P}}$, $\pi_2(t)(s) = \pi_2(t')(s) + \pi_2(a)(\pi_1(t')(s))$. In particular, it holds with $s = s_{\text{in}}$. Recall that $\pi_1(t') = \varphi_A(t')$. Also, by induction, $\pi_2(t')(s_{\text{in}}) = \psi(\chi_A(t'))$. Hence, with $s = \varphi_A(t')(s_{\text{in}})$,

$$\begin{aligned}\pi_2(t)(s_{\text{in}}) &= \pi_2(t')(s_{\text{in}}) + \pi_2(a)(\pi_1(t')(s_{\text{in}}))\\ &= \psi(\chi_A(t')) + \pi_2(a)(s)\\ &= \psi(\chi_A(t')) + \psi((a, s_a))\\ &= \psi(\chi_A(t')(a, s_a))\\ &= \psi(\chi_A(t))\end{aligned}$$

The last equality follows from Lemma 18. So, $t \in \chi_A^{-1}(L)$ if and only if $\chi_A(t) \in L$ if and only if $\psi(\chi_A(t))(q_{\text{in}}) \in Q_{\text{fin}}$ if and only if $\pi_2(t)(s_{\text{in}})(q_{\text{in}}) \in Q_{\text{fin}}$ if and only if $\eta(t)(s_{\text{in}}, q_{\text{in}}) \in S_{\mathcal{P}} \times Q_{\text{fin}}$. This shows that η recognizes $\chi_A^{-1}(L)$ with $(s_{\text{in}}, q_{\text{in}}) \in S_{\mathcal{P}} \times Q_{\mathcal{P}}$ as the initial global state, and $S_{\mathcal{P}} \times Q_{\text{fin}} \subseteq S_{\mathcal{P}} \times Q_{\mathcal{P}}$ as the set of final global states. ◀

Now we focus our attention on what is usually termed as the wreath product principle.

▶ **Theorem 20** (Local Wreath Product Principle). *Let T_1 and T_2 be atm's and let $L \subseteq TR(\widetilde{\Sigma})$ be a trace language recognized by an asynchronous morphism $\eta\colon TR(\widetilde{\Sigma}) \to T_1 \wr T_2$, with initial global state (s_{in}, q_{in}). For each $a \in \Sigma$, let $\eta(a) = (m_a, f_a)$. Then $\varphi\colon TR(\widetilde{\Sigma}) \to T_1$, defined by $\varphi(a) = m_a$, is an asynchronous morphism. Finally, let $A = A_\varphi$ be the asynchronous automaton associated to φ and s_{in}, and let χ_A be the corresponding local asynchronous transducer. Then L is a finite union of languages of the form $U \cap \chi_A^{-1}(V)$, where $U \subseteq TR(\tilde{\Sigma})$ is recognized by T_1, and $V \subseteq TR(\widetilde{\Sigma^{\|s}})$ is recognized by T_2.*

Proof. We write $T_1 = (\{S_i\}, M)$ and $T_2 = (\{Q_i\}, N)$. Consider $a \in \Sigma$ and the a-map $\eta(a) = (m_a, f_a) \in M \times \mathcal{F}(S_{\mathcal{P}}, N)$. This means that $\eta(a)$ is an extension of a map from $S_a \times Q_a$ to $S_a \times Q_a$. By Lemma 15, $m_a \in M$ is an a-map (of T_1) and $f_a\colon S_{\mathcal{P}} \to N$ is such that, for $s \in S_{\mathcal{P}}$, $f_a(s) \in N$ is an a-map (of T_2) and it depends only on s_a. In particular, $f_a\colon S_{\mathcal{P}} \to N$ may be viewed as $f_a\colon S_a \to N$. Below we will use f_a in this sense.

Now we define an asynchronous morphism $\psi\colon TR(\widetilde{\Sigma^{\|s}}) \to T_2$ as follows: $\psi((a, s_a)) = f_a(s_a)$. Note that, by Lemma 9, ψ is indeed an asynchronous morphism as $f_a(s_a)$ is an a-map. Further, as m_a is an a-map, $\varphi\colon TR(\widetilde{\Sigma}) \to T_1$, defined by $\varphi(a) = m_a$, also extends to an asynchronous morphism.

Our aim is to express L in terms of languages recognized by T_1 and T_2. It suffices to show the result when L is recognized with a single final global state, say $(s_{\text{fin}}, q_{\text{fin}})$. Then $L = \{t \in TR(\widetilde{\Sigma}) \mid \eta(t)((s_{\text{in}}, q_{\text{in}})) = (s_{\text{fin}}, q_{\text{fin}})\}$.

For $t \in TR(\widetilde{\Sigma})$, we write $\eta(t) = (\pi_1(t), \pi_2(t))$. It follows from the definition of φ that $\varphi(t) = \pi_1(t)$. Hence, we can alternatively write L as

$$L = \{t \in TR(\widetilde{\Sigma}) \mid \varphi(t)(s_{\text{in}}) = s_{\text{fin}} \text{ and } \pi_2(t)(s_{\text{in}})(q_{\text{in}}) = q_{\text{fin}}\}$$

Let $U = \{t \in TR(\widetilde{\Sigma}) \mid \varphi(t)(s_{\text{in}}) = s_{\text{fin}}\}$. Then, with $W = \{t \in TR(\widetilde{\Sigma}) \mid \pi_2(t)(s_{\text{in}})(q_{\text{in}}) = q_{\text{fin}}\}$, $L = U \cap W$. By using essentially the same ideas as in the proof of Theorem 19, we can show that $\pi_2(t)(s_{\text{in}}) = \psi(\chi_A(t))$. Therefore, $W = \{t \in TR(\widetilde{\Sigma}) \mid \psi(\chi_A(t))(q_{\text{in}}) = q_{\text{fin}}\}$.

It follows that, with $V = \{t' \in TR(\widetilde{\Sigma^{\|s}}) \mid \psi(t')(q_{\text{in}}) = q_{\text{fin}}\}$, $W = \chi_A^{-1}(V)$. Clearly, U is recognized by the atm T_1 (via φ), V is recognized by the atm T_2 (via ψ) and $L = U \cap \chi_A^{-1}(V)$. This completes the proof. ◀

4 Towards a Decomposition Result

In this section, we use the algebraic framework developed so far to propose an analogue of the fundamental Krohn-Rhodes decomposition theorem over traces. We first recall the Krohn-Rhodes theorem in the purely algebraic setting of transformation monoids. We briefly explain how it is used to analyze/decompose morphisms from the free monoid and point out some difficulties that arise when we consider morphisms from the trace monoid.

Let M and N be monoids. We say that M divides N (in notation, $M \prec N$) if M is a homomorphic image of some submonoid of N. This notion can be extended to transformation monoids. Let (X, M) and (Y, N) be two tm's. We say that (X, M) *divides* (Y, N), denoted $(X, M) \prec (Y, N)$, if there exists a pair of mappings (f, φ) where $f\colon Y \to X$ is a surjective function and $\varphi\colon N' \to M$ is a surjective morphism from a submonoid N' of N, such that $\varphi(n)(f(y)) = f(n(y))$ for all $n \in N'$ and all $y \in Y$.

Recall that $U_2 = (\{1, 2\}, \{\mathrm{id}, r_1, r_2\})$ denotes the *reset* transformation monoid on two elements. Along with it, the following class of transformation monoids plays an important role in the Krohn Rhodes theorem.

▶ **Example 21.** Let G be a group. Then (G, G) is a tm where the monoid element g *represents* the transformation $m_g\colon G \to G$ of the set G, which is the right multiplication by g. In other words, for $h \in G, m_g(h) = hg$. ⌟

We are now in a position to state the Krohn-Rhodes theorem [11]. See [20] for a classical proof of the theorem, and [5] for a modern proof.

▶ **Theorem 22** (Krohn-Rhodes Theorem). *Every finite transformation monoid $T = (X, M)$ divides a wreath product of the form $T_1 \wr \ldots \wr T_n$ where each factor T_i is either U_2 or is of the form (G, G) for some non-trivial simple group G dividing M.*

Henceforth, we will be dealing with only finite tm's and sometimes we will omit the qualifier "finite". Now we turn our attention to the use of this decomposition theorem for analysing word languages recognized by morphisms from the free monoid.

▶ **Definition 23.** *Let $\varphi\colon \Sigma^* \to T = (X, M)$ be a morphism. Further, let $\psi\colon \Sigma^* \to T' = (Y, N)$ be another morphism. We say that ψ* simulates *φ if there exists a surjective function $f\colon Y \to X$ such that, for all $a \in \Sigma$ and all $y \in Y$, $f(\psi(a)(y)) = \varphi(a)(f(y))$.*

$$\begin{array}{ccc} Y & \xrightarrow{\psi(a)} & Y \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f} \\ X & \xrightarrow[\varphi(a)]{} & X \end{array}$$

Figure 2 Visual illustration of condition $f(\psi(a)(y)) = \varphi(a)(f(y))$ in Definition 23.

Observe that if ψ simulates φ then a language recognized by φ is also recognized by ψ.

▶ **Proposition 24.** *Let $\varphi\colon \Sigma^* \to T = (X, M)$ be a morphism. Then there exists a morphism $\psi\colon \Sigma^* \to T'$ which simulates φ such that the tm T' is of the form $T_1 \wr \ldots \wr T_n$ where each factor T_i is either U_2 or (G, G) for some non-trivial simple group G dividing M.*

Proof. Given T, we get $T' = T_1 \wr \ldots \wr T_n = (Y, N)$ by the Krohn-Rhodes theorem. Since $T \prec T'$, there exists a pair of mappings (f, θ) where $f\colon Y \to X$ is a surjective function and $\theta\colon N' \to M$ is a surjective morphism from a submonoid N' of N, such that $\theta(n)(f(y)) = f(n(y))$ for

all $n \in N'$ and all $y \in Y$. Construct $\psi\colon \Sigma \to N$ by mapping $\psi(a)$, for each a in Σ, to an arbitrary element in $\theta^{-1}(\varphi(a))$. Thanks to the fact that Σ^* is a free monoid, ψ uniquely extends to a morphism $\psi\colon \Sigma^* \to T'$. It is easily checked that ψ simulates φ. ◀

Combined with the wreath product principle, the above proposition provides a powerful inductive tool to prove many non-trivial results in the theory of finite words. See [14, 3].

Motivated by these applications, we look for an analogue of the above proposition for the setting of traces. We now point to some problems that arise if one tries to naively lift the Krohn-Rhodes theorem to the setting of traces. The first problem is that, unlike in the word scenario, division does not imply simulation of morphisms from the *trace monoid*. By simulation of morphisms from the trace monoid, we simply mean an obvious adaptation of the Definition 23 to the morphisms from the trace monoid. See the extended version [1] for an example of the problem of the first kind. The second problem is that even if there is a morphism from $TR(\widetilde{\Sigma})$ to a wreath product of tm's, in general it does not induce morphisms from trace monoids to the individual tm's beyond the first one. This is primarily because the output of the *sequential* transducer associated with the first tm is *not* a trace.

▶ **Example 25.** Assume the DFA in Figure 3 represents the induced morphism to the first tm in a wreath product chain. The figure below shows the outputs of the sequential transducer associated with this DFA on three different linearizations of a single input trace. These outputs have different sets of letters and can not constitute a single trace. ⌟

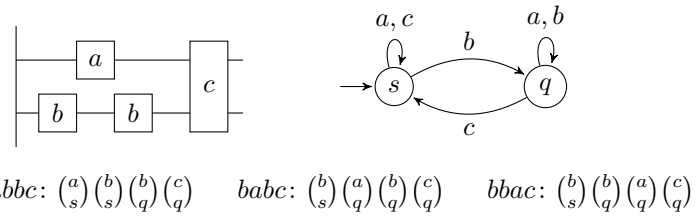


Figure 3 Sequential transducer outputs for all linearizations of a trace.

Prior work in [8] devised a wreath product principle for traces, but it uses non-trace structures to circumvent the second problem, thus limiting its applicability.

As seen in the previous section, the new algebraic framework of asynchronous structures supports true concurrency and is well suited to reason about trace languages. Most importantly, an asynchronous morphism to a wreath product chain gives rise to asynchronous morphisms to individual atm's of the chain (see the proof of Theorem 20 for an illustration). This can be seen as a resolution of the second problem mentioned above.

Going ahead, we extend the notion of simulation to the case when the "simulating" morphism is an asynchronous morphism to an atm.

▶ **Definition 26.** *Let $\varphi\colon TR(\widetilde{\Sigma}) \to T = (X, M)$ be a morphism to a tm. Further, let $T' = (\{S_i\}, N)$ be an atm and $\psi\colon TR(\widetilde{\Sigma}) \to T'$ be an asynchronous morphism. We say that ψ is an asynchronous simulation of φ (or simply ψ simulates φ) if there exists a surjective function $f\colon S_{\mathcal{P}} \to X$ such that, for all $a \in \Sigma$ and all $s \in S_{\mathcal{P}}$, $f(\psi(a)(s)) = \varphi(a)(f(s))$.*

The fundamental theorem of Zielonka [21] states that every recognizable language is accepted by some asynchronous automaton. See [15] for another proof of the theorem. From the viewpoint of our algebraic setup and the previous definition, it guarantees the existence of a simulating asynchronous morphism.

▶ **Theorem 27** (Zielonka's Theorem). *Let $\varphi\colon TR(\widetilde{\Sigma}) \to T$ be a morphism to a finite tm. There exists an asynchronous morphism $\psi\colon TR(\widetilde{\Sigma}) \to T'$, to a finite atm, which simulates φ.*

Recall that the atm $U_2[\ell]$, defined in Example 6, is a natural extension of the tm U_2 to the process ℓ. In a similar vein, for a group G, the atm $G[\ell]$ denotes the natural extension of the tm (G, G) from Example 21 to the process ℓ. We will use a similar notation to extend a tm to an atm localized to a particular process.

Now we formulate the following decomposition question:

▶ **Question 1.** *Let $\varphi\colon TR(\widetilde{\Sigma}) \to (X, M)$ be a morphism to a finite tm. Does there exist an asynchronous morphism $\psi\colon TR(\widetilde{\Sigma}) \to T'$ to a finite atm, such that ψ simulates φ, and the atm T' is of the form $T_1 \wr \ldots \wr T_n$ where each factor T_i is, for some $\ell \in \mathcal{P}$, either the atm $U_2[\ell]$ or is of the form $G[\ell]$ for some non-trivial simple group G dividing M?*

In view of our discussion so far, it is clear that the above question asks for a simultaneous generalization of the Krohn-Rhodes theorem for the setting of words (that is, Proposition 24), and Zielonka's theorem for the setting of traces (that is, Theorem 27). Question 1 in general remains open. However we answer it positively in a particular case, namely that of acyclic architectures, which is general enough to include the common client-server settings.

▶ **Definition 28.** *Let $\widetilde{\Sigma} = \{\Sigma_i\}_{i\in\mathcal{P}}$ be a distributed alphabet. Then its communication graph is $G = (\mathcal{P}, E)$ where $E = \{(i, j) \in \mathcal{P} \times \mathcal{P} \mid i \neq j \text{ and } \Sigma_i \cap \Sigma_j \neq \emptyset\}$. If the communication graph is acyclic, then the distributed alphabet is called an acyclic architecture.*

Observe that if $\widetilde{\Sigma}$ is an acyclic architecture, then no action is shared by more than two processes. The work [10] provides a simpler proof of Zielonka's theorem in this case.

▶ **Theorem 29.** *If $\widetilde{\Sigma}$ is an acyclic architecture, then Question 1 admits a positive answer.*

5 Local and Global Cascade Products

In this section, we introduce distributed automata-theoretic operations called local and global cascade products.

5.1 Local Cascade Product

As seen before, asynchronous morphisms are the algebraic counterparts of asynchronous automata. It turns out that an asynchronous morphism into a wreath product of atm's corresponds to the "local cascade product" of asynchronous automata. See [1] for details. Here we simply define the local cascade product of two asynchronous automata.

▶ **Definition 30.** *Let $A_1 = (\{S_i\}, \{\delta_a\}, s_{in})$ over $\widetilde{\Sigma}$, and $A_2 = (\{Q_i\}, \{\delta_{(a,s_a)}\}, q_{in})$ over $\widetilde{\Sigma^{\|s}}$. We define the local cascade product of A_1 and A_2 to be the asynchronous automaton $A_1 \circ_\ell A_2 = (\{S_i \times Q_i\}, \{\Delta_a\}, (s_{in}, q_{in}))$ over $\widetilde{\Sigma}$, where, for $a \in \Sigma$ and $(s_a, q_a) \in S_a \times Q_a$, $\Delta_a((s_a, q_a)) = (\delta_a(s_a), \delta_{(a,s_a)}(q_a))$.*

The operational working of $A = A_1 \circ_\ell A_2$ can be understood in terms of A_1 and A_2 using the local asynchronous transducer $\chi_{A_1}\colon TR(\widetilde{\Sigma}) \to TR(\widetilde{\Sigma^{\|s}})$ (associated with A_1) as follows: for an input trace $t \in TR(\widetilde{\Sigma})$, the run of A on t ends in global state (s, q) if and only if the run of A_1 on t ends in global state s and the run of A_2 on $\chi_{A_1}(t)$ ends in global state q. This *operational cascade* of A_1 followed by A_2 is summarized in the right part of the Figure 4 and is the essence of the local wreath product principle. Further, it is not difficult to check that the local cascade product is associative and $\chi_{A_1 \circ_\ell A_2}(t) = \chi_{A_2}(\chi_{A_1}(t))$ for all $t \in TR(\widetilde{\Sigma})$.

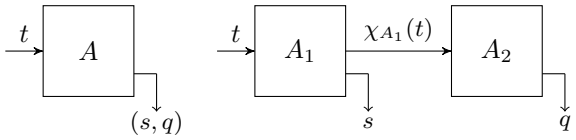


Figure 4 Operational view of local cascade product.

5.2 Global Asynchronous Transducer and its Local Implementation

Let $A = (\{S_i\}, \{\delta_a\}, s_{\text{in}})$ be an asynchronous automaton over $\widetilde{\Sigma}$. Recall that the local asynchronous transducer χ_A preserves the underlying set of events and, at an event, simply records the previous local states of the processes *participating* in that event.

Now we introduce a natural variant of χ_A which is called the *global asynchronous transducer*. In this variant, at an event, we record the *maximal/best global state* that causally precedes the current event. This is the best global state that the processes participating in the current event are (collectively) aware of. It is important to note that the global and local asynchronous transducers coincide in the sequential setting.

We first define the alphabet $\Sigma^{S_{\mathcal{P}}} = \Sigma \times S_{\mathcal{P}}$ where each letter in Σ is extended with global state information of A. This can naturally be viewed as a distributed alphabet $\widetilde{\Sigma^{S_{\mathcal{P}}}}$ where for all $a \in \Sigma$ and $s \in S_{\mathcal{P}}$, we have $(a, s) \in \Sigma_i^{S_{\mathcal{P}}}$ if and only if $a \in \Sigma_i$.

▶ **Definition 31** (Global Asynchronous Transducer). *Let A be an asynchronous automaton over $\widetilde{\Sigma}$. The global asynchronous transducer of A is the map $\theta_A \colon TR(\widetilde{\Sigma}) \to TR(\widetilde{\Sigma^{S_{\mathcal{P}}}})$ defined as follows. If $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$, then $\theta_A(t) = (E, \leq, \mu) \in TR(\widetilde{\Sigma^{S_{\mathcal{P}}}})$ with the labelling $\mu \colon E \to \Sigma \times S_{\mathcal{P}}$ defined by:*

$$\forall e \in E,\ \mu(e) = (a, s) \text{ where } a = \lambda(e) \text{ and } s = \rho_t(\downarrow e \setminus \{e\})$$

▶ **Example 32.** For t and A_φ from Example 17, Figure 5 shows its global asynchronous transducer output $\theta(t)$. Note the difference from Figure 1. For example, here the only p_3-event has process p_1 state 2 in its label (which is the best process p_1 state in its causal past) even though processes p_1 and p_3 never interact directly. ⌟

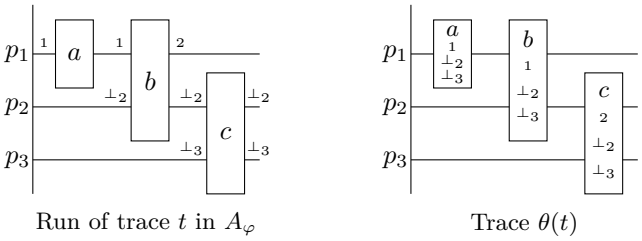


Figure 5 Global asynchronous transducer output on a trace.

It is possible, albeit non-trivial, to give a uniform translation from the automaton A to another automaton $\mathcal{G}(A)$ such that the global asynchronous transducer of A is realized by the local asynchronous transducer of $\mathcal{G}(A)$. It turns out that one must make crucial use of the latest information that the agents have about each other when defining the automaton $\mathcal{G}(A)$. It has been shown in [16] that this information can be kept track of by a deterministic asynchronous automaton. See [1] for more details.

5.3 Global Cascade Product

Now we are ready to define a cascade model which uses the global asynchronous transducer.

▶ **Definition 33** (Operational Global Cascade Product). *Let $A_1 = (\{S_i\}, \{\delta_a\}, s_{in})$ be an asynchronous automaton over $\widetilde{\Sigma}$, and $A_2 = (\{Q_i\}, \{\delta_{(a,s)}\}, q_{in})$ be an asynchronous automaton over $\widetilde{\Sigma^{S_\mathcal{P}}}$. Then their* operational *global cascade product, denoted by $A_1 \circ_g A_2$, is a cascade model where, for any input trace $t \in TR(\widetilde{\Sigma})$, A_1 runs on t (and "outputs" $\theta_{A_1}(t)$) and A_2 runs on $\theta_{A_1}(t)$. See Figure 6.*

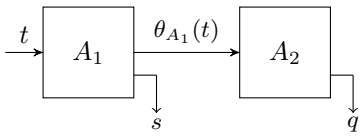


Figure 6 Operational view of global cascade product.

Note that $A_1 \circ_g A_2$ is not, a priori, an asynchronous automaton, but in view of the discussion in the preceeding subsection, it is simulated by the asynchronous automaton $\mathcal{G}(A_1) \circ_\ell A_2$.

For simplicity, we view $A_1 \circ_g A_2$ as an automaton with $S_\mathcal{P} \times Q_\mathcal{P}$ as its global states, and extend the notions of run, acceptance etc. to it in a natural way (see [1]). Henceforth, we refer to the operational global cascade product as the simply global cascade product. It turns out that the global cascade product is associative in a natural sense. See [1] for more details. Thanks to this, we can also talk about the global cascade product of a *sequence* of asynchronous automata.

The following *global cascade product principle* is an easy consequence of the definitions.

▶ **Theorem 34** (Global Cascade Product Principle). *Let A (resp. B) be a global cascade product over $\widetilde{\Sigma}$ (resp. $\widetilde{\Sigma^{S_\mathcal{P}}}$), where $S_\mathcal{P}$ is the set of global states of A. Then any language $L \subseteq TR(\widetilde{\Sigma})$ accepted by $A \circ_g B$ is a finite union of languages of the form $U \cap \theta_A^{-1}(V)$ where $U \subseteq TR(\widetilde{\Sigma})$ is accepted by A, and $V \subseteq TR(\widetilde{\Sigma^{S_\mathcal{P}}})$ is accepted by B.*

6 Temporal Logics, Aperiodic Trace Languages & Cascade Products

An automata-theoretic consequence of Theorem 29 is that any aperiodic trace language (that is, a trace language recognized by an aperiodic monoid) over an acyclic architecture is accepted by a local cascade product of localized two-state reset automata. We call these automata $U_2[\ell]$ as well. In this section, we generalize this result to any distributed alphabet, but using global cascade product of $U_2[\ell]$s.

Our proof uses a process-based past local temporal logic (over traces) called $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ that exactly defines aperiodic trace languages. This expressive completeness property of $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ is an easy consequence of a non-trivial result from [4], where the future version of a similar local temporal logic is shown to coincide with first-order logic definable, equivalently, aperiodic trace languages. The syntax of $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ is as follows.

$$\text{Event formula } \alpha = a \mid \neg\alpha \mid \alpha \vee \alpha \mid \mathsf{Y}_i\,\alpha \mid \alpha\,\mathsf{S}_i\,\alpha \qquad a \in \Sigma, i \in \mathcal{P}$$

$$\text{Trace formula } \beta = \exists_i \alpha \mid \neg\beta \mid \beta \vee \beta$$

The semantics of the logic is given below. Each event formula is evaluated at an event of a trace. Let $t = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$ be a trace with $e \in E$. For any event x in t and $i \in \mathcal{P}$, we denote by x_i the unique maximal event of $(\downarrow x \setminus \{x\}) \cap E_i$, if it exists.

$t, e \models a$	if $\lambda(e) = a$
$t, e \models \neg\alpha$	if $t, e \not\models \alpha$
$t, e \models \alpha \vee \beta$	if $t, e \models \alpha$ or $t, e \models \beta$
$t, e \models \mathsf{Y}_i\, \alpha$	if e_i exists, and $t, e_i \models \alpha$
$t, e \models \alpha\, \mathsf{S}_i\, \alpha'$	if $e \in E_i$ and $\exists f \in E_i$ such that $f < e$ and $t, f \models \alpha'$ and $\forall g \in E_i \;\; f < g < e \Rightarrow t, g \models \alpha$

Note that the since operator is a strict version. $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ trace formulas are evaluated for traces, with the following semantics.

$t \models \exists_i \alpha$	if there exists a maximal i-event e in t such that $t, e \models \alpha$

The semantics of the boolean combinations of trace formulas are obvious. Any $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ trace formula β over $\widetilde{\Sigma}$ defines the trace language $L_\beta = \{t \in TR(\widetilde{\Sigma}) \mid t \models \beta\}$. The following theorem gives a global cascade product characterization of $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ definable languages.

▶ **Theorem 35.** *A trace language is defined by a* $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ *formula if and only if it is accepted by a global cascade product of* $U_2[\ell]$*.*

By the expressive completeness of $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ from [4], this gives a new characterization of aperiodic or equivalently, first-order logic definable trace languages in terms of global cascade products of localized two state asynchronous reset automata. It is in the spirit of the classical Krohn-Rhodes theorem for aperiodic word languages.

We now give a temporal logic characterization of local cascade product of $U_2[\ell]$. The local temporal logic $\mathsf{LocTL}[\mathsf{S}_i]$ is simply the fragment of $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$ where Y_i is disallowed. The semantics is inherited. It is unknown whether the logic $\mathsf{LocTL}[\mathsf{S}_i]$ is as expressive as $\mathsf{LocTL}[\mathsf{Y}_i, \mathsf{S}_i]$.

▶ **Theorem 36.** *A trace language is defined by a* $\mathsf{LocTL}[\mathsf{S}_i]$ *formula if and only if it is recognized by a local cascade product of* $U_2[\ell]$*.*

Note that if our postulated decomposition (see Question 1) were true, it would imply that $\mathsf{LocTL}[\mathsf{S}_i]$ is expressively complete, which would be a stronger temporal logic characterization for aperiodic trace languages than what is currently known. In particular, by Theorem 29, $\mathsf{LocTL}[\mathsf{S}_i]$ is expressively complete over tree architecture. And this holds true for any distributed alphabet where Question 1 admits a positive answer.

7 Conclusion

We have presented an algebraic framework equipped with wreath products and proved a wreath product principle which is well suited for the analysis of trace languages. Building on this framework, we have postulated a natural decomposition theorem which has been proved for the case of acyclic architectures. This special case already provides an interesting generalization of the Krohn-Rhodes theorem. It simultaneously proves Zielonka's theorem for acyclic architectures.

The wreath product operation in the new framework, when viewed in terms of automata, manifests itself in the form of a local cascade product of asynchronous automata. We have also proposed global cascade products of asynchronous automata and applied them to

arrive at a novel decomposition of aperiodic trace languages. This is a non-trivial and truly concurrent generalization of the cascade decomposition of aperiodic word languages using two-state reset automata.

References

1 Bharat Adsul, Paul Gastin, Saptarshi Sarkar, and Pascal Weil. Wreath/cascade products and related decomposition results for the concurrent setting of mazurkiewicz traces (extended version), 2020. arXiv:2007.07940.

2 Bharat Adsul and Milind A. Sohoni. Asynchronous automata-theoretic characterization of aperiodic trace languages. In *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*, pages 84–96, 2004. doi:10.1007/978-3-540-30538-5_8.

3 Joëlle Cohen, Dominique Perrin, and Jean-Eric Pin. On the expressive power of temporal logic. *Journal of Computer and System Sciences*, 46(3):271–294, 1993.

4 Volker Diekert and Paul Gastin. Pure future local temporal logics are expressively complete for mazurkiewicz traces. *Inf. Comput.*, 204(11):1597–1619, 2006. doi:10.1016/j.ic.2006.07.002.

5 Volker Diekert, Manfred Kufleitner, and Benjamin Steinberg. The Krohn-Rhodes theorem and local divisors. *Fundam. Inform.*, 116(1-4):65–77, 2012. doi:10.3233/FI-2012-669.

6 Volker Diekert and Grzegorz Rozenberg. *The Book of Traces*. World Scientific Publishing Co., Inc., USA, 1995.

7 Samuel Eilenberg. *Automata, Languages and Machines*, volume B. Academic Press, 1976.

8 Giovanna Guaiana, Raphaël Meyer, Antoine Petit, and Pascal Weil. An extension of the wreath product principle for finite Mazurkiewicz traces. *Information Processing Letters*, 67(6):277–282, 1998. doi:10.1016/S0020-0190(98)00123-9.

9 Hans Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, UCLA, 1968.

10 Siddharth Krishna and Anca Muscholl. A quadratic construction for Zielonka automata with acyclic communication structure. *Theor. Comput. Sci.*, 503(C):109–114, September 2013.

11 Kenneth Krohn and John Rhodes. Algebraic theory of machines I. prime decomposition theorem for finite semigroups and machines. *Transactions of The American Mathematical Society*, 116, April 1965. doi:10.2307/1994127.

12 Antoni Mazurkiewicz. Concurrent program schemes and their interpretations. *DAIMI Report Series*, 6(78), 1977. doi:10.7146/dpb.v6i78.7691.

13 Robert McNaughton and Seymour A. Papert. *Counter-Free Automata*. M.I.T. Research Monograph Nr 65. The MIT Press, 1971.

14 Albert R. Meyer. A note on star-free events. *J. ACM*, 16(2):220–225, April 1969. doi:10.1145/321510.321513.

15 Madhavan Mukund. Automata on distributed alphabets. In Deepak D'Souza and Priti Shankar, editors, *Modern applications of automata theory*, pages 257–288. World Scientific, 2012.

16 Madhavan Mukund and Milind A. Sohoni. Keeping track of the latest gossip in a distributed system. *Distributed Computing*, 10(3):137–148, 1997. doi:10.1007/s004460050031.

17 Dominique Perrin and Jean-Eric Pin. *Infinite words - automata, semigroups, logic and games*, volume 141 of *Pure and applied mathematics series*. Elsevier Morgan Kaufmann, 2004.

18 Jean-Éric Pin. Mathematical foundations of automata theory, 2019. URL: https://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf.

19 M.P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965. doi:10.1016/S0019-9958(65)90108-7.

20 Howard Straubing. *Finite automata, formal logic, and circuit complexity*. Birkhaüser Verlag, Basel, Switzerland, 1994.

21 Wieslaw Zielonka. Notes on finite asynchronous automata. *RAIRO-Theoretical Informatics and Applications*, 21(2):99–135, 1987.

Partially Observable Concurrent Kleene Algebra

Jana Wagemaker
Radboud University Nijmegen, The Netherlands
j.wagemaker@cs.ru.nl

Paul Brunet
University College London, UK

Simon Docherty
University College London, UK

Tobias Kappé
University College London, UK

Jurriaan Rot
Radboud University Nijmegen, The Netherlands

Alexandra Silva
University College London, UK

Abstract

We introduce partially observable concurrent Kleene algebra (POCKA), an algebraic framework to reason about concurrent programs with variables as well as control structures, such as conditionals and loops, that depend on those variables. We illustrate the use of POCKA through concrete examples. We prove that POCKA is a sound and complete axiomatisation of a model of partial observations, and show the semantics passes an important check for sequential consistency.

2012 ACM Subject Classification Theory of computation → Semantics and reasoning; Theory of computation → Concurrency; Theory of computation → Formal languages and automata theory

Keywords and phrases Concurrent Kleene algebra, Kleene algebra with tests, observations, axiomatisation, completeness, sequential consistency

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.20

Related Version A version with detailed proofs is available at `https://arxiv.org/abs/2007.07593`.

Funding ERC Starting Grant ProFoundNet (679127).
Paul Brunet: EPSRC grant IRIS (EP/R006865/1).
Simon Docherty: EPSRC grant (EP/S013008/1).

1 Introduction

Kleene Algebra (KA) was originally proposed as the algebra of regular languages [21, 4, 16, 13], but its well-developed meta-theory facilitates applications in the analysis and verification of sequential programs. Many extensions of KA were studied in the last decades, notably Kleene Algebra with Tests (KAT) [14], which enables reasoning about control structures such as if-statements and while-loops. Orthogonally, Concurrent Kleene Algebra (CKA) was proposed as an extension of KA to analyse concurrent program behaviour [9].

It is a natural question whether concurrent Kleene algebra can be extended with tests as in KAT. This question was studied by Jipsen [10] and later by Kappé et al. [11, 12], who proposed Concurrent Kleene Algebra with Observations (CKAO). Observations are tests in a concurrent setting, and they are governed by different axioms than tests, hence justifying their name change. It was illustrated that extending CKA with tests in a naive way results in an algebraic framework that is unusable in program verification. In a nutshell, the interactions of parallel threads are lost if we identify the conjunction of observations

with their sequential composition, as is done in KAT. Instead, an algebra where conjunction and sequential composition are kept distinct is essential to capture concurrent interaction between conditionals in different threads – this distinguishes tests from observations.

In this paper we demonstrate how this class of techniques can be used for a more fine-grained analysis of concurrent programs. We focus our development around the issue of *sequential consistency*, i.e., whether programs behave as if memory accesses taking place were interleaved and executed sequentially [17]. A standard way of testing this property is the so-called *store buffering litmus test* [1]. Consider the following program with two threads:

$$\begin{array}{lll} \mathsf{T0}\colon & x \leftarrow 1; & \Big\| \quad \mathsf{T1}\colon \quad y \leftarrow 1; \\ & r_0 \leftarrow y; & \Big\| \qquad\quad r_1 \leftarrow x; \end{array}$$

A sequentially consistent implementation should satisfy the following property: if initially both registers r_0 and r_1 are set to 0, after running the program one of them should have value 1. Therefore, we can detect failures of sequential consistency by observing behaviour that deviates from this specification. This test can be encoded algebraically [15] as:

$$\Big((r_0 = 0 \wedge r_1 = 0); (\mathsf{T0} \parallel \mathsf{T1}); \neg(r_0 = 1 \vee r_1 = 1)\Big) \equiv 0. \tag{$\dagger$}$$

That is, the program that asserts that r_0 and r_1 are both 0, executes $\mathsf{T0}$ and $\mathsf{T1}$ in parallel, and then asserts that neither r_0 nor r_1 is 1, is equivalent to the program 0, which has no valid behaviour. To reason in this fashion, our algebraic framework should include *observations* of the shape $v = n$ as well as *assignments* $v \leftarrow n$ and $v \leftarrow v'$, where v, v' and n range over some fixed sets of variables and values. To that end, we propose *Partially Observable Concurrent Kleene Algebra* (*POCKA*), an algebraic theory built on top of CKA that allows for an analysis of concurrent programs manipulating memory, such as the simple program above. POCKA has a natural interpretation in terms of pomset languages over assignments and memory states, encoded as partial functions, similarly to separation logic [20], which describe the behaviour of concurrent programs that can access variables and values (Section 3). We prove soundness and completeness with respect to this interpretation (Section 4).

POCKA deviates from KAT and CKAO by using *partial* observations in its semantics. These are crucial in a concurrent setting, where a single thread may have only a partial view of the memory. Whilst memory as a whole depends on the combined action of all threads, these partial views may be analysed on a thread-by-thread basis. This shift from total to partial observations thus allows for a richer compositional semantic model. Formally, this means that we move from a Boolean algebra of observations, as in CKAO or KAT, to a pseudocomplemented distributive lattice (PCDL) [3], as proposed by Jipsen and Moshier [10].

To ensure compositionality, semantics of concurrent programs should capture not only isolated program behaviour, but also all *possible* behaviours of the program when run in parallel with another program. For example, take the program $P = (x = 1); (x = 2)$, which asserts that x has value 1 and then value 2, and the program $Q = (x \leftarrow 2)$, which assigns the value 2 to x. In an interpretation that captures isolated program behaviour, the semantics of P would be empty, as x cannot change between the tests. In contrast, the program $P \parallel Q$ (i.e., P and Q in parallel) *does* have behaviour, because the assignment may be interleaved between the two observations. Hence, the isolated semantics of P is not sufficient.

Thus, the semantics of a POCKA term accommodates possible interference by an outside context. As a result, the test ($\dagger$) fails at this stage, meaning this semantics is not sequentially consistent. This raises the question of how to study the isolated program behaviour. To this end, we identify a subset of the semantics that captures isolated program behaviour, and show that this fragment coincides with guarded pomsets [10] (Section 5). This turns out to fix the defect in sequential consistency we observe earlier, as we show in Section 6.

2 Preliminaries

Throughout this section we fix a finite alphabet Σ. We recall pomsets [7, 8], a generalisation of words that model concurrent traces. First, a *labelled poset* over Σ is a tuple $\mathbf{u} = \langle S_{\mathbf{u}}, \leq_{\mathbf{u}}, \lambda_{\mathbf{u}} \rangle$, where $S_{\mathbf{u}}$ is a finite set (the *carrier* of $\mathbf{u}$), $\leq_{\mathbf{u}}$ is a partial order on $S_{\mathbf{u}}$ (the *order* of $\mathbf{u}$), and $\lambda_{\mathbf{u}} \colon S_{\mathbf{u}} \to \Sigma$ is a function (the *labelling* of $\mathbf{u}$). Pomsets are labelled posets up to isomorphism:

▶ **Definition 2.1** (Poset isomorphism, pomset). *Let $\mathbf{u}, \mathbf{v}$ be labelled posets over Σ. We say $\mathbf{u}$ is* isomorphic *to $\mathbf{v}$, denoted $\mathbf{u} \cong \mathbf{v}$, if there exists a bijection $h \colon S_{\mathbf{u}} \to S_{\mathbf{v}}$ that preserves labels, and preserves and reflects ordering. More precisely, we require that $\lambda_{\mathbf{v}} \circ h = \lambda_{\mathbf{u}}$, and $s \leq_{\mathbf{u}} s'$ if and only if $h(s) \leq_{\mathbf{v}} h(s')$. A* pomset *over Σ is an isomorphism class of labelled posets over Σ, i.e., the class $[\mathbf{v}] = \{\mathbf{u} \mid \mathbf{u} \cong \mathbf{v}\}$ for some labelled poset $\mathbf{v}$.*

When two pomsets are in scope, we tacitly assume that they are represented by labelled posets with disjoint carriers. We write $\mathsf{Pom}(\Sigma)$ for the set of pomsets over Σ, and 1 for the empty pomset. When $\mathsf{a} \in \Sigma$, we write a for the pomset represented by the labelled poset whose sole element is labelled by a. Pomsets can be composed in sequence and in parallel:

▶ **Definition 2.2** (Pomset composition). *Let $U = [\mathbf{u}]$ and $V = [\mathbf{v}]$ be pomsets over Σ.*

We write $U \parallel V$ for the parallel composition *of U and V, which is the pomset over Σ represented by the labelled poset $\mathbf{u} \parallel \mathbf{v}$, where $S_{\mathbf{u}\parallel\mathbf{v}} = S_{\mathbf{u}} \cup S_{\mathbf{v}}$, $\leq_{\mathbf{u}\parallel\mathbf{v}} = \leq_{\mathbf{u}} \cup \leq_{\mathbf{v}}$ and for $x \in S_{\mathbf{u}}$ we have $\lambda_{\mathbf{u}\parallel\mathbf{v}}(x) = \lambda_{\mathbf{u}}(x)$ and for $x \in S_{\mathbf{v}}$ we let $\lambda_{\mathbf{u}\parallel\mathbf{v}}(x) = \lambda_{\mathbf{v}}(x)$.*

We write $U \cdot V$ for the sequential composition *of U and V, that is, the pomset represented by the labelled poset $\mathbf{u} \cdot \mathbf{v}$, where $S_{\mathbf{u}\cdot\mathbf{v}} = S_{\mathbf{u}\parallel\mathbf{v}}$, $\leq_{\mathbf{u}\cdot\mathbf{v}} = \leq_{\mathbf{u}} \cup \leq_{\mathbf{v}} \cup (S_{\mathbf{u}} \times S_{\mathbf{v}})$ and $\lambda_{\mathbf{u}\cdot\mathbf{v}} = \lambda_{\mathbf{u}\parallel\mathbf{v}}$.*

The pomsets that we use can be built using sequential and parallel composition.

▶ **Definition 2.3** (Series-parallel pomsets). *The set of* series-parallel pomsets (sp-pomsets) *over Σ, denoted $\mathsf{SP}(\Sigma)$, is the smallest subset of $\mathsf{Pom}(\Sigma)$ such that $1 \in \mathsf{SP}(\Sigma)$ and $\mathsf{a} \in \mathsf{SP}(\Sigma)$ for every $\mathsf{a} \in \Sigma$, and is furthermore closed under parallel and sequential composition.*

One way of comparing pomsets is to see whether they have the same events and labels, except that one is "more sequential" in the sense that more events are ordered. This is captured by the notion of *subsumption* [7], defined as follows.

▶ **Definition 2.4** (Subsumption). *Let $U = [\mathbf{u}]$ and $V = [\mathbf{v}]$. We say U* is subsumed by *V, written $U \sqsubseteq V$, if there exists a label- and order-preserving bijection $h \colon S_{\mathbf{v}} \to S_{\mathbf{u}}$. That is, h is a bijection such that $\lambda_{\mathbf{u}} \circ h = \lambda_{\mathbf{v}}$ and if $s \leq_{\mathbf{v}} s'$, then $h(s) \leq_{\mathbf{u}} h(s')$.*

In the rest of this paper we only consider the relation $\sqsubseteq$ restricted to series-parallel pomsets. We will also need the notion of *pomset contexts* [12].

▶ **Definition 2.5.** *Let $*$ be a symbol not in Σ. The set of* pomset contexts*, denoted $\mathsf{PC}(\Sigma)$, is the smallest subset of $\mathsf{SP}(\Sigma \cup \{*\})$ satisfying*

$$\frac{}{* \in \mathsf{PC}(\Sigma)} \qquad \frac{X \in \mathsf{SP}(\Sigma \cup \{*\}) \qquad C \in \mathsf{PC}(\Sigma)}{X \cdot C \in \mathsf{PC}(\Sigma) \qquad C \cdot X \in \mathsf{PC}(\Sigma)} \qquad \frac{X \in \mathsf{SP}(\Sigma \cup \{*\}) \qquad C \in \mathsf{PC}(\Sigma)}{X \parallel C \in \mathsf{PC}(\Sigma)}$$

Alternatively, $\mathsf{PC}(\Sigma)$ consists of the sp-pomsets over $\Sigma \cup \{*\}$ with exactly one occurrence of $*$.

One can think of $*$ as a gap where another pomset can be inserted: given $C \in \mathsf{PC}$ and $U \in \mathsf{Pom}$, we can insert U into the gap in C to obtain $C[U]$. More precisely, we define

$$*[U] = U \qquad (C \cdot X)[U] = C[U] \cdot X \qquad (X \cdot C)[U] = X \cdot C[U] \qquad (X \parallel C)[U] = X \parallel C[U]$$

This insertion is well-defined, and can in fact be extended to pomsets in general [12]. We extend the notation to a set of pomsets $L \subseteq \mathsf{Pom}$ by $C[L] = \{C[U] \mid U \in L\}$.

Bi-Kleene Algebra (BKA): syntax and semantics. *Bi-Kleene Algebra* [18] adds a binary operator, denoted $\|$, to KA, which satisfies a few basic axioms but does not interact with the other KA operators. *BKA-terms* over Σ, denoted $\mathcal{E}_\Sigma$ (the subscript is omitted if it is clear from the context), also called series-rational expressions [19], are generated by the grammar

$$e, f ::= 0 \mid 1 \mid \mathtt{a} \in \Sigma \mid e + f \mid e \cdot f \mid e \parallel f \mid e^*$$

The semantics of a BKA-term is a *pomset language*, i.e., an element of 2^{SP}. Formally, the BKA-semantics is a function $[\![-]\!] \colon \mathcal{E} \to 2^{\mathsf{SP}}$ defined inductively, as follows:

$$[\![0]\!] = \emptyset \qquad [\![1]\!] = \{1\} \qquad [\![e + f]\!] = [\![e]\!] \cup [\![f]\!] \qquad [\![e \cdot f]\!] = [\![e]\!] \cdot [\![f]\!]$$
$$[\![e^*]\!] = [\![e]\!]^* \qquad [\![\mathtt{a}]\!] = \{\mathtt{a}\} \qquad [\![e \parallel f]\!] = [\![e]\!] \parallel [\![f]\!]$$

In this definition we use the pointwise lifting of sequential and parallel composition from pomsets to pomset languages, e.g., $L \cdot K = \{U \cdot V \mid U \in L,\ V \in K\}$. The Kleene star of a pomset language L is defined as $L^* = \bigcup_{n \in \mathbb{N}} L^n$, where $L^0 = \{1\}$ and $L^{n+1} = L^n \cdot L$.

We write $\equiv_{\mathsf{BKA}}$ or simply $\equiv$ for the smallest congruence on $\mathcal{E}$ generated by the Kleene algebra axiom together with the additional bi-Kleene algebra axioms, which govern the parallel operator $\|$; it is associative, commutative, has a unit and distributes over $+$ (Table 1). Soundness and completeness of $\equiv_{\mathsf{BKA}}$ w.r.t. the pomset language semantics was proved in [18]:

▶ **Theorem 2.6** (Soundness and Completeness BKA). *Let $e, f \in \mathcal{E}$. Then $e \equiv f \Leftrightarrow [\![e]\!] = [\![f]\!]$.*

Given alphabets Σ and Γ, a function $h \colon \Sigma \to \mathcal{E}_\Gamma$ extends inductively to a map $\hat{h} \colon \mathcal{E}_\Sigma \to \mathcal{E}_\Gamma$ (e.g., $\hat{h}(e + f) = \hat{h}(e) + \hat{h}(f)$) which we refer to as the *homomorphism generated by h*.

Concurrent Kleene Algebra with Hypotheses (CKAH). *Concurrent Kleene algebra with Hypotheses* [12] (see also [5] for the case of KA), allows for a set of additional axioms, called *hypotheses*, to be added to the axioms of BKA. Based on these hypotheses, one can then derive a sound model. This facilitates a modular completeness proof of POCKA based on the completeness of BKA, as POCKA extends BKA with additional axioms.

▶ **Definition 2.7.** *A* hypothesis *is an inequation $e \leq f$ where $e, f \in \mathcal{E}$. When H is a set of hypotheses, we write $\equiv^H$ for the smallest congruence on $\mathcal{E}$ generated by the hypotheses in H as well as the axioms and implications that build the equational theory of BKA. More concretely, whenever $e \leq f \in H$, also $e \leqq^H f$.*

▶ **Definition 2.8.** *Let $L \subseteq \mathsf{Pom}$. We define the H-closure of L, written $L{\downarrow^H}$, as the smallest language containing L such that for all $e \leq f \in H$ and $C \in \mathsf{PC}$, if $C[[\![f]\!]] \subseteq L{\downarrow^H}$, then $C[[\![e]\!]] \subseteq L{\downarrow^H}$. We stress here the use of the BKA-semantics for defining the H-closure of any language. Formally, $L{\downarrow^H}$ may be described as the smallest language satisfying:*

$$\frac{}{L \subseteq L{\downarrow^H}} \qquad \frac{e \leq f \in H \qquad C \in \mathsf{PC} \qquad C[[\![f]\!]] \subseteq L{\downarrow^H}}{C[[\![e]\!]] \subseteq L{\downarrow^H}}$$

The H-closure adds those pomsets that are needed to ensure soundness of the axioms generated by H. This yields a sound model for BKA with the set of hypotheses H [12]:

▶ **Lemma 2.9** (Soundness). *If $e \equiv^H f$, then $[\![e]\!]{\downarrow^H} = [\![f]\!]{\downarrow^H}$.*

An axiom often added to BKA is the *exchange law*, and together with BKA it axiomatises Concurrent Kleene Algebra (CKA). It can be added in the form of a set of hypotheses [12]:

▶ **Definition 2.10.** *We write* exch *for the set* $\{(e \parallel f)\cdot(g \parallel h) \leq (e\cdot g) \parallel (f\cdot h) \mid e,f,g,h \in \mathcal{E}\}$.

These hypotheses encode the interleavings of a program: when $e \cdot g$ runs in parallel with $f \cdot h$, one possible behaviour is that e first runs in parallel with f, followed by g in parallel with h.

The exch-closure coincides with the downwards closure w.r.t. the subsumption order [12].

▶ **Lemma 2.11.** *Let* $L \subseteq \mathsf{SP}$ *and* $U \in \mathsf{SP}$. $U \in L{\downarrow^{\mathsf{exch}}} \Leftrightarrow$ *there exists a* $V \in L$ *s.t.* $U \sqsubseteq V$.

▶ **Definition 2.12.** *A map* $c\colon \mathcal{E} \to \mathcal{E}$ *is a* syntactic closure *for* H *when for all* $e \in \mathcal{E}$ *it holds that* $e \equiv^H c(e)$ *and* $[\![e]\!]{\downarrow^H} = [\![c(e)]\!]$.

Syntactic closures are used in modular constructions of completeness proofs: their existence implies a completeness result for H, by reducing it to completeness of BKA, i.e. Theorem 2.6.

3 Partially Observable Concurrent Kleene Algebra

In this section we define *partially observable concurrent Kleene algebra* (*POCKA*). The syntax of POCKA is given by BKA terms over an alphabet tailor-made to reason about programs that can access variables and values. Specifically, this alphabet holds *assignments* of the form $(v \leftarrow n)$ and $(v \leftarrow v')$, and *observations* of the form $(v = n)$. We say $(v \leftarrow n)$ assigns the value n to variable v, $(v \leftarrow v')$ copies the value of variable v' to v, and $(v = n)$ asserts that v must have value n. Formally, we define the alphabets

$$\mathsf{Act} = \{(v \leftarrow n), (v \leftarrow v') \mid v, v' \in \textsc{Var}, n \in \textsc{Val}\} \qquad \mathsf{Obs} = \{(v = n) \mid v \in \textsc{Var}, n \in \textsc{Val}\}$$

where Var and Val are finite sets of variables and values, respectively (see Remark 3.12 for a discussion on the finiteness assumption). An example POCKA term would be $(x = 1) \cdot (x \leftarrow 2) \cdot (x = 2)$, which asserts that x must start with value 1, assigns the value 2 to x, and then asserts that x holds the value 2.

We will later give semantics to POCKA terms using program states, which are partial functions from Var to Val: $\mathsf{State} = \{\alpha \mid \alpha\colon \textsc{Var} \rightharpoonup \textsc{Val}\}$. The domain of a state α is denoted $\mathsf{dom}(\alpha)$. State carries a partial order $\leq$, where $\alpha \leq \beta$ iff $\mathsf{dom}(\beta) \subseteq \mathsf{dom}(\alpha)$ and for all $x \in \mathsf{dom}(\beta)$ we have $\alpha(x) = \beta(x)$, which we will use to generate the algebra of observations.

3.1 Observation algebra: axiomatisation and semantics

To obtain POCKA, we define the *observation* algebra (OA) that will be added to CKA as the algebraic structure of observations. This is similar to how a Boolean algebra enriches Kleene algebra into Kleene algebra with tests. In contrast with KAT, the observation algebra of POCKA is a pseudocomplemented distributive lattice, which is a generalisation of Boolean algebra in which the law of excluded middle does not necessarily hold.

▶ **Definition 3.1** (Pseudocomplemented Distributive Lattice)**.** *A* pseudocomplemented distributive lattice *(PCDL) is a tuple* $(A, \wedge, \vee, \overline{\cdot}, \top, \bot)$ *such that* $(A, \wedge, \vee, \top, \bot)$ *is a bounded distributive lattice and* $\overline{\cdot}\colon A \to A$ *is such that for* $p, q \in A$ *we have* $p \wedge q = \bot$ *iff* $p \leq \overline{q}$.

For a poset $(X, \leq)$ and a set $S \subseteq X$, define the *downwards-closure* of S by $S_{\leq} ::= \{x \mid \exists y \in S \text{ s.t. } x \leq y\}$ and $P_{\leq}(X) ::= \{Y \subseteq X \mid Y = Y_{\leq}\}$. It is well-known that $P_{\leq}(X)$ carries the structure of a bounded distributive lattice, with intersection as meet, union as join, X as top and $\emptyset$ as bottom. Further, if $(X, \leq)$ is finite, the lattice is itself finite and thus carries a (necessarily unique) pseudocomplement defined by $\overline{Y} ::= \bigcup\{Z \in P_{\leq}(X) \mid Y \cap Z = \emptyset\}$. This simply reifies that the pseudocomplement of an element is the largest element incompatible with it, which is guaranteed to exist in any complete lattice with bottom.

Table 1 Axioms of POCKA, built over an alphabet of actions Act and observations Obs. The left column contains the axioms of Concurrent Kleene Algebra. The right column axiomatises the partial observations: they form a pseudocomplemented distributive lattice, subject to constraints on the interface axioms that connect the lattice operators to the Kleene algebra ones. The last group of axioms applies to the observation alphabet Obs. We write $e \leqq f$ as a shorthand for $e + f \equiv f$.

Kleene Algebra Axioms

$$e + (f + g) \equiv (f + g) + h$$
$$e + f \equiv f + e$$
$$e + 0 \equiv e$$
$$e + e \equiv e$$
$$e \cdot (f \cdot g) \equiv (e \cdot f) \cdot g$$
$$e \cdot 1 \equiv e \equiv 1 \cdot e$$
$$e \cdot 0 \equiv 0 \equiv 0 \cdot e$$
$$e \cdot (f + g) \equiv e \cdot f + e \cdot h$$
$$(e + f) \cdot g \equiv e \cdot g + f \cdot g$$
$$e^* \equiv 1 + ee^*$$
$$e + f \cdot g \leqq f \Rightarrow e \cdot g^* \leqq f$$
$$e^* \equiv 1 + e^*e$$
$$e + f \cdot g \leqq g \Rightarrow f^* \cdot e \leqq g$$

Additional Bi-Kleene Algebra Axioms

$$e \parallel 1 \equiv e$$
$$e \parallel (f \parallel g) \equiv (e \parallel f) \parallel g$$
$$e \parallel 0 \equiv 0$$
$$e \parallel (f + g) \equiv e \parallel f + e \parallel g$$
$$e \parallel f \equiv f \parallel e$$

Exchange law

$$(e \parallel f) \cdot (g \parallel h) \leqq (e \cdot g) \parallel (f \cdot h)$$

Bounded Distributive Lattice Axioms

$$p \vee \bot \equiv p \equiv p \wedge \top$$
$$p \vee q \equiv q \vee p$$
$$p \wedge q \equiv q \wedge p$$
$$p \wedge (q \wedge r) \equiv (p \wedge q) \wedge r$$
$$p \vee (q \vee r) \equiv (p \vee q) \vee r$$
$$p \vee (p \wedge q) \equiv p \equiv p \wedge (p \vee q)$$
$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$$
$$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$$

Pseudocomplement

$$p \leqq \overline{q} \Leftrightarrow p \wedge q \equiv \bot$$

Observation Axioms

$$v = n \wedge v = m \equiv \bot \qquad (n \neq m)$$
$$\overline{v = n} \leqq \bigvee_{n \neq m} v = m$$
$$\overline{\bigwedge_i v_i = n_i} \leqq \bigvee_i \overline{v_i = n_i} \qquad (\forall i \neq j . v_i \neq v_j)$$

Interface Axioms

$$p \wedge q \leqq p \cdot q$$
$$p \vee q \equiv p + q$$
$$0 \equiv \bot$$
$$\top \cdot p \leqq p \qquad p \cdot \top \leqq p \qquad (p \in \mathcal{O})$$
$$\top \cdot a \leqq a \qquad a \cdot \top \leqq a \qquad (a \in \mathsf{Act})$$

▶ **Definition 3.2** (Observation Algebra)**.** *The* Observation Algebra *is the PCDL* OA $::= (P_{\leq}(\mathsf{State}), \cap, \cup, \overline{\cdot}, \mathsf{State}, \emptyset)$ *generated by* $(\mathsf{State}, \leq)$.

Taking Obs as our set of propositions, we generate a term language $\mathcal{O}$ over the signature of PCDLs as follows:

$$p, q ::= \bot \mid \top \mid o \in \mathsf{Obs} \mid p \vee q \mid p \wedge q \mid \overline{p}.$$

This language is interpreted in OA by the homomorphic extension of the assignment

$$[\![v = n]\!] ::= \{\{v \mapsto n\}\}_{\leq} = \{\alpha \in \mathsf{State} \mid \alpha(v) = n\}.$$

Intuitively, the behaviour of an observation p consists of all partial functions that agree with p. This is captured algebraically below, in Lemma 3.7. For instance, $[\![v = n]\!]$ is the set containing all partial functions assigning n to v, and this is downwards closed because any partial function with a larger domain that also assigns n to v is included in this set.

If threads have only partial information about the machine state, an observation should be satisfied only if there is *positive evidence* for it. Hence, $\overline{v = n}$ should be satisfied only when v has a value that is different from n. To see why a Boolean algebra does not capture

the intended meaning of $\overline{v = n}$, consider using a BA over sets of partial functions, with negation as set-complement. This entails that $[\![\overline{v = n}]\!]$ will include all partial functions where v either gets a different value than n or no value at all. In the latter case, were we to obtain more information about the machine state we may discover that the actual value of v is in fact n, and it was therefore incorrect to assert $\overline{v = n}$. Our pseudocomplement provides a notion of negation that correctly excludes states for which this error could manifest, and this motivates our use of a PCDL rather than a Boolean algebra. This can be calculated directly:

▶ **Example 3.3.** Consider the semantics of $\overline{v = n}$:

$$\begin{aligned}[\![\overline{v = n}]\!] &= \bigcup\{Z \in P_{\leq}(\mathsf{State}) \mid [\![v = n]\!] \cap Z = \emptyset\} \\ &= \{\alpha \mid \alpha \in Z \text{ and } \{\beta \mid \beta(v) = n\} \cap Z = \emptyset\} \\ &= \{\alpha \mid \alpha(v) = m \text{ and } m \neq n\}\end{aligned}$$

In the last step we use that Z is downwards closed: if $\alpha(v)$ were undefined, then the partial function α' which is the same as α except that $\alpha'(v) = n$ would also occur in Z, making the intersection with $[\![v = n]\!]$ non-empty. Thus $\alpha \in [\![\overline{v = n}]\!]$ only if $\alpha(v)$ is defined, and evaluates to a value distinct from n. This witnesses the failure of the law of excluded middle.

▶ **Definition 3.4** (Axiomatisation). *$\equiv_{\mathsf{OA}}$, or simply $\equiv$, is the smallest congruence on $\mathcal{O}$ generated by the distributive lattice, pseudocomplement and observation axioms in Table 1.*

This axiomatisation supplements a standard axiomatisation of PCDLs with domain-specific axioms to capture the propositional theory of observations. For instance, the axiom $(v = n \wedge v = m \equiv \bot)$ states that a variable cannot have two different values at the same time. The axiom $(\overline{v = n} \leqq \bigvee_{n \neq m} v = m)$ tells us that the pseudocomplement of a variable having a value n is the assertion that the variable holds *some* distinct value m (the axiom is an inequality, but the other way around also holds). The last domain-specific axiom enforces specific instances of a De Morgan law that does not hold generally hold in arbitrary PCDLs.

Soundness of this axiomatisation follows straightforwardly from the fact that OA is a PCDL, together with basic consequences of the definition of the poset $(\mathsf{State}, \leq)$.

▶ **Lemma 3.5** (Soundness OA). *For all $p, q \in \mathcal{O}$, if $p \equiv q$ then $[\![p]\!] = [\![q]\!]$.*

Let $\pi_\alpha ::= \bigwedge_{\alpha(v)=n} v = n$. Note that if α is the empty function, then $\pi_\alpha = \bigwedge \emptyset = \top$.

▶ **Lemma 3.6.** *For all $\alpha, \beta \in \mathsf{State}$: $\alpha \in [\![\pi_\beta]\!]$ iff $\alpha \leq \beta$ iff $\pi_\alpha \leqq \pi_\beta$.*

In the following sections, we will silently assume that $\mathsf{State} \subseteq \mathcal{O}$. This is possible because π_- provides us with a sound way of injecting State inside $\mathcal{O}$. In order to prove completeness for $[\![-]\!]$ w.r.t. $\equiv$, we need an intermediary result, which allows us to syntactically rewrite any OA-expression in terms of elements of State.

▶ **Lemma 3.7.** *For all $p \in \mathcal{O}$, we have $p \equiv \bigvee\{\alpha \in \mathsf{State} \mid \pi_\alpha \leqq p\}$.*

With this result, we can then prove completeness of $[\![-]\!]$ w.r.t. $\equiv$ on terms from $\mathcal{O}$. In short, from $[\![p]\!] = [\![q]\!]$, $\pi_\alpha \leqq p$ iff $\pi_\alpha \leqq q$ can be established, from which $p \equiv q$ follows.

▶ **Theorem 3.8** (Completeness OA). *For all $p, q \in \mathcal{O}$, we have $p \equiv q$ if and only if $[\![p]\!] = [\![q]\!]$.*

3.2 POCKA: axiomatisation and semantics

▶ **Definition 3.9.** *The POCKA-terms, denoted $\mathcal{T}$, are formed by the following grammar:*

$$e, f ::= 0 \mid 1 \mid a \in \mathsf{Act} \mid p \in \mathcal{O} \mid e + f \mid e \cdot f \mid e \parallel f \mid e^* \ .$$

Note that $\mathcal{T} = \mathcal{E}_{\mathsf{Act} \cup \mathcal{O}}$.

The language model for POCKA consists of pomset languages over $\mathsf{Act} \cup \mathsf{State}$. When using pomsets to reason about behaviours of programs, we would like actions and states to alternate, because the states allow one to take stock of the configuration of the machine in between actions. However, imposing such an alternation in the semantics can be problematic with the exchange law [11]. Imagine the program $(\alpha \cdot \beta) \parallel \mathtt{a}$, where $\alpha, \beta \in \mathsf{State}$ and $\mathtt{a} \in \mathsf{Act}$. We can derive the following:

$$\begin{aligned}
\alpha \cdot \mathtt{a} \cdot \beta &\equiv (\alpha \parallel 1) \cdot (1 \parallel \mathtt{a}) \cdot (\beta \parallel 1) && \text{(Unit axiom)} \\
&\leqq (\alpha \parallel 1) \cdot ((1 \cdot \beta) \parallel (\mathtt{a} \cdot 1)) && \text{(Exchange Law)} \\
&\equiv (\alpha \parallel 1) \cdot (\beta \parallel \mathtt{a}) && \text{(Unit Axiom)} \\
&\leqq (\alpha \cdot \beta) \parallel (1 \cdot \mathtt{a}) \equiv (\alpha \cdot \beta) \parallel \mathtt{a} && \text{(Exchange Law, Unit Axiom)}
\end{aligned}$$

However, if the semantics $(\!|-|\!)$ contain only pomsets with alternating assignments and states, $(\!|\alpha \cdot \beta|\!)$ would have to consist of one state, and hence be empty if $\alpha \neq \beta$. This would make $(\!|(\alpha \cdot \beta) \parallel \mathtt{a}|\!)$ empty as well. As $(\!|\alpha \cdot \mathtt{a} \cdot \beta|\!)$ should not be empty, the exchange law is unsound. Thus, the POCKA-semantics is not restricted to pomsets with alternating states and actions.

▶ **Definition 3.10** (Semantics). *Let $(\!|-|\!) \colon \mathcal{T} \to 2^{\mathsf{SP}}$, where 2^{SP} are pomset languages over $\mathsf{Act} \cup \mathsf{State}$. For $p \in \mathcal{O}$, $(v \leftarrow n), (v \leftarrow v') \in \mathsf{Act}$ and $e, f \in \mathcal{T}$ we have:*

$$\begin{aligned}
(\!|v \leftarrow n|\!) &= \mathsf{State}^* \cdot \{v \leftarrow n\} \cdot \mathsf{State}^* & (\!|e + f|\!) &= (\!|e|\!) \cup (\!|f|\!) & (\!|e^*|\!) &= (\!|e|\!)^* \\
(\!|v \leftarrow v'|\!) &= \mathsf{State}^* \cdot \{v \leftarrow v'\} \cdot \mathsf{State}^* & (\!|e \cdot f|\!) &= (\!|e|\!) \cdot (\!|f|\!) & (\!|0|\!) &= \emptyset \\
(\!|p|\!) &= \mathsf{State}^* \cdot [\![p]\!]_{\mathsf{OA}} \cdot \mathsf{State}^* & (\!|e \parallel f|\!) &= (\!|e|\!) \parallel (\!|f|\!) & (\!|1|\!) &= \{1\}
\end{aligned}$$

We define the POCKA-semantics of $e \in \mathcal{T}$ as $(\!|e|\!)\!\downarrow = (\!|e|\!)\!\downarrow^{\mathsf{exch} \cup \mathsf{contr}}$, where we use the closure definition from Definition 2.8, and $\mathsf{contr} = \{\alpha \leq \alpha \cdot \alpha \mid \alpha \in \mathsf{State}\}$*, referred to as* contraction.

We briefly explain closure under exch and contr. These closures are not part of the axiomatisation, but exist to ensure soundness of some of the axioms. The set of hypotheses exch closes the POCKA-semantics under subsumption and ensures soundness for the exchange law familiar from CKA. The set contr encodes that one way of observing α twice is to make both observations on the same state. This provides soundness for the axiom $p \wedge q \leqq p \cdot q$, which was introduced in [11]. This axiom captures that if p and q hold simultaneously in some state, it is possible to observe p and q in sequence (the converse should not hold as some action could happen in between the two obervations in a parallel thread).

▶ Remark 3.11. The assignment $(v \leftarrow v')$ cannot be simulated. In a sequential setting, we could express $(v \leftarrow v')$ as $\sum_{n \in \mathrm{VAL}}((v' = n) \cdot (v \leftarrow n))$. However, in a parallel setting this does not work, since some action can change the value of v' in between the observation that $(v' = n)$ and the assignment $(v \leftarrow n)$, meaning that v does not get assigned the value of v'.

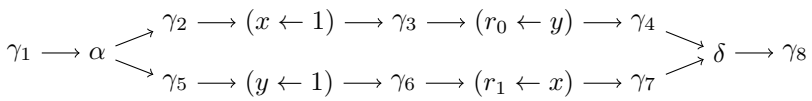
▶ Remark 3.12. In this paper we assume the set of variables VAR and the set of values VAL are both finite, in keeping with other verification frameworks, e.g. in model-checking.

The restriction on VAR could be lifted, since the finite set of variables that appear syntactically in a term completely determine its semantics. However, this is not the case for the set of values: for instance, the term $\overline{v = 0}$ evaluates to $\emptyset$ if the set of values is $\textsc{Val} = \{0\}$, but contains the partial function $[v \mapsto n]$ if VAL contains some value $n \neq 0$.

It is possible that a more sophisticated reduction could still work: indeed it seems unlikely that our finite terms would be able to manipulate non-trivially infinitely many values. For now though, this question is left open for future investigations.

The POCKA-semantics of a program e contains the possible behaviours of e in any possible context, where the context refers to any expression that could be put in parallel with e. For instance, $(\!|(v \leftarrow n)|\!)\downarrow$ contains pomsets that consist of a string of possible states of the machine, where the state of the machine can have been influenced by other parallel threads, followed at some point by the assignment $(v \leftarrow n)$, followed by another string of states. In Section 5, we will show how to reason about programs in isolation, i.e., under the hypothesis that there is no outside context to prompt state-modifying actions.

▶ **Example 3.13.** Let $t = (r_0 = 0 \wedge r_1 = 0) \cdot (\mathsf{T0} \parallel \mathsf{T1}) \cdot \overline{(r_0 = 1 \vee r_1 = 1)}$ as in (†) be our litmus test. A pomset in $(\!|t|\!)$ may look as follows, where we depict a pomset graphically with nodes labelled by actions or observations and their ordering with arrows.



Here, $\gamma_i \in \mathsf{State}$, $\alpha(r_0) = 0 = \alpha(r_1)$ and $\delta(r_0) = 0 = \delta(r_1)$. However, as stated in the introduction, if POCKA is sequentially consistent, this litmus test should pass, which means that the semantics of t should instead be empty. The reason it is not empty is that our semantics gives the behaviour of a program in any possible context, and indeed, if we put the litmus test in parallel with a program such as $(r_0 \leftarrow 0) \cdot (r_1 \leftarrow 0)$, the final assertion becomes satisfiable. In Sections 5 and 6, we look at how to execute the litmus test in isolation.

We also have axioms to algebraically describe equivalence between POCKA-terms, including some domain-specific axioms tailored to the alphabet. We define $\equiv$ as the smallest congruence on $\mathcal{T}$ generated by the axioms in Table 1.

▶ **Theorem 3.14** (Soundness POCKA). *For all $e, f \in \mathcal{T}$, if $e \equiv f$ then $(\!|e|\!)\downarrow = (\!|f|\!)\downarrow$.*

4 Completeness

In this section we prove completeness of the POCKA-semantics w.r.t. the axioms provided in Section 3.2. First, we show that POCKA terms can be normalised to a simpler form, where the only observations that appear are states. Next, we show that the resulting POCKA-terms can be used to describe the POCKA-semantics using BKA-semantics and closure. We then use the techniques from [12] to obtain a completeness result with respect to this semantics. Finally, we put all of the above together to obtain a completeness result for POCKA proper.

In order to normalise POCKA terms, we replace every observation by the summation of states to which it corresponds. This is done using the homomorphism $\hat{r}$ generated by

$$r(a) = \begin{cases} \sum_{\alpha \leq_{\mathsf{OA}} a} \alpha & a \in \mathcal{O} \\ a & a \in \mathsf{Act} \end{cases}$$

As a straightforward consequence of Lemma 3.7 and the interface axioms, we then obtain:

▶ **Lemma 4.1.** *For all $e \in \mathcal{T}$, it holds that $e \equiv \hat{r}(e)$.*

Proof. This can be proven by induction on the structure of e. If $e = \mathsf{a} \in \mathsf{Act}$, then $\hat{r}(\mathsf{a}) = \mathsf{a} \equiv \mathsf{a}$ immediately. Otherwise, if $p \in \mathcal{O}$, then we derive $\hat{r}(p) = \sum_{\alpha \leqq_{\mathsf{OA}} p} \alpha \equiv \bigvee_{\alpha \leqq_{\mathsf{OA}} p} \alpha \equiv p$, where we apply Lemma 3.7 in the last step. The inductive step follows trivially. ◀

The effect of $\hat{r}$ is to bridge the gap between the semantics of BKA and that of observation algebra: indeed for an observation $p \in \mathcal{O}$, we have $[\![\hat{r}(p)]\!] = [\![p]\!]_{\mathsf{OA}}$. However, this does not bring us fully to the unclosed POCKA-semantics $(\!|-|\!)$, as the latter inserts state-nodes in between actions and observations. For instance, $(\!|v \leftarrow n|\!)$ includes pomsets like $\alpha \cdot (v \leftarrow n) \cdot \beta$, while $[\![v \leftarrow n]\!] = \{v \leftarrow n\}$. We cover for this by means of the following set of hypotheses:

$$\mathsf{top} = \{\alpha \cdot c \leq c, c \cdot \alpha \leq c \mid \alpha \in \mathsf{State}, c \in \mathsf{Act} \cup \mathsf{State}\}$$

The hypotheses in top allow us to connect the unclosed POCKA semantics to the BKA-semantics, by filling in surrounding or preceding state-labelled nodes as necessary.

▶ **Lemma 4.2.** *For all $e \in \mathcal{T}$, we have $(\!|e|\!) = [\![\hat{r}(e)]\!]{\downarrow^{\mathsf{top}}}$.*

Sketch. Proceed by induction on the construction of e; the case where $e \in \mathsf{Act} \cup \mathcal{O}$ is fairly straightforward. The inductive case follows from the fact that closure w.r.t. top is compatible with pomset language composition, i.e., that $(L \cup K){\downarrow^{\mathsf{top}}} = L{\downarrow^{\mathsf{top}}} \cup K{\downarrow^{\mathsf{top}}}$ as well as $(L \cdot K){\downarrow^{\mathsf{top}}} = L{\downarrow^{\mathsf{top}}} \cdot K{\downarrow^{\mathsf{top}}}$, and similarly for the other operators defining $[\![-]\!]$. ◀

The next step is to provide syntactic closures for the sets of hypotheses involved. First, we note that there exists a syntactic closure for $\mathsf{exch} \cup \mathsf{contr}$, as shown in [12, Theorem 5.6].

▶ **Lemma 4.3.** *There exists a syntactic closure k for $\mathsf{exch} \cup \mathsf{contr}$.*

For the set top we still need to provide a syntactic closure. To this end, we simply take every action or observation in a term and surround it by a sequence of states of arbitrary length, which can be done using the homomorphism generated by

$$s(a) = \Big(\sum_{\alpha \in \mathsf{State}} \alpha\Big)^* \cdot a \cdot \Big(\sum_{\alpha \in \mathsf{State}} \alpha\Big)^*$$

It is fairly straightforward to show that this gives rise to a syntactic closure.

▶ **Lemma 4.4.** *The homomorphism generated by s is a syntactic closure for top.*

Sketch. The proof proceeds by induction on the construction of a term e. In the base, we can show for $a \in \mathsf{Act} \cup \mathcal{O}$ that $s(a) \equiv^{\mathsf{top}} a$ and $[\![a]\!]{\downarrow^{\mathsf{top}}} = [\![s(a)]\!]$. The inductive step follows by an argument similar to that in Lemma 4.2. ◀

The final step needed for the completeness proof of POCKA is a relation between the axioms that generate $\equiv$ and the hypotheses found in top and contr.

▶ **Lemma 4.5.** *For all $e, f \in \mathcal{T}$, if $e \leq f \in \mathsf{top} \cup \mathsf{contr}$, then $e \leqq f$.*

We now have all the ingredients in place for the desired completeness proof.

▶ **Theorem 4.6.** *For all $e, f \in \mathcal{T}$, we have $e \equiv f$ if and only if $(\!|e|\!){\downarrow} = (\!|f|\!){\downarrow}$.*

Proof. The direction from left to right was already established in Theorem 3.14. For the other direction, suppose that $e, f \in \mathcal{T}$ such that $(\!|e|\!)\downarrow = (\!|f|\!)\downarrow$. We can then derive that

$$\begin{aligned}
(\!|e|\!)\downarrow &= (\!|e|\!)\downarrow^{\mathsf{exch}\cup\mathsf{contr}} && \text{(def. } (\!|-|\!)\downarrow) \\
&= \left(\llbracket \hat{r}(e) \rrbracket\downarrow^{\mathsf{top}}\right)\downarrow^{\mathsf{exch}\cup\mathsf{contr}} && \text{(Lemma 4.2)} \\
&= \llbracket \hat{s} \circ \hat{r}(e) \rrbracket\downarrow^{\mathsf{exch}\cup\mathsf{contr}} && \text{(Lemma 4.4)} \\
&= \llbracket k \circ \hat{s} \circ \hat{r}(e) \rrbracket && \text{(Lemma 4.3)}
\end{aligned}$$

Similarly, $(\!|f|\!)\downarrow = \llbracket k \circ \hat{s} \circ \hat{r}(e) \rrbracket$. Since $(\!|e|\!)\downarrow = (\!|f|\!)\downarrow$, also $\llbracket k \circ \hat{s} \circ \hat{r}(e) \rrbracket = \llbracket k \circ \hat{s} \circ \hat{r}(f) \rrbracket$; by Theorem 2.6, it then follows that $k \circ \hat{s} \circ \hat{r}(e) \equiv_{\mathsf{BKA}} k \circ \hat{s} \circ \hat{r}(e)$. We then derive that

$$\begin{aligned}
e &\equiv \hat{r}(e) && \text{(Lemma 4.1)} \\
&\equiv^{\mathsf{top}}_{\mathsf{BKA}} \hat{s} \circ \hat{r}(e) && \text{(Lemma 4.4)} \\
&\equiv^{\mathsf{exch}\cup\mathsf{contr}}_{\mathsf{BKA}} k \circ \hat{s} \circ \hat{r}(e) && \text{(Lemma 4.3)} \\
&\equiv_{\mathsf{BKA}} k \circ \hat{s} \circ \hat{r}(f) && \text{(Observation above)} \\
&\equiv^{\mathsf{exch}\cup\mathsf{contr}}_{\mathsf{BKA}} \hat{s} \circ \hat{r}(f) && \text{(Lemma 4.3)} \\
&\equiv^{\mathsf{top}}_{\mathsf{BKA}} \hat{r}(f) && \text{(Lemma 4.4)} \\
&\equiv f && \text{(Lemma 4.1)}
\end{aligned}$$

By Lemma 4.5, $\equiv^{\mathsf{top}}_{\mathsf{BKA}}$ and $\equiv^{\mathsf{exch}\cup\mathsf{contr}}_{\mathsf{BKA}}$ are contained in $\equiv$; we conclude that $e \equiv f$. ◀

5 Guarded Pomsets

We now identify a fragment of the semantics that we use in the analysis of the litmus test from (†), namely the *guarded* pomsets. This term comes from Jipsen and Moshier [10], and was meant to define guarded pomsets in analogy to guarded strings in KAT.

We need two pieces of notation. First, we define the result of a state after updating it for one value. Let $\mathsf{a} \in \mathsf{Act}$ and $\alpha \in \mathsf{State}$. We say that $\alpha[\mathsf{a}]$ *exists* if $\mathsf{a} = v \leftarrow n$ for some $n \in \textsc{Val}$ or $\mathsf{a} = v \leftarrow v'$ and $v' \in \mathsf{dom}(\alpha)$. If $\alpha[\mathsf{a}]$ exists, we define it for all $w \in \textsc{Var}$ as follows:

$$\alpha[v \leftarrow n](w) = \begin{cases} n & \text{if } w = v \\ \alpha(w) & \text{otherwise} \end{cases} \qquad \alpha[v \leftarrow v'](w) = \begin{cases} \alpha(v') & \text{if } w = v \\ \alpha(w) & \text{otherwise} \end{cases}$$

Second, we define a binary operator $\oplus$ on State to combine states. For $\alpha, \beta \in \mathsf{State}$:

$$\alpha \oplus \beta = \begin{cases} \alpha \cup \beta & \text{if } \alpha(v) = \beta(v) \text{ for all } v \in \mathsf{dom}(\alpha) \cap \mathsf{dom}(\beta) \\ \text{undefined} & \text{otherwise} \end{cases}$$

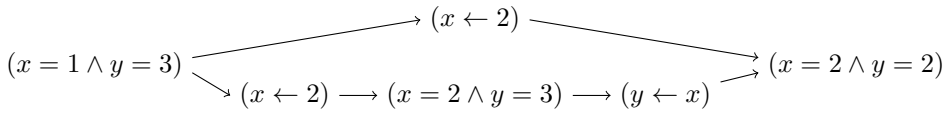
▶ **Definition 5.1.** *The set of* guarded pomsets*, denoted $\mathcal{G}$, is the smallest set satisfying:*

$$\frac{\alpha \in \mathsf{State}}{\alpha \in \mathcal{G}} \qquad \frac{\alpha \in \mathsf{State} \qquad \mathsf{a} \in \mathsf{Act} \qquad \alpha[\mathsf{a}] \textit{ exists}}{\alpha \cdot \mathsf{a} \cdot \alpha[\mathsf{a}] \in \mathcal{G}} \qquad \frac{U \cdot \alpha, \alpha \cdot V \in \mathcal{G} \qquad \alpha \in \mathsf{State}}{U \cdot \alpha \cdot V \in \mathcal{G}}$$

$$\frac{\alpha \cdot U \cdot \beta \qquad \gamma \cdot V \cdot \delta \in \mathcal{G} \qquad \alpha \oplus \gamma \textit{ defined} \qquad \beta \oplus \delta \textit{ defined} \qquad \alpha, \beta, \gamma, \delta \in \mathsf{State}}{\alpha \oplus \gamma \cdot (U \parallel V) \cdot \beta \oplus \delta \in \mathcal{G}}$$

This definition is close to [10]. The definition in op. cit. is not catered to a specific alphabet, and the operator $\oplus$ to combine states does not allow for the two states to have any shared variable in their domains. We deliberately deviate from this, allowing threads to share variables as long as they do so in a consistent manner.

▶ **Example 5.2.** Consider the guarded pomsets $(x = 1) \cdot (x \leftarrow 2) \cdot (x = 2)$ and $(x = 1 \wedge y = 3) \cdot (x \leftarrow 2) \cdot (x = 2 \wedge y = 3) \cdot (y \leftarrow x) \cdot (x = 2 \wedge y = 2)$. The final rule for the construction of $\mathcal{G}$ guarantees that the following pomset is again guarded:



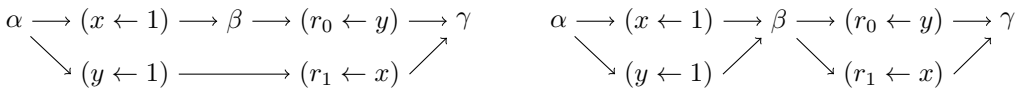
Note how $(x = 2) \oplus (x = 2 \wedge y = 3)$ is defined, because both states agree on the value of x.

In the execution of parallel threads in pomsets, no interaction between the threads takes place: the threads execute "truly" concurrently. To account for interactions, we consider the interleavings that result from closure w.r.t. the exchange law (c.f. Lemma 2.11).

▶ **Example 5.3.** Consider the a slightly adjusted version of the litmus test t discussed earlier:

$$t' = (r_0 = 0 \wedge r_1 = 0); (\mathsf{T0} \parallel \mathsf{T1}); (r_0 = 1 \vee r_1 = 1)$$

The *unclosed* semantics of t' includes (but is not limited to) the pomset below on the left, for all $\alpha, \beta, \gamma \in \mathsf{State}$, where $\alpha(r_0) = \alpha(r_1) = 0$, and $\gamma(r_0) = 1$ or $\gamma(r_1) = 1$. As a result of the exchange law, the *closed* semantics includes the pomset below on the right.



In the special case where $\alpha = \{r_0 \mapsto 0, r_1 \mapsto 0\}$, $\beta = \{r_0 \mapsto 0, r_1 \mapsto 0, x \mapsto 1, y \mapsto 1\}$, $\gamma = \{r_0 \mapsto 1, r_1 \mapsto 1, x \mapsto 1, y \mapsto 1\}$, the latter is a guarded pomset.

Guardedness in pomsets can be characterised by the conjunction of seven properties, which we will discuss now. On the one hand, these properties have an intuitive explanation as characteristics of behaviours of (possibly concurrent) programs running in isolation. Hence, if a pomset represents some execution of an isolated program, it is guarded. On the other hand, the characterisation in terms of these properties provides a proof method to show that a pomset is *not* guarded, by demonstrating the failure of one such property.

We start by observing that guarded pomsets alternate states and actions. Formally, we can capture this in three properties. Let $U = [\mathbf{u}] \in \mathsf{SP}(\mathsf{Act} \cup \mathsf{State})$. We say that $s' \in S_{\mathbf{u}}$ is a *predecessor* of s if it is the latest node ordered strictly before s – i.e., $s\ s' <_{\mathbf{u}} s$ and for all $s'' \in S_{\mathbf{u}}$ such that $s'' <_{\mathbf{u}} s$ it holds that $s'' \leq_{\mathbf{u}} s'$. The notion of *successor* is defined dually. A node is a *state-node* if it is labelled by an element of State, and an *action-node* otherwise.
(A1) U admits a unique minimum and maximum, $*_{\min}, *_{\max} \in S_{\mathbf{u}}$, labelled by states.
(A2) Every two related state-nodes are separated by an action-node.
(A3) Action-nodes have unique state-nodes as neighbours (their predecessor and successor).

The next property formalises the idea that two related observations cannot contradict each other, such as in the program $(x = 1) \cdot (x = 2)$. To this end, we need the notion of a *path*. A path for a variable v from a state-node u to another state-node s is a chain such that the changes in the value of v between u and s are explained by the actions between them and recorded in all the states between u and s.

▶ **Definition 5.4** (Path). *Let $U = [\mathbf{u}] \in \mathsf{Pom}(\mathsf{Act} \cup \mathsf{State})$ and $u_1, u_2 \in S_{\mathbf{u}}$ such that $u_1 \leq_{\mathbf{u}} u_2$ and $\lambda_{\mathbf{u}}(u_1), \lambda_{\mathbf{u}}(u_1) \in \mathsf{State}$. We say a* path *$p_v$ from u_1 to u_2 for variable $v \in \textsc{Var}$ is a sequence of nodes $q_1, a_1, \ldots, a_n, q_{n+1} \in S_{\mathbf{u}}$ that satisfy the following conditions:*

(P1) *For all* $1 \leq i \leq n$*, we have* $\lambda_{\mathbf{u}}(a_i) \in \mathsf{Act}$ *and* $u_1 \leq_{\mathbf{u}} a_i \leq_{\mathbf{u}} u_2$ *for all* i*. Additionally we require that* $a_i \leq_{\mathbf{u}} a_{i+1}$ *for* $1 \leq i < n$*.*

(P2) *For all* $1 \leq i \leq n+1$ *it holds that* $\lambda_{\mathbf{u}}(q_i) \in \mathsf{State}$*, and for all* $1 \leq i \leq n$*, the predecessor of* a_i *is* q_i*, and the successor of* a_i *is* q_{i+1}*. Additionally we have that* $\lambda_{\mathbf{u}}(q_1) = \lambda_{\mathbf{u}}(u_1)$*,* $v \in \mathsf{dom}(\lambda_{\mathbf{u}}(u_1))$ *and* $\lambda_{\mathbf{u}}(q_{n+1}) = \lambda_{\mathbf{u}}(u_2)$*. Lastly, for* $1 \leq i \leq n$ *we have:*

$$\lambda_{\mathbf{u}}(q_{i+1})(v) = \begin{cases} n & \lambda_{\mathbf{u}}(a_i) = v \leftarrow n \text{ for some } n \in \textsc{Val} \\ \lambda_{\mathbf{u}}(q_i)(v') & \lambda_{\mathbf{u}}(a_i) = v \leftarrow v' \text{ for some } v' \in \textsc{Var}, v' \in \mathsf{dom}(\lambda_{\mathbf{u}}(q_i)) \\ \lambda_{\mathbf{u}}(q_i)(v) & \text{otherwise} \end{cases}$$

▶ **Example 5.5.** The following is a path for x:

$$(x = 1) \cdot (y \leftarrow 3) \cdot (x = 1) \cdot (x \leftarrow 2) \cdot (x = 2 \wedge y = 3) \cdot (x \leftarrow y) \cdot (x = 3)$$

Note that this is not a path for y, because it is not assigned a value by the final atom.

We can now formulate another criterion for a pomset executing in isolation: for every variable in the domain of a state-node, there is a path explaining the changes in value of that variable between the state-node and the maximum node of the pomset.

(A4) For all state-nodes $u \in S_{\mathbf{u}}$ and $v \in \mathsf{dom}(\lambda_{\mathbf{u}}(u))$, there is a path for v from u to $*_{\max}$.

▶ **Example 5.6.** The first pomset below satisfies (A4), and the second pomset does not, as there is no path from beginning to end for x: the value of x in the second observation is not in accordance to the previous assignment.

$$(x = 2 \wedge y = 2) \begin{array}{l} \nearrow (x \leftarrow 4) \searrow \\ \searrow (y \leftarrow 3) \nearrow \end{array} (x = 4 \wedge y = 3)$$

$$(x = 2 \wedge y = 4) \longrightarrow (x \leftarrow 4) \longrightarrow (x = 5 \wedge y = 4) \longrightarrow (y \leftarrow 2) \longrightarrow (x = 4 \wedge y = 2)$$

If a pomset represents an isolated program, an action has an effect on its successor. If that action is of the form $v \leftarrow n$, then the sucessor should assign n to v; likewise, if the action is of the form $v \leftarrow v'$, then the successor should assign the value of v' to v, but the predecessor should also be aware of a value for v'.

(A5) If $u \in S_{\mathbf{u}}$ such that $\lambda_{\mathbf{u}}(u) = v \leftarrow n$ for some $v \in \textsc{Var}$ and $n \in \textsc{Val}$, we require that the successor of u is s s.t. $\lambda_{\mathbf{u}}(s)(v) = n$.

(A6) Let $u \in S_{\mathbf{u}}$ s.t. $\lambda_{\mathbf{u}}(u) = v \leftarrow v'$ for some $v, v' \in \textsc{Var}$ and let p and s be the predecessor, resp. successor, of u. Then $v' \in \mathsf{dom}(\lambda_{\mathbf{u}}(p))$ and $\lambda_{\mathbf{u}}(s)(v) = \lambda_{\mathbf{u}}(s)(v') = \lambda_{\mathbf{u}}(p)(v')$.

▶ **Example 5.7.** The pomset below on the left violates (A5), because the successor of $(x \leftarrow 1)$ does not assign 2 to x. On the other hand, the pomset on the right satisfies (A6), because the predecessor of $(x \leftarrow y)$ has a value for y, and that value is assigned to x in the successor.

$$(x = 0) \longrightarrow (x \leftarrow 1) \longrightarrow (x = 2) \qquad (x = 1 \wedge y = 2) \longrightarrow (x \leftarrow y) \longrightarrow (x = 2 \wedge y = 2)$$

Finally, isolated programs cannot observe variables that have not been assigned a value anywhere in the program. On the pomset-level, this translates to:

(A7) Let $u \in S_{\mathbf{u}}$ be a state-node. Then for all $v \in \mathsf{dom}(\lambda_{\mathbf{u}}(u))$, there exists a path for v from $s \in S_{\mathbf{u}}$ to u such that either $v \in \mathsf{dom}(\lambda_{\mathbf{u}}(s))$ and $s = *_{\min}$ or s is the successor of an assignment-node with label $v \leftarrow k$ with $k \in \textsc{Var} \cup \textsc{Val}$.

▶ **Example 5.8.** The pomset on the left does not satisfy (A7), but the one on the right does.

$$(x = 1) \longrightarrow (x \leftarrow 2) \longrightarrow (x = 2 \wedge y = 2) \qquad (x = 1) \longrightarrow (y \leftarrow 2) \longrightarrow (x = 1 \wedge y = 2)$$

Guarded pomsets satisfy (A1)–(A7). In fact, there exists an equivalence:

▶ **Theorem 5.9.** *For $U \in \mathsf{SP}$, U is guarded if and only if U satisfies (A1)–(A7).*

Sketch. The forward implication is proved by induction on the construction of $\mathcal{G}$. For the other direction, we perform induction on the size of U (which is possible because U is series-parallel and therefore finite). The induction hypothesis then states that whenever V is strictly smaller than U, and V satisfies the seven properties, then V is guarded. Since U satisfies (A1), we know that either it consists of one node labelled by a state, in which case U is immediately guarded, or $U = \alpha \cdot V \cdot \beta$ for $\alpha, \beta \in \mathsf{State}$ and $V \in \mathsf{SP}$. This gives us four cases to consider: $V = 1$, $V = \mathsf{a}$ for some $\mathsf{a} \in \mathsf{Act} \cup \mathsf{State}$, $V = V_0 \cdot V_1$ or $V = V_0 \parallel V_1$. The first case can be disregarded as U would then violate (A2). In the second case, (A4)–(A7) can be used to show that $\beta = \alpha[\mathsf{a}]$. In the latter two cases we show that U is built out of two strictly smaller pomsets that satisfy (A1)–(A7), making them guarded by the induction hypothesis. When these two pomsets are combined to form U, this is done according to the rules of guarded pomsets, making U guarded as well. The details can be found in the full version of this paper [22]. ◀

6 Litmus Test

The POCKA-semantics of a program captures all *possible* behaviours of the program, including all behaviours that could arise when it is put in parallel with other programs. In this section we look at the behaviour of the litmus test when it is executed in isolation. In the previous section we saw that if a pomset represents an execution of a program in isolation, it is guarded, and hence it is sufficient to look at the guarded pomsets. We demonstrate that there are in fact no guarded pomsets in the semantics of the litmus test, which shows that it passes. This suggests the guarded fragment of the POCKA-semantics is sequentially consistent: the programs behave as if memory accesses performed concurrently are interleaved and executed sequentially and writes to memory are broadcasted to all threads instantaneously.

Recall the litmus test t we considered before, with $\textsc{Var} = \{x, y, r_0, r_1\}$ and $\textsc{Val} = \{0, 1\}$:

$$t := (r_0 = 0 \wedge r_1 = 0) \cdot ((x \leftarrow 1 \cdot r_0 \leftarrow y) \parallel (y \leftarrow 1 \cdot r_1 \leftarrow x)) \cdot \overline{(r_0 = 1 \vee r_1 = 1)}$$

Our strategy for showing that the semantics of t does not contain guarded pomsets, is to first show that all pomsets in the semantics of t have certain property. We then claim that if a pomset has this property, then it is not guarded, using (A1)–(A7) from Section 5.

▶ **Definition 6.1** (Litmus Pomsets). *Let $x, y, r_0, r_1, w \in \textsc{Var}$ be distinct and $0, 1 \in \textsc{Val}$. A pomset $U = [\mathbf{u}]$ has property P, denoted $P(U)$, if there exists $u_1, u_2, v_1, v_2, w \in S_{\mathbf{u}}$ s.t.*

1. *the following conditions hold:*

$$\lambda_{\mathbf{u}}(u_1) = (x \leftarrow 1) \qquad \lambda_{\mathbf{u}}(u_2) = (y \leftarrow 1) \qquad \lambda_{\mathbf{u}}(v_1) = (r_0 \leftarrow y) \qquad \lambda_{\mathbf{u}}(v_2) = (r_1 \leftarrow x)$$

$$\lambda_{\mathbf{u}}(w)(r_0) = 0 = \lambda_{\mathbf{u}}(w)(r_1) \qquad u_1 \leq_{\mathbf{u}} v_1 \leq_{\mathbf{u}} w \qquad u_2 \leq_{\mathbf{u}} v_2 \leq_{\mathbf{u}} w$$

Graphically, we can represent these conditions as the following diagram:

$$\begin{array}{ccc} u_1 : x \leftarrow 1 & \longrightarrow & v_1 : r_0 \leftarrow x \\ & \searrow & \\ & & w : \left[\begin{array}{c} r_0 \mapsto 0 \\ r_1 \mapsto 0 \end{array}\right] \\ & \nearrow & \\ u_2 : y \leftarrow 1 & \longrightarrow & v_2 : r_1 \leftarrow y \end{array}$$

2. *For other assignment-nodes in U, we have the following conditions. Let $k \in \textsc{Val} \cup \textsc{Var}$.*

$$\forall z. \lambda_{\mathbf{u}}(z) = (x \leftarrow k) \Rightarrow z \leq_{\mathbf{u}} u_1 \qquad \forall z. \lambda_{\mathbf{u}}(z) = (y \leftarrow k) \Rightarrow z \leq_{\mathbf{u}} u_2$$

$$\forall z. \lambda_{\mathbf{u}}(z) = (r_0 \leftarrow k) \Rightarrow z \leq_{\mathbf{u}} v_1 \qquad \forall z. \lambda_{\mathbf{u}}(z) = (r_1 \leftarrow k) \Rightarrow z \leq_{\mathbf{u}} v_2$$

The property P describes the actions and observations found in the litmus test, and their relative ordering. For instance, $\forall z.\lambda_{\mathbf{u}}(z) = (x \leftarrow n) \Rightarrow z \leq_{\mathbf{u}} u_1$ states that all action-nodes that change the value of x, occur before node u_1 Hence, the maximal node that alters the value of x, changes x to 1. The other requirements are explained similarly.

▶ **Lemma 6.2.** *Let $U = [\mathbf{u}] \in \mathsf{SP}$. If $P(U)$ then U is not guarded.*

We show that P is an invariant under closure w.r.t. exch and contr. To this end, it is useful to study the effect of the contraction order on the level of pomsets; we introduce the following partial order relation on pomsets, analogous to the subsumption order.

▶ **Definition 6.3** (Contraction Order)**.** *Let $U = [\mathbf{u}]$ and $V = [\mathbf{v}]$ be pomsets over $\mathsf{Act} \cup \mathsf{State}$. We write $U \preceq V$ holds iff there exists a surjection $h\colon S_{\mathbf{v}} \to S_{\mathbf{u}}$ satisfying: (i) $\lambda_{\mathbf{u}} \circ h = \lambda_{\mathbf{v}}$; (ii) $v \leq_{\mathbf{v}} v'$ implies $h(v) \leq_{\mathbf{u}} h(v')$; (iii) if $h(v) \leq_{\mathbf{u}} h(v')$, then $\lambda_{\mathbf{v}}(v), \lambda_{\mathbf{v}}(v') \in \mathsf{State}$ implies $v \leq_{\mathbf{v}} v'$ or $v' \leq_{\mathbf{v}} v$, and $\lambda_{\mathbf{v}}(v)$ or $\lambda_{\mathbf{v}}(v') \notin \mathsf{State}$ implies $v \leq_{\mathbf{v}} v'$.*

We then prove the analogue of Lemma 2.11, relating $\preceq$ to closure w.r.t. contr as follows.

▶ **Lemma 6.4.** *Let $L \subseteq \mathsf{SP}$ and $U \in \mathsf{SP}$. Now $U \in L{\downarrow^{\mathsf{contr}}}$ iff $U \preceq V$ for some $V \in L$.*

With this characterisation in hand, we can prove that P is invariant under closure.

▶ **Lemma 6.5.** *Let $e \in \mathcal{T}$. If $\forall U \in (\!|e|\!)$ we have $P(U)$, then $\forall V \in (\!|e|\!)\downarrow$ it holds that $P(V)$.*

Sketch. By [12, Lemma 5.4] we know that $(\!|e|\!)\downarrow = \left((\!|e|\!)\downarrow^{\mathsf{exch}}\right)\downarrow^{\mathsf{contr}}$. It then follows, by Lemma 2.11 and Lemma 6.4, that if $V \in (\!|e|\!)\downarrow$, then there must exist $W, X \in \mathsf{SP}$ with $X \in (\!|e|\!)$ and $V \preceq W \sqsubseteq X$. We then show that P is preserved by both of these orders. ◀

▶ **Corollary 6.6.** *The semantics of the litmus test contains no guarded pomsets: $(\!|t|\!)\downarrow \cap \mathcal{G} = \emptyset$.*

Proof. All pomsets in $(\!|t|\!)$ have property P if we pick for u_1 the node with label $(x \leftarrow 1)$, for v_1 the node with label $(r_0 \leftarrow y)$, and same for u_2 and v_2 (see Example 3.13). Lastly, we pick for w the node with label δ. By Lemma 6.5 we can conclude that all pomsets in $(\!|t|\!)\downarrow$ have property P, and by Lemma 6.2 we infer that t has no guarded pomsets in its semantics. ◀

We showed that we can correctly analyse the litmus test in our algebraic framework. In the next example we show that addition of one extra axiom, which is a commonly made assumption in programming languages, makes the litmus test fail on the guarded semantics.

▶ **Example 6.7.** We add the following axiom, which states that assignments to different variables can be swapped as long as the assigned values are none of the involved variables:

$$v \leftarrow k \cdot v' \leftarrow k' \equiv v' \leftarrow k' \cdot v \leftarrow k \text{ for } v, v' \in \textsc{Var},\ k, k' \in \textsc{Var} \cup \textsc{Val},\ k' \neq v \neq v',\ k \neq v \neq v'$$

We show that with this assumption, which is commonly made in programming languages, we get guarded pomsets in the semantics of the litmus program. We can derive:

$$\begin{aligned}
((r_0 \leftarrow y) \cdot (r_1 \leftarrow x)) &\equiv ((r_0 \leftarrow y) \parallel 1) \cdot (1 \parallel (r_1 \leftarrow x)) && \text{(Unit axiom)} \\
&\leqq ((r_0 \leftarrow y) \cdot 1) \parallel (1 \cdot (r_1 \leftarrow x)) && \text{(Exchange Law)} \\
&\equiv (r_0 \leftarrow y) \parallel (r_1 \leftarrow x) && \text{(Unit axiom)}
\end{aligned}$$

Similarly, we can derive that $(x \leftarrow 1) \cdot (y \leftarrow 1) \leqq (x \leftarrow 1) \parallel (y \leftarrow 1)$. Hence, we have

$$\begin{aligned}
&((r_0 \leftarrow y) \cdot (r_1 \leftarrow x)) \cdot ((x \leftarrow 1) \cdot (y \leftarrow 1)) \leqq ((r_0 \leftarrow y) \parallel (r_1 \leftarrow x)) \cdot ((x \leftarrow 1) \parallel (y \leftarrow 1)) \\
&\leqq ((r_0 \leftarrow y) \cdot (x \leftarrow 1)) \parallel ((r_1 \leftarrow x) \cdot (y \leftarrow 1)) && \text{(Exchange law)} \\
&\equiv ((x \leftarrow 1) \cdot (r_0 \leftarrow y)) \parallel ((y \leftarrow 1) \cdot (r_1 \leftarrow x)) && \text{(New axiom)}
\end{aligned}$$

Let $e = ((r_0 \leftarrow y) \cdot (r_1 \leftarrow x)) \cdot ((x \leftarrow 1) \cdot (y \leftarrow 1))$. We can conclude that

$$(r_0 = 0 \wedge r_1 = 0) \cdot e \cdot \overline{(r_0 = 1 \vee r_1 = 1)} \leqq t$$

From soundness, we infer that $(\!|(r_0 = 0 \wedge r_1 = 0) \cdot e \cdot \overline{(r_0 = 1 \vee r_1 = 1)}|\!)\downarrow \subseteq (\!|t|\!)\downarrow$. In the left set we find at least one guarded pomset. Let $\alpha = (r_0 = 0 \wedge r_1 = 0 \wedge x = 0 \wedge y = 0)$, $\beta = (r_0 = 0 \wedge r_1 = 0 \wedge x = 1 \wedge y = 0)$ and $\gamma = (r_0 = 0 \wedge r_1 = 0 \wedge x = 1 \wedge y = 1)$.
$\alpha \longrightarrow (r_0 \leftarrow y) \longrightarrow \alpha \longrightarrow (r_1 \leftarrow x) \longrightarrow \alpha \longrightarrow (x \leftarrow 1) \longrightarrow \beta \longrightarrow (y \leftarrow 1) \longrightarrow \gamma$

It is easy to show that this pomset is guarded by observing that $\alpha \cdot (r_0 \leftarrow y) \cdot \alpha$, $\alpha \cdot (r_1 \leftarrow x) \cdot \alpha$, $\alpha \cdot (x \leftarrow 1) \cdot \beta$ and $\beta \cdot (y \leftarrow 1) \cdot \gamma$ are all guarded. Hence, by adding this one extra axiom, we find guarded pomsets in the semantics of the litmus test, meaning that this axiom breaks sequential consistency.

7 Discussion

We presented POCKA, a sound and complete algebraic framework that can be used to analyse concurrent programs that manipulate variables. We identified the guarded fragment of the semantics, and showed this fragment captures the behaviour of programs executing in isolation. We demonstrated reasoning in POCKA by analysing a litmus test, also suggesting that the guarded fragment of the POCKA-semantics is sequentially consistent.

This work is built on Kleene algebra and extensions thereof. It is closest to Concurrent Kleene algebra with Observations [11, 12], which was proposed to integrate concurrency with a form of tests (i.e., observations). We deviate from CKAO by using partial observations and accordingly changing the algebraic structure of observations (a PCDL instead of a Boolean algebra), and by incorporating explicit assignments and tests to manipulate variables. Programs such as the litmus test that we analyse in POCKA are outside the scope of CKAO.

The idea of using a PCDL and partial functions in the semantics comes from Jipsen and Moshier [10]. In the current paper we establish completeness w.r.t. the partial function model, which is missing in *loc. cit.* A further contrast is that POCKA includes as basic syntax atomic programs and assertions pertaining to variable assignment, as occur in the litmus test. The definition of guarded pomset that we used is close to the one proposed in [10]. We provided an extensive analysis of guarded pomsets and showed how they can be used to study concrete program behaviour: our new characterisation in terms of concrete properties of pomsets (Theorem 5.9) is essential for the analysis of the litmus test in Section 6.

We suggest three avenues for future research. Firstly, the concrete observations and assignments that we have used are reminiscent of NetKAT [2, 6], an algebraic framework based on Kleene algebra with tests that allows for reasoning about networks. POCKA is thus suggestive of a concurrent version of NetKAT, in which algebraic reasoning about concurrent networks could be studied. While NetKAT arises as a particular instance of KAT, POCKA is *not* an instance of its closest relative in the Kleene algebra family, CKAO, due to the aforementioned move from an *arbitrary* Boolean algebra of observations to a *concrete* PCDL. It would therefore be of interest to formulate the necessary metatheory for the analogous framework of CKA with partial observations (where partial observations are given by an arbitrary PCDL), and situate POCKA within it.

This naturally leads to a third line of research. We have used the CKAH framework to obtain a completeness proof, and it turned out that the proof technique was perfectly amenable to a replacement of the Boolean algebra structure of observations with our observation algebra. This raises the question: which conditions are necessary on the algebraic structure of observations to be able to prove completeness in a similar manner? In particular, what

conditions are needed for a result similar to Lemma 4.1 to hold? Our conjecture is that the observation algebra needs to be such that all elements can be written as a finite sum of join-irreducible elements of the algebra (cf. Lemma 3.7).

References

1 Jade Alglave, Luc Maranget, Susmit Sarkar, and Peter Sewell. Litmus: Running tests against hardware. In *TACAS*, pages 41–44, 2011. doi:10.1007/978-3-642-19835-9_5.

2 Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: semantic foundations for networks. In *POPL*, pages 113–126, 2014. doi:10.1145/2535838.2535862.

3 Thomas S. Blyth. *Lattices and Ordered Algebraic Structures*. Springer-Verlag London, 2005.

4 John Horton Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, Ltd., London, 1971.

5 Amina Doumane, Denis Kuperberg, Damien Pous, and Pierre Pradic. Kleene algebra with hypotheses. In *FOSSACS*, pages 207–223, 2019. doi:10.1007/978-3-030-17127-8_12.

6 Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. A coalgebraic decision procedure for NetKAT. In *POPL*, pages 343–355, 2015. doi:10.1145/2676726.2677011.

7 Jay L. Gischer. The equational theory of pomsets. *Theor. Comput. Sci.*, 61:199–224, 1988. doi:10.1016/0304-3975(88)90124-7.

8 Jan Grabowski. On partial languages. *Fundam. Inform.*, 4(2):427, 1981.

9 Tony Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. Concurrent Kleene algebra. In *CONCUR*, pages 399–414, 2009. doi:10.1007/978-3-642-04081-8_27.

10 Peter Jipsen and M. Andrew Moshier. Concurrent Kleene algebra with tests and branching automata. *J. Log. Algebr. Meth. Program.*, 85(4):637–652, 2016. doi:10.1016/j.jlamp.2015.12.005.

11 Tobias Kappé, Paul Brunet, Jurriaan Rot, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. Kleene algebra with observations. In *CONCUR*, pages 41:1–41:16, 2019. doi:10.4230/LIPIcs.CONCUR.2019.41.

12 Tobias Kappé, Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. Concurrent Kleene algebra with observations: From hypotheses to completeness. In *FOSSACS*, pages 381–400, 2020. doi:10.1007/978-3-030-45231-5_20.

13 Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Inf. Comput.*, 110(2):366–390, 1994. doi:10.1006/inco.1994.1037.

14 Dexter Kozen. Kleene algebra with tests and commutativity conditions. In *TACAS*, pages 14–33, 1996. doi:10.1007/3-540-61042-1_35.

15 Dexter Kozen. On Hoare logic and Kleene algebra with tests. *ACM Trans. Comput. Log.*, 1(1):60–76, 2000. doi:10.1145/343369.343378.

16 Daniel Krob. A complete system of B-rational identities. In *ICALP*, pages 60–73, 1990. doi:10.1007/BFb0032022.

17 Leslie Lamport. How to make a correct multiprocess program execute correctly on a multiprocessor. *IEEE Trans. Computers*, 46(7):779–782, 1997. doi:10.1109/12.599898.

18 Michael R. Laurence and Georg Struth. Completeness theorems for bi-Kleene algebras and series-parallel rational pomset languages. In *RAMiCS*, pages 65–82, 2014. doi:10.1007/978-3-319-06251-8_5.

19 Kamal Lodaya and Pascal Weil. Series-parallel languages and the bounded-width property. *Theoretical Computer Science*, 237(1):347–380, 2000. doi:10.1016/S0304-3975(00)00031-1.

20 John C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. In *LiCS*, July 2002. doi:10.1109/LICS.2002.1029817.

21 Arto Salomaa. Two complete axiom systems for the algebra of regular events. *J. ACM*, 13(1):158–169, 1966. doi:10.1145/321312.321326.

22 Jana Wagemaker, Paul Brunet, Simon Docherty, Tobias Kappé, Jurriaan Rot, and Alexandra Silva. Partially observable concurrent Kleene algebra, 2020. arXiv:2007.07593.

A Proofs about observation algebra

▶ **Lemma 3.5** (Soundness OA). *For all $p, q \in \mathcal{O}$, if $p \equiv q$ then $[\![p]\!] = [\![q]\!]$.*

Proof. The fact that OA is a PCDL and the assignment $[\![-]\!]$ is well-defined establishes the soundness of the PCDL axioms. We thus only have the domain-specific axioms left to verify.

- If $n \neq m$, it is immediate that

$$[\![v = n \wedge v = m]\!] = [\![v = n]\!] \cap [\![v = m]\!] = \{\alpha \mid \alpha(v) = n\} \cap \{\alpha \mid \alpha(v) = m\} = \emptyset = [\![\bot]\!]$$

- Next we show $[\![\overline{v = n}]\!] \subseteq \left[\!\!\left[\bigvee_{m \neq n} v = m\right]\!\!\right]$. Assume

$$\alpha \in [\![\overline{v = n}]\!] = \bigcup \{B \in P_{\leq}(\mathsf{State}) \mid B \cap [\![v = n]\!] = \emptyset\}.$$

  We have some downwards-closed $B \subseteq \mathsf{State}$ such that $\alpha \in B$ and $B \cap [\![v = n]\!] = \emptyset$. Suppose towards a contradiction that $\alpha(v)$ is undefined, or that $\alpha(v) = n$. We can then choose $\alpha' \in \mathsf{State}$ to be n on v, and identical to α elsewhere; in that case, $\alpha' \leq \alpha$, which means that $\alpha' \in B$. But then, since $\alpha' \in [\![v = n]\!]$ by construction, we have a contradiction with the fact that $B \cap [\![v = n]\!] = \emptyset$. Thus, there exists an $m \in \textsc{Val}$ such that $\alpha(v) = m$ and $m \neq n$. It then follows that $\alpha \in \left[\!\!\left[\bigvee_{m \neq n} v = m\right]\!\!\right]$.

- Next we prove that $\left[\!\!\left[\overline{\bigwedge_i v_i = n_i}\right]\!\!\right] \subseteq [\![\bigvee_i \overline{v_i = n_i}]\!]$, if the v_i are distinct. Let $\alpha \in B$ such that $B \cap [\![\bigwedge_i v_i = n_i]\!] = \emptyset$. We claim that for some i, $\alpha(v_i) = m \neq n_i$. Suppose otherwise: then for each v_i either $\alpha(v_i) = n_i$ or $\alpha(v_i)$ is undefined. Define:

$$\alpha'(v) ::= \begin{cases} n_i & \text{if } v = v_i \text{ and } \alpha(v_i) \text{ undefined;} \\ \alpha(v) & \text{otherwise.} \end{cases}$$

  This is well-defined by the assumption that the v_i are all distinct. By construction, $\alpha' \leq \alpha$ and $\alpha' \in \bigcap_i \{\beta \mid \beta(v_i) = n_i\} = [\![\bigwedge_i v_i = n_i]\!]$. As B is downwards-closed, we get $\alpha' \in B$, contradicting $B \cap [\![\bigwedge_i v_i = n_i]\!] = \emptyset$. Hence for some i, $\alpha(v_i) = m \neq n_i$. Hence $\alpha \in [\![\overline{v_i = n_i}]\!] \subseteq [\![\bigvee_i \overline{v_i = n_i}]\!]$ as required.

For the inductive step, we verify that the closure rules for congruence preserve soundness. This is all immediate from the definition of $[\![-]\!]$. For instance, if $e = e_0 \vee e_1$, $f = f_0 \vee f_1$, $e_0 \equiv f_0$ and $e_1 \equiv f_1$, then $[\![e]\!] = [\![e_0]\!] \cup [\![e_1]\!] = [\![f_0]\!] \cup [\![f_1]\!] = [\![f]\!]$, where we have used that $[\![e_0]\!] = [\![f_0]\!]$ and $[\![e_1]\!] = [\![f_1]\!]$ by the induction hypothesis. ◀

▶ **Lemma 3.6.** *For all $\alpha, \beta \in \mathsf{State}$: $\alpha \in [\![\pi_\beta]\!]$ iff $\alpha \leq \beta$ iff $\pi_\alpha \leqq \pi_\beta$.*

Proof. Assume $\alpha \in [\![\pi_\beta]\!]$. Then for all $v \in \mathsf{dom}(\beta)$, $\alpha(v)$ is defined and $\alpha(v) = \beta(v)$, hence $\alpha \leq \beta$. Assume $\alpha \leq \beta$. Then $\pi_\alpha \leqq \pi_\beta$ is established from $\pi_\alpha \wedge \pi_\beta \equiv \pi_\alpha$: by the assumption, every conjunct in $\bigwedge_{\beta(v)=n} v = n$ appears as a conjunct in $\bigwedge_{\alpha(v)=n} v = n$, so by idempotence $\pi_\alpha \wedge \pi_\beta \equiv \left(\bigwedge_{\alpha(v)=n} v = n\right) \wedge \left(\bigwedge_{\beta(v)=n} v = n\right) \equiv \bigwedge_{\alpha(v)=n} v = n \equiv \pi_\alpha$. Finally, assume $\pi_\alpha \leqq \pi_\beta$. By soundness $[\![\pi_\alpha]\!] \subseteq [\![\pi_\beta]\!]$, and it is trivial to establish $\alpha \in [\![\pi_\alpha]\!]$. ◀

▶ **Lemma 3.7.** *For all $p \in \mathcal{O}$, we have $p \equiv \bigvee \{\alpha \in \mathsf{State} \mid \pi_\alpha \leqq p\}$.*

Proof. Noting that $\bigvee \{\alpha \in \mathsf{State} \mid \alpha \leqq p\} \leqq p$ by definition, we focus on the other inequality, proceeding by induction on p. For the base cases, $\bot \leqq \bigvee \{\alpha \in \mathsf{State} \mid \alpha \leqq \bot\}$ by definition; $\top \leqq \bigvee \{\alpha \in \mathsf{State} \mid \alpha \leqq \top\}$ as $\top \equiv \pi_\emptyset \in \{\alpha \in \mathsf{State} \mid \alpha \leqq \top\}$; and $v = n \leqq \bigvee \{\alpha \in \mathsf{State} \mid \alpha \leqq v = n\}$ as $v = n \equiv \pi_{\{v \mapsto n\}} \in \{\alpha \in \mathsf{State} \mid \alpha \leqq v = n\}$.

In the induction step we have three cases.

- If $p = p_0 \wedge p_1$, by the inductive hypothesis and distributivity we obtain

$$p_0 \wedge p_1 \leqq \bigvee\{\alpha \wedge \beta \mid \alpha \leqq p_0, \beta \leqq p_1\}.$$

We claim that $\{\alpha \wedge \beta \mid \alpha \leqq p_0, \beta \leqq p_1\} \subseteq \{\alpha \mid \alpha \leqq p_0 \wedge p_1\} \cup \{\bot\}$. Call $\alpha, \beta \in \mathsf{State}$ *compatible* if, for any $v \in \mathsf{dom}(\alpha) \cap \mathsf{dom}(\beta)$, $\alpha(v) = \beta(v)$. Take $\alpha \leqq p_0$ and $\beta \leqq p_1$. There are two cases. In the first, α and β are compatible. Then define γ by

$$\gamma(v) ::= \begin{cases} \alpha(v) & \text{if } \alpha(v) \text{ defined;} \\ \beta(v) & \text{if } \beta(v) \text{ defined;} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

This is well-defined by compatability. Then $\gamma \leq \alpha$ as well as $\gamma \leq \beta$, and hence by Lemma 3.6 we find $\gamma \leqq \alpha \wedge \beta \leqq p_0 \wedge p_1$. In the other case, α and β are not compatible: hence for some distinct n and m, $v = n$ and $v = m$ are among the conjuncts of $\alpha \wedge \beta$. By the axiom $v = n \wedge v = m \equiv \bot$, it then follows that $\alpha \wedge \beta \equiv \bot$. We obtain

$$\bigvee\{\alpha \wedge \beta \mid \alpha \leqq p_0, \beta \leqq p_1\} \leqq \bigvee(\{\alpha \mid \alpha \leqq p_0 \wedge p_1\} \cup \{\bot\}) \equiv \bigvee\{\alpha \mid \alpha \leqq p_0 \wedge p_1\}.$$

- If $p = p_0 \vee p_1$, we derive

$$\begin{aligned} p_0 \vee p_1 &\leqq \bigvee\{\alpha \in \mathsf{State} \mid \alpha \leqq p_0\} \vee \bigvee\{\beta \in \mathsf{State} \mid \beta \leqq p_1\} && \text{(IH)} \\ &\leqq \bigvee\{\alpha \in \mathsf{State} \mid \alpha \leqq p_1 \vee p_2\} && (\alpha \leqq p_0 \leqq p_0 \vee p_1, \text{ similar for } \beta) \end{aligned}$$

- If $p = \overline{p_0}$, we derive

$$\begin{aligned} \overline{p_0} &\equiv \overline{\bigvee\{\alpha \mid \alpha \leqq p_0\}} && \text{(IH)} \\ &\equiv \bigwedge\{\overline{\alpha} \mid \alpha \leqq p_0\} && \text{(De Morgan)} \\ &\equiv \bigwedge\{\overline{\bigwedge_{\alpha(v)=n} v = n} \mid \alpha \leqq p_0\} && \text{(Definition of } \pi_\alpha = \alpha) \\ &\leqq \bigwedge\{\bigvee_{\alpha(v)=n} \overline{v = n} \mid \alpha \leqq p_0\} && \text{(De Morgan-like domain-specific axiom)} \\ &\leqq \bigwedge\{\bigvee_{\substack{\alpha(v)=n \\ m \neq n}} v = m \mid \alpha \leqq p_0\} && \text{(Pseudocomplement domain-specific axiom)} \end{aligned}$$

Note that the De Morgan law applied in the second step is indeed satisfied by PCDLs [3]. Now, define $K ::= \{\alpha \in \mathsf{State} \mid \alpha \leqq p_0\}$, $J_\alpha ::= \{(v, m) \mid \alpha(v) = n \neq m\}$, $J ::= \bigcup_{\alpha \in K} J_\alpha$ and $F ::= \{f : K \to J \mid \forall \alpha \in K, f(\alpha) \in J_\alpha\}$. Further, let

$$p_{\alpha,(v,m)} ::= \begin{cases} v = m & \text{if } \alpha(v) = n \neq m; \\ \bot & \text{otherwise} \end{cases}$$

Then

$$\bigwedge\{\bigvee_{\substack{\alpha(v)=n \\ m \neq n}} v = m \mid \alpha \leqq p_0\} \equiv \bigwedge_{\alpha \in K} \bigvee_{(v,m) \in J_\alpha} p_{\alpha,(v,m)} \equiv \bigvee_{f \in F} \bigwedge_{\alpha \in K} p_{\alpha, f(\alpha)}$$

by distributivity. For each $f \in F$, if the $p_{\alpha,f(\alpha)}$ are compatible, $\bigwedge_{\alpha\in K} p_{\alpha,f(\alpha)} \equiv \beta_f$, for a β_f with the property that for every $\alpha \leqq p_0$, $\beta_f \wedge \alpha = \bot$ (as by definition, for each such α, β_f has some $v = m$ as a conjunct, where α has a conjunct $v = n$ for $n \neq m$). If they are incompatible, $\bigwedge_{\alpha\in K} p_{\alpha,f(\alpha)} \equiv \bot$. Hence

$$\bigvee_{f\in F} \bigwedge_{\alpha\in K} p_{\alpha,f(\alpha)} \leqq \bigvee\{\beta \mid \text{for all } \alpha, \alpha \leqq p_0 \text{ implies } \alpha \wedge \beta \equiv \bot\}.$$

For any β satisfying the property that for all $\alpha, \alpha \leqq p_0$ implies $\alpha \wedge \beta \equiv \bot$, we have $\beta \wedge p_0 \equiv \beta \wedge \bigvee\{\alpha \mid \alpha \leqq p_0\} \equiv \bigvee\{\alpha \wedge \beta \mid \alpha \leqq p_0\} \equiv \bot$, so $\beta \leqq \overline{p_0}$ and

$$\bigvee\{\beta \mid \text{for all } \alpha \leqq p_0 \text{ implies } \alpha \wedge \beta \equiv \bot\} \leqq \bigvee\{\beta \mid \beta \leqq \overline{p_0}\},$$

completing the proof. ◀

▶ **Theorem 3.8** (Completeness OA). *For all $p, q \in \mathcal{O}$, we have $p \equiv q$ if and only if $[\![p]\!] = [\![q]\!]$.*

Proof. The left-to-right direction follows from Lemma 3.5. For the right-to-left direction, suppose that $[\![p]\!] = [\![q]\!]$. By Lemma 3.7, we obtain that

$$[\![p]\!] = \left[\!\!\left[\bigvee\{\alpha \in \mathsf{State} \mid \alpha \leqq p\}\right]\!\!\right] = \left[\!\!\left[\bigvee\{\beta \in \mathsf{State} \mid \beta \leqq q\}\right]\!\!\right] = [\![q]\!].$$

We prove $\alpha \leqq p$ if and only if $\alpha \leqq q$. Take $\alpha \leqq p$. Then $\alpha \in [\![\alpha]\!] \subseteq [\![p]\!] = \bigcup_{\beta\leqq q} [\![\beta]\!]$ by Lemmas 3.5 and 3.6. Hence for some $\beta \leqq q$, $\alpha \in [\![\beta]\!]$, and by Lemma 3.6 once more, $\alpha \leqq \beta \leqq q$. The other direction is symmetric. It follows that

$$p \equiv \bigvee\{\alpha \in \mathsf{State} \mid \alpha \leqq p\} \equiv \bigvee\{\alpha \in \mathsf{State} \mid \alpha \leqq q\} \equiv q,$$

as required. ◀

B Proofs towards completeness

The following three results are all needed in the proofs that follow, and come from [12]. First of all, we can prove the following useful properties about the interaction between closure and other operators on pomset languages:

▶ **Lemma B.1.** *Let $L, K \subseteq \mathsf{Pom}$ and $C \in \mathsf{PC}$. The following hold.*

1. $L \subseteq K{\downarrow^H}$ *iff* $L{\downarrow^H} \subseteq K{\downarrow^H}$.
2. *If* $L \subseteq K$, *then* $L{\downarrow^H} \subseteq K{\downarrow^H}$.
3. $(L \cup K){\downarrow^H} = \left(L{\downarrow^H} \cup K{\downarrow^H}\right){\downarrow^H}$
4. $(L \cdot K){\downarrow^H} = \left(L{\downarrow^H} \cdot K{\downarrow^H}\right){\downarrow^H}$
5. $(L \parallel K){\downarrow^H} = \left(L{\downarrow^H} \parallel K{\downarrow^H}\right){\downarrow^H}$
6. $(L^*){\downarrow^H} = \left(\left(L{\downarrow^H}\right)^*\right){\downarrow^H}$
7. *If* $L{\downarrow^H} \subseteq K{\downarrow^H}$, *then* $C[L]{\downarrow^H} \subseteq C[K]{\downarrow^H}$.
8. *If* $L \subseteq \mathsf{SP}$, *then* $L{\downarrow^H} \subseteq \mathsf{SP}$.

Second, we can note the following result about the interaction between exch and contr:

▶ **Lemma B.2.** *For any $L \in 2^{\mathsf{SP}}$, we have $L{\downarrow^{\mathsf{contr}\cup\mathsf{exch}}} = (L{\downarrow^{\mathsf{exch}}}){\downarrow^{\mathsf{contr}}}$.*

We can then prove soundness of the POCKA semantics w.r.t. the axioms.

▶ **Theorem 3.14** (Soundness POCKA). *For all $e, f \in \mathcal{T}$, if $e \equiv f$ then $(\!|e|\!){\downarrow} = (\!|f|\!){\downarrow}$.*

Proof. By construction it is immediate that $(\!|-|\!){\downarrow}$ is closed under closure with respect to $\mathsf{exch} \cup \mathsf{contr}$. We then proceed by induction on $\equiv$. For all the pairs from $\equiv_{\mathsf{BKA}}$, it follows from Theorem 2.6 that $(\!|e|\!) = (\!|f|\!)$. Then immediately their POCKA-semantics also coincide. For all the pairs from $\equiv_{\mathsf{OA}}$, we make use of Theorem 3.8. Note that $(\!|-|\!){\downarrow}$ almost coincides

with $[\![-]\!]_{\mathsf{OA}}$ on $\mathcal{O}$, so the proof is very straightforward. For instance, take $p \vee (p \wedge q) \equiv_{\mathsf{OA}} p$. Then $(\!|p \vee (p \wedge q)|\!) = \mathsf{State}^* \cdot [\![p \vee (p \wedge q)]\!]_{\mathsf{OA}} \cdot \mathsf{State}^*$. From Theorem 3.8, we know that $[\![p \vee (p \wedge q)]\!]_{\mathsf{OA}} = [\![p]\!]_{\mathsf{OA}}$, and thus we obtain $(\!|p \vee (p \wedge q)|\!) = \mathsf{State}^* \cdot [\![p]\!]_{\mathsf{OA}} \cdot \mathsf{State}^* = (\!|p|\!)$. Then from this we can conclude that $(\!|p \vee (p \wedge q)|\!)\downarrow = (\!|p \vee (p \wedge q)|\!)\downarrow^{\mathsf{exch} \cup \mathsf{contr}} = (\!|p|\!)\downarrow^{\mathsf{exch} \cup \mathsf{contr}} = (\!|p|\!)\downarrow$. We can prove soundness of the other observation algebra axioms analogously.

The next axiom is the exchange law. We show that $(\!|(e \parallel f) \cdot (g \parallel h)|\!)\downarrow \subseteq (\!|(e \cdot g) \parallel (f \cdot h)|\!)\downarrow$. By Lemma B.1(1), it suffices to prove that $(\!|(e \parallel f) \cdot (g \parallel h)|\!) \subseteq (\!|(e \cdot g) \parallel (f \cdot h)|\!)\downarrow$. Take an element in $(\!|(e \parallel f) \cdot (g \parallel h)|\!)$. This is thus a pomset of the form $(X \parallel Y) \cdot (V \parallel W)$ for $X \in (\!|e|\!)$, $Y \in (\!|f|\!)$, $V \in (\!|g|\!)$ and $W \in (\!|h|\!)$. Thus we immediately obtain that $(X \cdot V) \parallel (Y \cdot W) \in (\!|(e \cdot g) \parallel (f \cdot h)|\!)$. From Lemma B.2, we know that $(\!|-|\!)\downarrow = ((\!|-|\!)\downarrow^{\mathsf{exch}})\downarrow^{\mathsf{contr}}$. We know that $(X \parallel Y) \cdot (V \parallel W) \sqsubseteq (X \cdot V) \parallel (Y \cdot W)$ and that $(X \cdot V) \parallel (Y \cdot W) \in ((\!|(e \cdot g) \parallel (f \cdot h)|\!)\downarrow^{\mathsf{exch}})\downarrow^{\mathsf{contr}}$. Then we can apply Lemmas 2.11 and B.2 to obtain that $(X \parallel Y) \cdot (V \parallel W) \in (\!|(e \cdot g) \parallel (f \cdot h)|\!)\downarrow$.

Left to verify are the interface axioms. To check $p \wedge q \leqq p \cdot q$ it again suffices to prove that $(\!|p \wedge q|\!) \subseteq (\!|p \cdot q|\!)\downarrow$ by Lemma B.1(1). We take an element in $(\!|p \wedge q|\!)$. This a pomset of the form $U \cdot \alpha \cdot V$ such that $U, V \in \mathsf{State}^*$ and $\alpha \in [\![p]\!]_{\mathsf{OA}} \cap [\![q]\!]_{\mathsf{OA}}$. We can establish that $U \cdot \alpha \cdot \alpha \cdot V \in (\!|p \cdot q|\!)$. Now take the pomset context $C = U \cdot * \cdot V$. We have that $C[[\![\alpha \cdot \alpha]\!]] = \{U \cdot \alpha \cdot \alpha \cdot V\} \subseteq (\!|p \cdot q|\!) \subseteq (\!|p \cdot q|\!)\downarrow$. Then by closure we find $C[[\![\alpha]\!]] = \{U \cdot \alpha \cdot V\} \subseteq (\!|p \cdot q|\!)\downarrow$.

Since $(\!|\bot|\!) = \emptyset = (\!|0|\!)$, it follows that $(\!|\bot|\!)\downarrow = \emptyset = (\!|0|\!)\downarrow$. Similarly, since $(\!|p + q|\!) = (\!|p|\!) \cup (\!|q|\!) = (\!|p \vee q|\!)$, we also have that $(\!|p + q|\!)\downarrow = (\!|p \vee q|\!)\downarrow$

Now we have four axioms left. The first one we verify is $\top \cdot p \leqq p$ for $p \in \mathcal{O}$. We immediately obtain that $(\!|\top \cdot p|\!) = \mathsf{State} \cdot \mathsf{State}^* \cdot [\![p]\!]_{\mathsf{OA}} \cdot \mathsf{State}^* \subseteq \mathsf{State}^* \cdot [\![p]\!]_{\mathsf{OA}} \cdot \mathsf{State}^* = (\!|p|\!) \subseteq (\!|p|\!)\downarrow$. The axioms $\top \cdot p \leqq p$, $a \cdot \top \leqq a$, and $\top \cdot a \leqq a$ for $a \in \mathsf{Act}$ are all verified in a similar manner.

In the inductive step we need to check whether the closure rules for congruence have been preserved. We distinguish four cases.

- If $e = e_0 + e_1$ and $f = f_0 + f_1$ with $e_0 \equiv f_0$ and $e_1 \equiv f_1$, then by induction we know that $(\!|e_0|\!)\downarrow = (\!|f_0|\!)\downarrow$ and $(\!|e_1|\!)\downarrow = (\!|f_1|\!)\downarrow$. By Lemma B.1(3), we can then derive that

$$\begin{aligned}
(\!|e|\!)\downarrow &= ((\!|e_0|\!) \cup (\!|e_1|\!))\downarrow^{\mathsf{exch} \cup \mathsf{contr}} \\
&= ((\!|e_0|\!)\downarrow^{\mathsf{exch} \cup \mathsf{contr}} \cup (\!|e_1|\!)\downarrow^{\mathsf{exch} \cup \mathsf{contr}})\downarrow^{\mathsf{exch} \cup \mathsf{contr}} \\
&= ((\!|f_0|\!)\downarrow^{\mathsf{exch} \cup \mathsf{contr}} \cup (\!|f_1|\!)\downarrow^{\mathsf{exch} \cup \mathsf{contr}})\downarrow^{\mathsf{exch} \cup \mathsf{contr}} \\
&= ((\!|f_0|\!) \cup (\!|f_1|\!))\downarrow^{\mathsf{exch} \cup \mathsf{contr}} \\
&= (\!|f|\!)\downarrow^{\mathsf{exch} \cup \mathsf{contr}}
\end{aligned}$$

- The cases for $\cdot$, $\parallel$ and $*$ are argued similarly. ◀

▶ **Lemma 4.5.** *For all $e, f \in \mathcal{T}$, if $e \leq f \in \mathsf{top} \cup \mathsf{contr}$, then $e \leqq f$.*

Proof. If $e \leq f \in \mathsf{contr}$, then $e = \alpha$ and $f = \alpha \cdot \alpha$ for some $\alpha \in \mathsf{State}$. We can then derive that $\alpha \equiv \alpha \wedge \alpha \leqq \alpha \cdot \alpha$, using the PCDL and the Interface axioms. Hence $e \leqq f$. If $e \leq f \in \mathsf{top}$, then we distinguish two cases.

1. Let $e = \alpha \cdot c$ and $g = c$ for $c \in \mathsf{Act}$ and $\alpha \in \mathsf{State}$. Then we derive

$$\alpha \cdot c \leqq \top \cdot c \leqq c \qquad \text{(PCDL and Interface axioms)}$$

   The case where $e = c \cdot \alpha$ is similar.
2. Let $e = \alpha \cdot \beta$ and $g = \beta$ for $\alpha, \beta \in \mathsf{State}$. Then we derive

$$\alpha \cdot \beta \leqq \top \cdot \beta \leqq \beta \qquad \text{(PCDL and Interface axioms)}$$

   The case where $e = \beta \cdot \alpha$ is similar. Hence, we can conclude that $e \leqq f$. ◀

C Proofs about the litmus test

▶ **Lemma 6.2.** *Let $U = [\mathbf{u}] \in \mathsf{SP}$. If $P(U)$ then U is not guarded.*

Proof. We prove by contradiction; assume that $P(U)$ and that U is guarded. Via Theorem 5.9 we conclude that U satisfies (A1)–(A7). From $P(U)$ we infer that there exists $u_1, u_2, w \in S_{\mathbf{u}}$ such that $\lambda_{\mathbf{u}}(u_1) = (x \leftarrow 1)$, $u_1 \leq w$ and $\lambda_{\mathbf{u}}(w)(r_0) = 0 = \lambda_{\mathbf{u}}(w)(r_1)$. From (A2) and (A5), we infer that u_1 has a unique successor node $s_1 \in S_{\mathbf{u}}$ such that s_1 is state-labelled, and $\lambda_{\mathbf{u}}(s_1)(x) = 1$. From (A4) there exists a path for x from s_1 to w. Hence, if $\lambda_{\mathbf{u}}(w)(x) \neq 1$, there must be at least one assignment between s_1 and w altering the value of x, as the path must explain how the value of x changed from 1 to 0. Hence, there exists a node $u_3 \in S_{\mathbf{u}}$ such that $s_1 \leq_{\mathbf{u}} u_3 \leq_{\mathbf{u}} w$ and $\lambda_{\mathbf{u}}(u_3) = (x \leftarrow n)$ for $n \in \textsc{Var} \cup \textsc{Val}$. However, from property P, we know that all such assignments occur before u_1, and thereby strictly before s_1. From this we can conclude that $\lambda_{\mathbf{u}}(w)(x) = 1$. Similarly, we obtain $\lambda_{\mathbf{u}}(w)(y) = 1$.

From (A2) and (A6), we know that v_1 has a unique successor node t_1, such that $\lambda_{\mathbf{u}}(t_1)(r_0) = \lambda_{\mathbf{u}}(t_1)(y)$. Then from (A4), there must be a path for r_0 from t_1 to w. With similar reasoning as for x above, we obtain $\lambda_{\mathbf{u}}(t_1)(r_0) = \lambda_{\mathbf{u}}(w)(r_0) = 0$. Similarly, we obtain a successor node t_2 of v_2 such that $\lambda_{\mathbf{u}}(t_2)(r_1) = \lambda_{\mathbf{u}}(t_2)(x) = 0$.

As we have that $\lambda_{\mathbf{u}}(t_1)(y) = 0$ and $\lambda_{\mathbf{u}}(w)(y) = 1$ and $t_1 \leq_{\mathbf{u}} w$, we can conclude from (A4) that there must be a path from t_1 to w for y such that this path contains at least one assignment that alters the value for y. Thus, there exists a node u_3 such that $t_1 \leq_{\mathbf{u}} u_3 \leq_{\mathbf{w}} w$ and u_3 has a label that changes the value of y. Similarly, we obtain a node u_4 such that $t_2 \leq_{\mathbf{u}} u_4 \leq_{\mathbf{w}} w$ and u_4 changes the value of x. From property P, we obtain $u_3 \leq_{\mathbf{u}} u_2$ and $u_4 \leq_{\mathbf{u}} u_1$. Then, making use of the fact that t_1 and t_2 are the successors of v_1 and v_2 respectively, we can derive: $v_2 \leq_{\mathbf{u}} t_2 \leq_{\mathbf{u}} u_4 \leq_{\mathbf{u}} u_1 \leq_{\mathbf{u}} v_1 \leq_{\mathbf{u}} t_1 \leq_{\mathbf{u}} u_3 \leq_{\mathbf{u}} u_2 \leq_{\mathbf{u}} v_2$ Then, by antisymmetry, all these nodes are equivalent. As they cannot be, we have a contradiction. Hence, U is not a guarded pomset. Hence, U is not a guarded pomset. ◀

▶ **Lemma 6.5.** *Let $e \in \mathcal{T}$. If $\forall U \in (\!|e|\!)$ we have $P(U)$, then $\forall V \in (\!|e|\!)\downarrow$ it holds that $P(V)$.*

Proof. First, note that $(\!|e|\!)\downarrow = (\!|e|\!)\downarrow^{\mathsf{exch} \cup \mathsf{contr}} = \big((\!|e|\!)\downarrow^{\mathsf{exch}}\big)\downarrow^{\mathsf{contr}}$ by Lemma B.2. Thus, if $V \in (\!|e|\!)\downarrow$, we apply Lemma 6.4, to infer that there exists a pomset $W \in (\!|e|\!)\downarrow^{\mathsf{exch}}$ such that $V \preceq W$. Next we can apply Lemma 2.11, to obtain a pomset $U \in (\!|e|\!)$ such that $W \sqsubseteq U$. We know that U has property P. We first show that W also has property P, and then that the same holds for V. From the definition of $\sqsubseteq$ we get that there exists a bijective pomset morphism h from W to U. Thus we have $U = [\mathbf{u}]$ and $W = [\mathbf{w}]$ and a bijective function $h \colon S_{\mathbf{u}} \to S_{\mathbf{w}}$ such that $\lambda_{\mathbf{w}} \circ h = \lambda_{\mathbf{u}}$ and if $u \leq_{\mathbf{u}} u'$ then $h(u) \leq_{\mathbf{w}} h(u')$. Now we need to verify the two properties of Definition 6.1.

1. As $\lambda_{\mathbf{w}}(h(u_1)) = \lambda_{\mathbf{u}}(u_1)$, we get $\lambda_{\mathbf{w}}(h(u_1)) = (x \leftarrow 1)$. The same for the other existential statements of Item 1. For the ordering: from $u_1 \leq_{\mathbf{u}} v_1 \leq_{\mathbf{u}} w$ we immediately obtain that $h(u_1) \leq_{\mathbf{w}} h(v_1) \leq_{\mathbf{w}} h(w)$ and similarly for $h(u_2) \leq_{\mathbf{w}} h(v_2) \leq_{\mathbf{w}} h(w)$.
2. Take a z such that $\lambda_{\mathbf{w}}(z) = (x \leftarrow n)$ for $n \in \textsc{Val} \cup \textsc{Var}$. As h is surjective, we know there exists a node $s \in S_{\mathbf{u}}$ such that $h(s) = z$ and $\lambda_{\mathbf{w}}(h(s)) = \lambda_{\mathbf{u}}(s)$. As $P(U)$, we get that $s \leq_{\mathbf{u}} u_1$. Hence, $h(s) \leq_{\mathbf{w}} h(u_1)$ and thus $z \leq_{\mathbf{w}} h(u_1)$. An analogue argument can be given for the other conditions in property (2).

This demonstrates that W has property P. We know that $V \preceq W$, and we will show this implies that V also has property P. From the definition of $\preceq$ we know that there exists a pomset morphism h from V to W. The argument to verify the two properties of Definition 6.1 is exactly the same as above. Hence we can conclude that V has property P. ◀

Model-Free Reinforcement Learning for Stochastic Parity Games

Ernst Moritz Hahn
University of Twente, Enschede, The Netherlands
E.M.Hahn@utwente.nl

Mateo Perez
University of Colorado Boulder, CO, USA
Mateo.Perez@Colorado.EDU

Sven Schewe
University of Liverpool, UK
Sven.Schewe@liverpool.ac.uk

Fabio Somenzi
University of Colorado Boulder, CO, USA
Fabio@Colorado.EDU

Ashutosh Trivedi
University of Colorado Boulder, CO, USA
Asutosh.Trivedi@Colorado.EDU

Dominik Wojtczak
University of Liverpool, UK
D.Wojtczak@liverpool.ac.uk

Abstract

This paper investigates the use of model-free reinforcement learning to compute the optimal value in two-player stochastic games with parity objectives. In this setting, two decision makers, player Min and player Max, compete on a finite game arena – a stochastic game graph with unknown but fixed probability distributions – to minimize and maximize, respectively, the probability of satisfying a parity objective. We give a reduction from stochastic parity games to a family of stochastic reachability games with a parameter ε, such that the value of a stochastic parity game equals the limit of the values of the corresponding simple stochastic games as the parameter ε tends to 0. Since this reduction does not require the knowledge of the probabilistic transition structure of the underlying game arena, model-free reinforcement learning algorithms, such as minimax Q-learning, can be used to approximate the value and mutual best-response strategies for both players in the underlying stochastic parity game. We also present a streamlined reduction from $1\frac{1}{2}$-player parity games to reachability games that avoids recourse to nondeterminism. Finally, we report on the experimental evaluations of both reductions.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Computing methodologies → Machine learning algorithms; Mathematics of computing → Markov processes; Theory of computation → Convergence and learning in games

Keywords and phrases Reinforcement learning, Stochastic games, Omega-regular objectives

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.21

Funding This work has been supported by the National Natural Science Foundation of China (Grant Nr. 61532019), by the Engineering and Physical Sciences Research Council grants EP/M027287/1 and EP/P020909/1, by a CU Boulder Research and Innovation Office grant, and by the National Science Foundation grant 2009022.

1 Introduction

Reinforcement Learning (RL, [34]) is an optimization approach applicable to stochastic games with unknown but fixed probability distributions (unknown stochastic games). Users of RL must specify a scalar reward signal whose expected return should be maximized. Therefore, an objective to be satisfied has to be expressed in terms of scalar rewards, whose expected value is then maximized. Turning a specification manually into such an optimization problem is laborious and error prone [38, 22]. We therefore need an automatic conversion from specifications to rewards in order to leverage the power of a logical specification in RL.

Reinforcement learning approaches can broadly be classified as model-free [33] and model-based [12, 37]. Model-based approaches sample the environment and rewards to learn a model of the stochastic game, while model-free approaches aim to compute optimal strategies without explicitly estimating the transition probabilities and rewards. Model-free approaches to RL, such as Q-learning [34], are asymptotically space-efficient [33] and enable the use of universal approximation architectures, such as deep neural networks [13, 22], to store optimal strategies; they have been demonstrated to scale well [33, 32, 15]. The focus of this paper is to enable model-free RL to learn optimal strategies in unknown stochastic games with ω-regular objectives [29] given as parity objectives.

We study stochastic parity games with unknown transition structure, where two players – Player Min and Player Max – take turns to choose actions on a stochastic game arena (SGA) to form an infinite play. Each position-action pair of the SGA is colored with a priority from a finite set of natural numbers. The goal of Player Min is to choose her actions so as to minimize the probability that the highest infinitely-occurring priority is an odd number, while the goal of Player Max is the opposite. It is known that stochastic parity games are positionally determined [8, 7] and, when the transition structure is known, the optimal strategies can be computed in NP $\cap$ co-NP. This paper investigates the use of model-free reinforcement learning in approximating the value of the stochastic parity game when the transition structure is not known.

Littman [25] proposed the *Markov Games* framework to study the optimal strategies of players in stochastic games with unknown but fixed probability distributions. For payoffs of the players defined using stochastic reachability payoffs (under the stopping game assumption) or, analogously, stochastic discounted payoffs, Littman generalized the classical Q-learning [36, 4] algorithm to compute the optimal value of a game. This algorithm, known as the minimax-Q algorithm, was shown to converge [26] to the optimal value of the game. To enable the application of off-the-shelf convergent RL algorithms, such as the minimax-Q algorithm, for stochastic parity objectives, one needs to reduce the stochastic parity objective to a stochastic reachability objective. Moreover, to enable model-free RL, such a reduction cannot use any information about the transition structure (such as end-component decomposition) of the underlying stochastic game arena. This paper provides such model-free reduction from stochastic parity games to stochastic reachability games.

There are translations of stochastic parity games to stochastic games with scalar payoffs that – unlike model-free RL – require that the model is known [8, 6, 2]. Other approaches are applicable to non-stochastic parity games [5], or to qualitative games [7], whose goal is to compute strategies that guarantee almost-sure satisfaction of the parity condition.

The problem of translating ω-regular objectives into scalar rewards was recently solved in the case of one strategic agent (also known as the case of $1\frac{1}{2}$ players) when the environment is modeled as a Markov Decision Process (MDP) [17]. The MDP is equipped with a Büchi acceptance condition, and the resulting $1\frac{1}{2}$-player Büchi game is reduced to a reachability

game by adding a sink state, which is reachable via all accepting transitions of the Büchi game with probability ε. As ε tends to 0, the probability of reaching the sink approaches the probability of satisfaction of the objective.

The translation of ω-regular objectives to scalar rewards for $1\frac{1}{2}$-player games makes use of Büchi automata with restricted nondeterminism (limit-deterministic automata [35, 10, 16, 31] and, more generally, good-for-MDPs (GFM) automata [18]).

However, for $2\frac{1}{2}$ players, even the restricted nondeterminism that is acceptable for $1\frac{1}{2}$ players may lead to incorrect results, because a strategic player may force the objective automaton to reject a valid computation. Therefore, to solve the case of two strategic agents, one can resort to deterministic parity (or similarly powerful [29, 21]) automata. Since all ω-regular objectives are expressible as deterministic parity conditions, we present in Section 3 a reduction of stochastic *parity* games [7, 8] to stochastic reachability games [30, 9]. The latter can be solved by known model-free RL algorithms [25, 26].

For $1\frac{1}{2}$-player games, a more efficient reduction from stochastic parity games to stochastic reachability games is possible, and we discuss such a reduction in Section 4. Such a direct translation relieves the RL agent from the task of controlling the nondeterministic choices made by the Büchi automaton that encodes the objective.

2 Preliminaries

A *probability distribution* over a finite set S is a function $d\colon S\to[0,1]$ such that $\sum_{s\in S} d(s) = 1$. Let $\mathcal{D}(S)$ denote the set of all discrete distributions over S. We say a distribution $d \in \mathcal{D}(S)$ is a *point distribution* if $d(s)=1$ for some $s \in S$. For $d \in \mathcal{D}(S)$ we write $supp(d)$ for $\{s \in S\colon d(s) > 0\}$.

2.1 Stochastic Parity Games and Simple Stochastic Games

A *stochastic game arena* (SGA) $\mathcal{G}$ is a tuple $(S, A, T, S_{\text{Min}}, S_{\text{Max}})$, where S is a finite set of states, A is a finite set of *actions*, $T\colon S \times A \nrightarrow \mathcal{D}(S)$ is the *probabilistic transition (partial) function*, and $\{S_{\text{Min}}, S_{\text{Max}}\}$ is a partition of the set of states S.

For $s \in S$, $A(s)$ denotes the set of actions that can be selected in state s. For states $s, s' \in S$ and $a \in A(s)$ we write $p(s'|s, a)$ for $T(s, a)(s')$. A *run* of $\mathcal{G}$ is an ω-word $\langle s_0, a_1, s_1, \ldots\rangle \in S \times (A \times S)^\omega$ such that $p(s_{i+1}|s_i, a_{i+1}) > 0$ for all $i \geq 0$. A finite run is a finite such sequence, that is, a word in $S \times (A \times S)^*$. For an infinite run r, we write $\mathsf{inf}(r)$ for the set of state-action pairs that appear infinitely often in r. We write $Runs^{\mathcal{G}}(FRuns^{\mathcal{G}})$ for the set of runs (finite runs) of the SGA $\mathcal{G}$ and $Runs^{\mathcal{G}}(s)(FRuns^{\mathcal{G}}(s))$ for the set of runs (finite runs) of the SGA $\mathcal{G}$ starting from state s. We write $last(r)$ for the last state of a finite run r.

A game on an SGA $\mathcal{G}$ starts with a token in an *initial state* $s \in S$; players Min and Max construct an infinite run by taking turns to choose enabled actions, and then moving the token to a successor state sampled from the selected distribution. A strategy of player Min in $\mathcal{G}$ is a partial function $\mu\colon FRuns \nrightarrow \mathcal{D}(A)$, defined for $r \in FRuns$ if and only if $last(r) \in S_{\text{Min}}$, such that $supp(\sigma(r)) \subseteq A(last(r))$. A strategy ν of player Max is defined analogously.

A strategy σ is *pure* if $\sigma(r)$ is a point distribution wherever it is defined; otherwise, σ is *mixed*. We say that σ is *stationary* if $last(r) = last(r')$ implies $\sigma(r) = \sigma(r')$ wherever σ is defined. A strategy is *positional* if it is both pure and stationary. Let Σ_{Min} and Σ_{Max} be the sets of all strategies of player Min and player Max, respectively. Similarly, Π_{Min} and Π_{Max} denote the sets of all *positional* strategies of player Min and player Max, respectively.

Let $Runs^{\mathcal{G}}_{\mu,\nu}(s)$ denote the subset of runs $Runs^{\mathcal{G}}(s)$ starting from state s that are consistent with player Min and player Max following strategies μ and ν, respectively. The behavior of an SGA $\mathcal{G}$ under a strategy pair $(\mu, \nu) \in \Sigma_{\text{Min}} \times \Sigma_{\text{Max}}$ is defined on a probability space

$(\mathit{Runs}^{\mathcal{G}}_{\mu,\nu}(s), \mathcal{F}_{\mathit{Runs}^{\mathcal{G}}_{\mu,\nu}(s)}, \Pr^{\mathcal{G}}_{\mu,\nu}(s))$ over the set of infinite runs $\mathit{Runs}^{\mathcal{G}}_{\mu,\nu}(s)$. Given a random variable $f\colon \mathit{Runs}^{\mathcal{G}} \to \mathbb{R}$ over the infinite runs of $\mathcal{G}$, we denote by $\mathbb{E}^{\mathcal{G}}_{\mu,\nu}(s)\{f\}$ the expectation of f over the runs in the probability space $(\mathit{Runs}^{\mathcal{G}}_{\mu,\nu}(s), \mathcal{F}_{\mathit{Runs}^{\mathcal{G}}_{\mu,\nu}(s)}, \Pr^{\mathcal{G}}_{\mu,\nu}(s))$.

Let $[k]$ denote the set of natural numbers $\{0, 1, \ldots, k-1\}$. We consider the following payoffs of player Min to player Max.

- **Stochastic Parity Payoff.** A stochastic parity payoff in an SGA $\mathcal{G}$ is defined by a priority function $\mathsf{pri} : S \times A \to [k]$ that assigns to each state-action pair a natural number called the priority (or color) of that pair. The stochastic parity payoff $\mathcal{P}^{\mathcal{G}}_{\mu,\nu}(s)$ for a strategy pair $(\mu,\nu) \in \Sigma_{\mathrm{Min}} \times \Sigma_{\mathrm{Max}}$ from an initial state $s \in S$ is the probability that the highest recurring priority is odd, i.e.,

$$\mathcal{P}^{\mathcal{G}}_{\mu,\nu}(s) = \Pr{}^{\mathcal{G}}_{\mu,\nu}(s)\left\{r \in \mathit{Runs}^{\mathcal{G}}_{\mu,\nu}(s) \colon \max\{\mathsf{pri}(s,a) \,:\, (s,a) \in \mathsf{inf}(r)\} \text{ is odd}\right\} .$$

  A *stochastic Büchi payoff* is a stochastic parity payoff with $k = 2$. It is customarily specified as a set of *accepting* transitions: those with priority 1.
- **Stochastic Reachability Payoff.** A stochastic reachability payoff in an SGA $\mathcal{G}$ is defined by two distinguished sink states: the *accepting sink* state $s_a \in S$ and the *rejecting sink* state $s_r \in S$. Recall that a sink state s satisfies $p(s|s,a) = 1$ for all $a \in A$. The stochastic reachability payoff $\mathcal{R}^{\mathcal{G}}_{\mu,\nu}(s)$ for a strategy pair $(\mu,\nu) \in \Sigma_{\mathrm{Min}} \times \Sigma_{\mathrm{Max}}$ from an initial state $s \in S$ is the probability of reaching the accepting sink s_a, i.e.,

$$\mathcal{R}^{\mathcal{G}}_{\mu,\nu}(s) = \Pr{}^{\mathcal{G}}_{\mu,\nu}(s)\left\{r \in \mathit{Runs}^{\mathcal{G}}_{\mu,\nu}(s) : r \text{ visits } s_a\right\} .$$

We refer to a stochastic game arena with a parity payoff as a stochastic parity game (SPG) $\mathbb{G} = (\mathcal{G}, \mathsf{pri})$, and to a stochastic game arena with a reachability payoff as a stochastic reachability game (SRG) $\mathbb{G}' = (\mathcal{G}, s_a, s_r)$.

We assume that an SRG is a *stopping game*, i.e., for every pair of strategies $(\mu,\nu) \in \Sigma_{\mathrm{Min}} \times \Sigma_{\mathrm{Max}}$ and every initial state $s \in S$, the set $\{s_a, s_r\}$ is visited with probability 1. This implies that there are no sinks besides s_a and s_r and, in addition, that at least one sink is reachable with positive probability from every state in S.

Given a payoff function $\mathcal{C} \in \{\mathcal{P}, \mathcal{R}\}$, the objective of player Max in the corresponding game $\mathbb{G}$ is to maximize the payoff, while the objective of player Min is the opposite. For every state $s \in S$, we define its *upper value* $\overline{\mathsf{Val}}(\mathbb{G}, s)$ as the minimum payoff player Min can ensure irrespective of player Max's strategy. Similarly, the *lower value* $\underline{\mathsf{Val}}(\mathbb{G}, s)$ of a state $s \in S$ is the maximum payoff player Max can ensure irrespective of player Min's strategy, i.e.,

$$\overline{\mathsf{Val}}(\mathbb{G}, s) = \inf_{\mu \in \Sigma_{\mathrm{Min}}} \sup_{\nu \in \Sigma_{\mathrm{Max}}} \mathcal{C}^{\mathcal{G}}_{\mu,\nu}(s) \qquad \text{and} \qquad \underline{\mathsf{Val}}(\mathbb{G}, s) = \sup_{\nu \in \Sigma_{\mathrm{Max}}} \inf_{\mu \in \Sigma_{\mathrm{Min}}} \mathcal{C}^{\mathcal{G}}_{\mu,\nu}(s) .$$

The inequality $\underline{\mathsf{Val}}(\mathbb{G}, s) \leq \overline{\mathsf{Val}}(\mathbb{G}, s)$ holds of all two-player zero-sum games. A game is *determined* when, for every state $s \in S$, its lower value and its upper value are equal; we then say that the value of the game Val exists and $\mathsf{Val}(\mathbb{G}, s) = \underline{\mathsf{Val}}(\mathbb{G}, s) = \overline{\mathsf{Val}}(\mathbb{G}, s)$ for every $s \in S$. For a strategy $\mu \in \Sigma_{\mathrm{Min}}$ of player Min and similarly, for a strategy $\nu \in \Sigma_{\mathrm{Max}}$ of player Max, we define their values Val^{μ} and Val^{ν} as

$$\mathsf{Val}^{\mu}\colon s \mapsto \sup_{\nu \in \Sigma_{\mathrm{Max}}} \mathcal{C}^{\mathcal{G}}_{\mu,\nu}(s) \qquad \text{and} \qquad \mathsf{Val}^{\nu}\colon s \mapsto \inf_{\mu \in \Sigma_{\mathrm{Min}}} \mathcal{C}^{\mathcal{G}}_{\mu,\nu}(s) .$$

We say that a positional strategy $\mu_* \in \Pi_{\mathrm{Min}}$ of player Min is optimal if $\mathsf{Val}^{\mu_*} = \mathsf{Val}$. Similarly, a positional strategy $\nu_* \in \Pi_{\mathrm{Max}}$ of player Max is optimal if $\mathsf{Val}^{\nu_*} = \mathsf{Val}$. We say that a game is *positionally determined* if both players have positional optimal strategies.

▶ **Theorem 1** ([8, 7, 9]). *Stochastic parity games and stochastic reachability games are positionally determined and are in NP ∩ co-NP.*

2.2 Markov Decision Processes and Markov Chains

If, for every state $s \in S_{\text{Min}}$, or for every state $s \in S_{\text{Max}}$, $A(s)$ is a singleton, then $\mathcal{G}$ is a *Markov Decision Process* (MDP). If we want to emphasize that only Player Max (Min) has choices, we refer to Max-MDPs (Min-MDPs). If, for every state $s \in S$, $A(s)$ is a singleton, then $\mathcal{G}$ is a *Markov chain*. We denote by $\mathcal{G}_\mu$ ($\mathcal{G}_\nu$) the MDP obtained by fixing the strategy of player Min (Max) to positional strategy $\mu \in \Pi_{\text{Min}}$ ($\nu \in \Pi_{\text{Max}}$). If the strategies of both players are fixed, we denote the resulting Markov chain by $\mathcal{G}_{\mu,\nu}$.

Since an MDP has one strategic player, we can define an MDP by the tuple (S, A, T). For an MDP $\mathcal{M} = (S, A, T)$, we define its directed underlying graph $GR(\mathcal{M}) = (V, E)$ where $V = S$ and $E = \{(s, s') : T(s, a)(s') > 0 \text{ for some } a \in A(s)\}$. A sub-MDP of $\mathcal{M}$ is an MDP $\mathcal{M}' = (S', A', T')$, where $S' \subset S$, $A' \subseteq A$, is such that $A'(s) \subseteq A(s)$ for every $s \in S'$, and T' is T restricted to S' and A'. $\mathcal{M}'$ is closed under probabilistic transitions, i.e., for all $s \in S'$ and $a \in A'$, we have that $T(s, a)(s') > 0$ implies that $s' \in S'$. An *end-component* [11] of an MDP is a sub-MDP such that its underlying graph is strongly connected. Once an end-component C of an MDP is entered, there is a strategy that visits every state-action combination in C with probability 1 and stays in C forever. Moreover, for every strategy the union of the end-components is visited with probability 1.

A *bottom strongly connected component* (BSCC) of a Markov chain is a recurrent class. A BSCC is *even* if the highest priority of its transitions is even; otherwise the BSCC is *odd*.

3 From Stochastic Parity Games to Stochastic Reachability Games

In this section, we now show how to construct, for a Stochastic Parity Game $\mathbb{G}$, a family of Stochastic Reachability Games (SRGs) $\mathbb{G}^\varepsilon$, parametrized by a parameter $\varepsilon \in (0, 1)$, that have strong convergence properties to $\mathbb{G}$: for sufficiently small ε, optimal positional strategies (of either player) for $\mathbb{G}^\varepsilon$ are also optimal positional strategies for $\mathbb{G}$, and the limit value (when ε goes to 0) of $\mathbb{G}^\varepsilon$ goes to the value of $\mathbb{G}$ for every state. Thus, learning optimal strategies for this family of SRGs can be used to obtain optimal strategies for $\mathbb{G}$.

3.1 Limit Reachability Theorem

Given a stochastic parity game $\mathbb{G} = (\mathcal{G} = (S, A, T, S_{\text{Min}}, S_{\text{Max}}), \mathsf{pri})$, with $\mathsf{pri} : S \times A \to [k]$, and $\varepsilon \in (0, 1)$, we define a related stochastic reachability game $\mathbb{G}^\varepsilon = (\mathcal{G}^\varepsilon = (S \cup \{s_a, s_r\}, A, T^\varepsilon, S_{\text{Min}} \cup \{s_r\}, S_{\text{Max}} \cup \{s_a\}), s_a, s_r)$ such that:

$$T^\varepsilon(s,a)(s') = \begin{cases} 1 & \text{if } s = s' = s_a \text{ or } s = s' = s_r \\ \varepsilon^{k-i} & \text{if } s \in S,\ s' = s_a,\ i = \mathsf{pri}(s,a), \text{ and } i \text{ is odd} \\ \varepsilon^{k-i} & \text{if } s \in S,\ s' = s_r,\ i = \mathsf{pri}(s,a), \text{ and } i \text{ is even} \\ (1-\varepsilon^{k-i}) \cdot T(s,a)(s') & \text{if } s, s' \in S \text{ and } i = \mathsf{pri}(s,a) \\ 0 & \text{otherwise.} \end{cases}$$

An example is shown in Figure 1, where boxes denote states in S_{Max}, large circles denote states in S_{Min} and small circles denote probabilistic branches. Intuitively, for small enough ε, the chance of prematurely moving to the wrong sink is negligible. Specifically, the probability of reaching a sink from a transient transition is negligible, while in a recurrent set of states, the probability of reaching a sink from a lower priority transition is negligible compared to the probability of doing so from a higher priority transition. The definition of T^ε also applies to $1\frac{1}{2}$-player games and to Markov chains; it is illustrated in Figure 2.

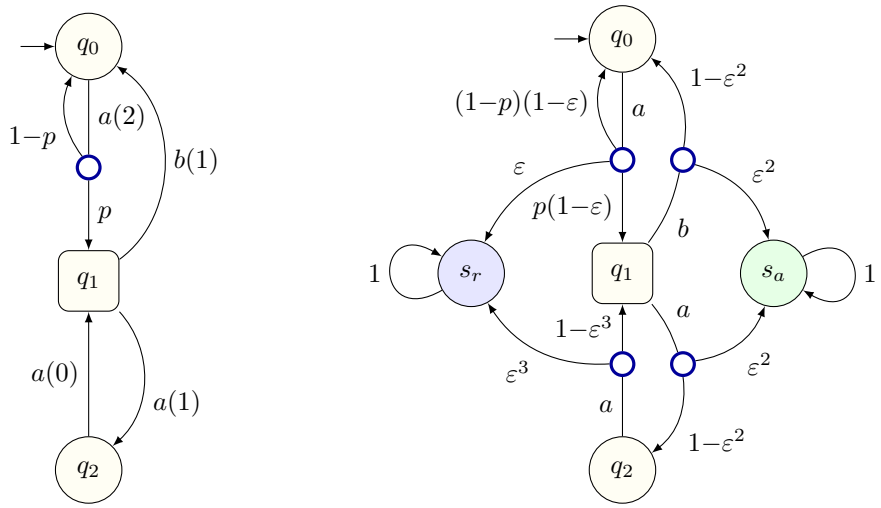


Figure 1 An SPG $\mathbb{G}$ (left) and the corresponding SRG $\mathbb{G}^\varepsilon$ (right).

▶ **Proposition 2.** *For every SPG $\mathbb{G}$, the SRG $\mathbb{G}^\varepsilon$ is a stopping game.*

In the next section, we prove the following lemma.

▶ **Lemma 3.** *For every positional strategy pair $(\mu,\nu) \in \Pi_{\text{Min}} \times \Pi_{\text{Max}}$ and every state $s \in S$ we have that $\mathcal{P}^{\mathcal{G}}_{\mu,\nu}(s) = \lim_{\varepsilon\downarrow 0} \mathcal{R}^{\mathcal{G}^\varepsilon}_{\mu,\nu}(s)$. Moreover, for sufficiently small ε, optimal strategies for $\mathbb{G}^\varepsilon$ are also optimal for $\mathbb{G}$.*

▶ **Theorem 4.** *For every stochastic parity game $\mathbb{G} = ((S, A, T, S_{\text{Min}}, S_{\text{Max}}), \mathsf{pri})$ and the set of stochastic reachability games $\mathbb{G}^\varepsilon$, we have that $\mathsf{Val}(\mathbb{G}, s) = \lim_{\varepsilon\downarrow 0} \mathsf{Val}(\mathbb{G}^\varepsilon, s)$, for all $s \in S$.*

Proof. Notice that, for every positional strategy $\nu \in \Pi_{\text{Max}}$, we have that:

$$\begin{aligned} \inf_{\mu\in\Sigma_{\text{Min}}} \mathcal{P}^{\mathcal{G}}_{\mu,\nu}(s) = \min_{\mu\in\Pi_{\text{Min}}} \mathcal{P}^{\mathcal{G}}_{\mu,\nu}(s) &= \min_{\mu\in\Pi_{\text{Min}}} \lim_{\varepsilon\downarrow 0} \mathcal{R}^{\mathcal{G}^\varepsilon}_{\mu,\nu}(s) \\ &= \lim_{\varepsilon\downarrow 0} \min_{\mu\in\Pi_{\text{Min}}} \mathcal{R}^{\mathcal{G}^\varepsilon}_{\mu,\nu}(s) = \lim_{\varepsilon\downarrow 0} \inf_{\mu\in\Sigma_{\text{Min}}} \mathcal{R}^{\mathcal{G}^\varepsilon}_{\mu,\nu}(s). \end{aligned} \tag{1}$$

The first and the last equalities follow due to the optimality of positional strategies in stochastic parity games (Theorem 1), while the second and the third equalities follows from Lemma 3. Now, observe that for all $s \in S$ we have that:

$$\begin{aligned} \mathsf{Val}_{\mathcal{P}}(\mathbb{G}, s) &= \sup_{\nu\in\Sigma_{\text{Max}}} \inf_{\mu\in\Sigma_{\text{Min}}} \mathcal{P}^{\mathcal{G}}_{\mu,\nu}(s) && \text{(by definition)} \\ &= \max_{\nu\in\Pi_{\text{Max}}} \inf_{\mu\in\Sigma_{\text{Min}}} \mathcal{P}^{\mathcal{G}}_{\mu,\nu}(s) && \text{(from Theorem 1)} \\ &= \max_{\nu\in\Pi_{\text{Max}}} \lim_{\varepsilon\downarrow 0} \inf_{\mu\in\Sigma_{\text{Min}}} \mathcal{R}^{\mathcal{G}^\varepsilon}_{\mu,\nu}(s) && \text{(from (1))} \\ &= \lim_{\varepsilon\downarrow 0} \max_{\nu\in\Pi_{\text{Max}}} \inf_{\mu\in\Sigma_{\text{Min}}} \mathcal{R}^{\mathcal{G}^\varepsilon}_{\mu,\nu}(s) && \text{(from Lemma 3)} \\ &= \lim_{\varepsilon\downarrow 0} \sup_{\nu\in\Sigma_{\text{Max}}} \inf_{\mu\in\Sigma_{\text{Min}}} \mathcal{R}^{\mathcal{G}^\varepsilon}_{\mu,\nu}(s) && \text{(from Theorem 1)} \\ &= \lim_{\varepsilon\downarrow 0} \mathsf{Val}_{\mathcal{R}}(\mathbb{G}^\varepsilon, s) && \text{(by definition).} \end{aligned}$$

◀

3.2 Absorption Probabilities

For a pair of positional strategies (μ, ν), a stochastic game arena $\mathcal{G}$ ($\mathcal{G}^\varepsilon$) reduces to a Markov chain $\mathcal{G}_{\mu,\nu}$ ($\mathcal{G}^\varepsilon_{\mu,\nu}$), whose states are partitioned into a set of transient states and one or more recurrent (communicating) classes, where a *communicating class* is a class that becomes recurrent when removing the sinks s_a and s_r. Comparing the Markov chains $\mathcal{G}_{\mu,\nu}$ and $\mathcal{G}^\varepsilon_{\mu,\nu}$, one observes that:

- Every transient state of the Markov chain $\mathcal{G}_{\mu,\nu}$ remains transient in $\mathcal{G}^\varepsilon_{\mu,\nu}$.
- All recurrent classes of $\mathcal{G}_{\mu,\nu}$ become communicating classes of $\mathcal{G}^\varepsilon_{\mu,\nu}$.
- The chain $\mathcal{G}^\varepsilon_{\mu,\nu}$ is absorbing; the runs that do not eventually reach either s_a or s_r form a set of measure 0.
- Since a positional strategy selects one action for each state, exactly one priority in $[k]$, denoted by $\mathsf{pri}(s)$ is associated to each state of $\mathcal{G}_{\mu,\nu}$.

Note that the runs of $\mathcal{G}_{\mu,\nu}$ that do not reach some recurrent class are a set of measure 0. Moreover, the runs that reach the absorbing states of $\mathbb{G}^\varepsilon$ without going through a recurrent class of $\mathbb{G}$ are a set, whose measure converges to 0 when ϵ goes to 0. Hence, we can analyze the Markov chains induced by positional strategies (μ, ν) one recurrent class at a time.

▶ **Lemma 5.** *Suppose the Markov chain $\mathcal{M}$ is recurrent.*

1. *The sum of the absorption probabilities of the two sinks of $\mathcal{M}^\varepsilon$ is always 1.*
2. *The limit, for ε that goes to 0, of the absorption probabilities of the odd sink of $\mathcal{M}^\varepsilon$ is 1 if, and only if, the highest priority of the states in $\mathcal{M}$ is odd.*
3. *The limit, for ε that goes to 0, of the absorption probabilities of the even sink of $\mathcal{M}^\varepsilon$ is 1 if, and only if, the highest priority of the states in $\mathcal{M}$ is even.*

Proof. The first claim follows from $\mathcal{M}^\varepsilon$ being a stopping game (cf. Proposition 2). For the second claim, let M be the $n \times n$ transition matrix of $\mathcal{M}$. Let $\mathsf{pri}(i)$ be the priority of state i. The Markov chain $\mathcal{M}^\varepsilon$ is absorbing and its transition matrix M^ε can be written in the following form:

$$M^\varepsilon = \begin{pmatrix} I_2 & 0 \\ R & Q \end{pmatrix} \ ,$$

where I_u is the $u \times u$ identity matrix, R is $n \times 2$, and Q is $n \times n$. The first two rows and columns of M^ε are named o and e (odd and even, respectively). The other rows and columns are numbered from 0 to $n-1$. Let E be the $n \times n$ diagonal matrix such that

$$e_{ii} = \varepsilon^{k-\mathsf{pri}(i)} \ .$$

The matrix R is defined by $r_{io} = e_{ii}$ if $\mathsf{pri}(i)$ is odd and 0 otherwise; likewise $r_{ie} = e_{ii}$ if $\mathsf{pri}(i)$ is even and 0 otherwise. The matrix Q is defined by

$$Q = (I_n - E) \cdot M \ .$$

The probabilities of reaching the sinks from the remaining states are computed as

$$P = (I_n - Q)^{-1} \cdot R \ ,$$

where $N = (I_n - Q)^{-1}$ is called the *fundamental matrix* for the absorbing chain M^ε and n_{ij} is the expected number of times the absorbing chain is in state j if it starts in state i [23, Theorem 3.2.1] [14, Theorem 11.4]. Since M is recurrent, $n_{ij} > 0$. Let $N_i^+(\varepsilon) = \max_j\{n_{ij}\}$ and $N_i^-(\varepsilon) = \min_j\{n_{ij}\}$. That gives lower and upper bounds for the rows of N that are strictly positive row vectors with uniform entries. Then,

$$\lim_{\varepsilon \downarrow 0} \frac{N_i^-(\varepsilon)}{N_i^+(\varepsilon)} \cdot \frac{\sum_{0 \le \ell < n} r_{\ell o}}{\sum_{0 \le \ell < n} r_{\ell e}} \le \lim_{\varepsilon \downarrow 0} \frac{p_{io}}{p_{ie}} \le \lim_{\varepsilon \downarrow 0} \frac{N_i^+(\varepsilon)}{N_i^-(\varepsilon)} \cdot \frac{\sum_{0 \le \ell < n} r_{\ell o}}{\sum_{0 \le \ell < n} r_{\ell e}} \ .$$

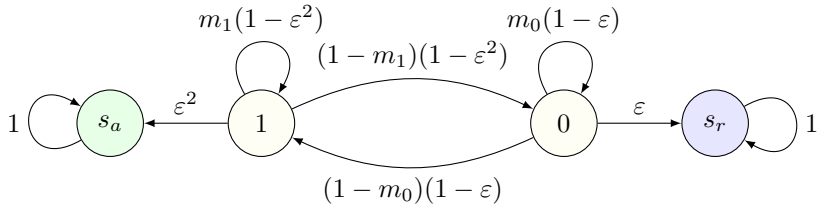


Figure 2 An augmented Markov chain.

Since $\frac{N_i^+(\varepsilon)}{N_i^-(\varepsilon)}$ converges to the ratio of the largest stationary probability in $\mathcal{M}$ to the smallest such probability, regardless of i, its limit is positive and finite. The limit of p_{io}/p_{ie} is therefore determined by the ratio of the sums of the columns of R. Both columns are polynomials in ε with no constant term, and the lowest degree of the two polynomials is different – one is even and the other is odd. Therefore, *either* p_{io}/p_{ie} goes to infinity (or the denominator stays 0), *or* p_{io}/p_{ie} goes to 0. Since $p_{io} + p_{ie} = 1$, this entails that one probability goes to 1 (the one for the column of R with the lowest degree term), while the other goes to 0.

The proof for the third claim is similar. ◀

▶ **Example 6.** Figure 2 shows an augmented Markov chain whose original Markov chain has two states, State 0 with priority 2 and State 1 with priority 1. The probabilities m_0 and m_1 are from $[0, 1]$. The transition matrix, M^ε, of the augmented chain is given by:

$$M^\varepsilon = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \varepsilon & m_0\,(1-\varepsilon) & (1-m_0)\,(1-\varepsilon) \\ \varepsilon^2 & 0 & (1-m_1)\left(1-\varepsilon^2\right) & m_1\left(1-\varepsilon^2\right) \end{pmatrix} .$$

The fundamental matrix N is:

$$N = \frac{1}{|I-Q|} \cdot \begin{pmatrix} 1-m_1+m_1\varepsilon^2 & (1-m_0)(1-\varepsilon) \\ (1-m_1)(1-\varepsilon^2) & 1-m_0+m_0\varepsilon \end{pmatrix} ,$$

with $|I-Q| = \varepsilon\big(1-m_1+\varepsilon(1-m_0)-\varepsilon^2(1-m_0-m_1)\big)$. The probabilities of eventually reaching the sinks are given by:

$$N \cdot R = \frac{\varepsilon}{|I-Q|} \begin{pmatrix} \varepsilon(1-\varepsilon)(1-m_0) & 1-m_1+m_1\varepsilon^2 \\ \varepsilon(1-m_0+m_0\varepsilon) & (1-\varepsilon^2)(1-m_1) \end{pmatrix} .$$

Both rows of $N \cdot R$ sum to 1 and

$$\lim_{\varepsilon \downarrow 0} N \cdot R = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} ,$$

as expected.

Proof of Lemma 3. For a given pair of positional strategies $(\mu, \nu) \in \Pi_{\text{Min}} \times \Pi_{\text{Max}}$ on $\mathcal{G}$, and for every state s of the Markov chain $\mathcal{G}_{(\mu,\nu)}$, Lemma 5 shows that the following holds for every state $s \in S$:

1. if the state s is in an even BSCC of $\mathcal{G}_{(\mu,\nu)}$ then $\lim_{\varepsilon\downarrow 0} \mathcal{R}^{\mathcal{G}^\varepsilon}_{\mu,\nu}(s) = 0 = \mathcal{P}^{\mathcal{G}}_{\mu,\nu}(s)$,
2. if the state s is in an odd BSCC of $\mathcal{G}_{(\mu,\nu)}$ then $\lim_{\varepsilon\downarrow 0} \mathcal{R}^{\mathcal{G}^\varepsilon}_{\mu,\nu}(s) = 1 = \mathcal{P}^{\mathcal{G}}_{\mu,\nu}(s)$,

Let $s \in S$ be a transient state of $\mathcal{G}_{(\mu,\nu)}$. Let $f_s^{\mu,\nu}$ be the expected number of transitions taken before reaching a BSCC when starting at s in $\mathcal{G}_{(\mu,\nu)}$. Note that, with argument similar to the one used in [17, Lemma 2], one shows that

$$\mathcal{R}^{\mathcal{G}^\varepsilon}_{\mu,\nu}(s) - \varepsilon f_s^{\mu,\nu} \leq \mathcal{P}^{\mathcal{G}}_{\mu,\nu}(s) \leq \mathcal{R}^{\mathcal{G}^\varepsilon}_{\mu,\nu}(s) + \varepsilon f_s^{\mu,\nu} \ .$$

That is, the effect of transient states vanishes with ε. Therefore, as ε goes to 0, $\mathcal{R}^{\mathcal{G}^\varepsilon}_{\mu,\nu}(s)$ tends to $\mathcal{P}^{\mathcal{G}}_{\mu,\nu}(s)$.

Note that this means that, for either player, for every strategy μ or ν that is superior over a strategy μ' or ν', respectively, in $\mathcal{P}^{\mathcal{G}}$, there is an $\varepsilon' > 0$ such that, for $\varepsilon \in (0, \varepsilon')$, μ or ν is superior over μ' or ν', respectively, in $\mathcal{R}^{\mathcal{G}^\varepsilon}$.

Given that optimal strategies are positional, and that there are only finitely many positional strategies, this implies that there is an $\varepsilon' > 0$ such that, for $\varepsilon \in (0, \varepsilon')$, optimal strategies for either player in $\mathcal{R}^{\mathcal{G}^\varepsilon}$ are also optimal in $\mathcal{P}^{\mathcal{G}}$. ◀

4 Markov Decision Processes with Parity Objectives

The reduction of Section 3 works for all stochastic parity games, but, for a parity condition with k priorities, it employs up to k distinct powers of ε. In practice, this may lead to slow convergence as a reinforcement learner will require long episodes. We introduce another reduction, which is only valid for $1\frac{1}{2}$-player games (Markov decision processes), but only uses the first power of ε. We consider Max-MDPs. (Min-MDPs can be treated by dualizing the MDP first.)

Our reduction is obtained by composing the MDP with the following *priority tracker* gadget.

▶ **Definition 7** (Priority Tracker). *Given a set of priorities $[k]$ and a parameter $0 \leq \varepsilon \leq 1$, the priority tracker $\mathcal{T}^\varepsilon$ is an MDP (S, A, T) where:*

- *$S = \{s_I, s_a\} \cup \{s_{2c} : 2c \in [k]\}$ is the set of states with $2 + \lceil k/2 \rceil$ states including a distinguished initial state s_I, an accepting sink state s_a, and a state s_{2c} for each pair $(2c, 2c+1)$ of priorities (w.l.o.g., we assume that k is even);*
- *$A = [k]$ is the set of actions that are labeled by priorities from the set $[k]$; and*
- *$T : S \times A \to \mathcal{D}(S)$ is the transition function defined in the following way.*

$$T(s,a)(s') = \begin{cases} 1-\varepsilon & \text{if } s = s' = s_I \\ \varepsilon & \text{if } s = s_I \text{ and } s' = s_{2c} \text{ and } a \in \{2c, 2c+1\} \\ 1 & \text{if } s = s' = s_{2c} \text{ and } a < 2c \\ \varepsilon & \text{if } s = 2c,\ a = 2c+1,\ \text{and } s' = s_a \\ 1-\varepsilon & \text{if } s = 2c,\ a = 2c+1,\ \text{and } s' = 2c \\ 1 & \text{if } s = 2c,\ a > 2c+1,\ \text{and } s' = 2\lfloor a/2 \rfloor \\ 1 & \text{if } s = s' = s_a \\ 0 & \text{otherwise.} \end{cases}$$

An example of the priority tracker for priority set $\{0, 1, \ldots, 5\}$ is shown in Figure 3. Intuitively, for small enough ε, the gadget is, with high probability, still in state s_I when the MDP enters an end component. Moreover, for small enough ε, the gadget is very likely to see the dominant priority of the end component before it reaches s_a, in which case it reaches s_a with probability 1 if and only if the dominating priority of the end component is odd.

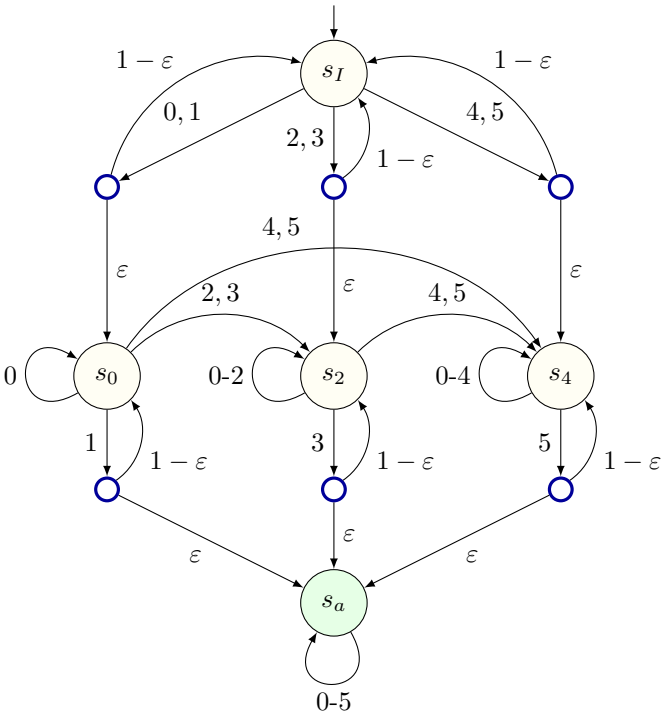


Figure 3 Priority tracker gadget for priorities 0-5.

To prove that the gadget of Figure 3 may be used for $1\frac{1}{2}$-player stochastic games (Max-MDPs), we make use of a Büchi automaton (i.e., with a parity condition with priorities 0 and 1), which is derived from the priority tracker gadget, as follows.

The *Parity to Büchi* (PtB) gadget for $2k$ priorities, $\mathcal{P}_k$, is a Büchi automaton over the alphabet $[2k]$ that accepts the language of the following LTL property:

$$\mathsf{G}\,\mathsf{F}\{2k-1\} \vee (\mathsf{F}\,\mathsf{G}\,[2k-2] \wedge (\mathsf{G}\,\mathsf{F}\{2k-3\} \vee \cdots)) \ . \tag{2}$$

The PtB $\mathcal{P}_6$ is shown in Figure 4. In general, the PtB is related to the priority tracker gadget for the same number of priorities by the following transformation. One replaces the transitions from the initial state by nondeterministic transitions (all non-accepting), and uses non-accepting transitions:

- from state $2c$, one self-loops with every priority $d < 2c+1$;
- from state $2c$, one moves to state $2 \cdot \lfloor d/2 \rfloor$ for every priority $d > 2c+1$;

and accepting transitions

- from state $2c$, one self-loops with every priority $d = 2c+1$.

▶ **Lemma 8.** *Let $\mathbb{B}$ be the synchronous composition of the Max-MDP $\mathbb{M} = ((S, A, T, \emptyset, S), \mathsf{pri})$ and $\mathcal{P}_{2k}$, with $\mathsf{pri} : S \times A \to [2k]$. Then, for every $s \in S$,*

$$\mathsf{Val}(\mathbb{M}, s) = \mathsf{Val}(\mathbb{B}, (s, s_I)) \ .$$

Proof. The automaton $\mathcal{P}_{2k}$ is good for MDPs [18], because it is a suitable limit-deterministic Büchi automaton [16, 31]. This means that it can be composed with any Max-MDP equipped with a parity condition to compute the probability of satisfaction of (2). ◀

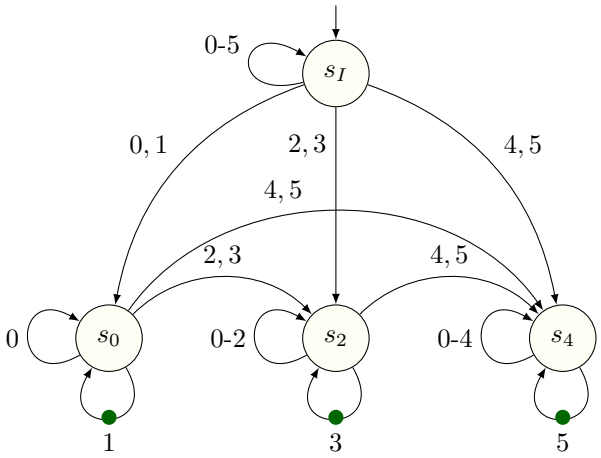


Figure 4 PtB gadget for priorities 0-5. The transitions marked with a dot are accepting.

Let $\mathbb{M} = (\mathcal{M}, \mathsf{pri})$ be a stochastic game arena, with no choices for Player Min and with parity objective, and $\mathcal{T}^\varepsilon$ be the priority tracker. We define $\mathbb{M}^\varepsilon_\mathcal{T}$ to be the synchronous composition of $\mathbb{M}$ with $\mathcal{T}^\varepsilon$, synchronized with the priorities of the transitions. We assume that $\mathbb{M}^\varepsilon_\mathcal{T}$ is equipped with a reachability objective with s_a as the accept state.

▶ **Theorem 9.** *For every Max-MDP* $\mathbb{M} = ((S, A, T, S_{\text{Min}}, S_{\text{Max}}), \mathsf{pri})$ *and its induced set of stochastic reachability games* $\mathbb{M}^\varepsilon_\mathcal{T}$*, we have that, for all* $s \in S$*,*

$$\mathsf{Val}(\mathbb{M}, s) = \lim_{\varepsilon \downarrow 0} \mathsf{Val}(\mathbb{M}^\varepsilon_\mathcal{T}, (s, s_I)) \ .$$

Proof. The proof is in two parts.

- **Bounding the limit from below.** In the first part of the proof we show that, for every $\delta > 0$, there exists an $\varepsilon_\delta > 0$ such that for every $\varepsilon < \varepsilon_\delta$ we have that

  $$\mathsf{Val}(\mathbb{M}^\varepsilon_\mathcal{T}, (s, s_I)) \geq \mathsf{Val}(\mathbb{M}, s) - \delta.$$

  Assume that player Max plays a positional strategy, which is optimal for $\mathbb{M}$. We first consider the special cases that s is in a winning or losing BSCC in the Markov chain induced by this strategy.
  - The state s is in an accepting BSCC with dominating odd priority o. In this case, the probability to satisfy the parity objective is 1.
    At the same time, in composition with the priority tracker gadget, no state s_e with $e > o$ can be reached in the priority tracker, and there is a path from every state to s_a in the finite product Markov chain. The reachability probability is therefore also 1. In this case $\mathsf{Val}(\mathbb{M}^\varepsilon_\mathcal{T}, (s, s_I)) = \mathsf{Val}(\mathbb{M}, s)$ holds.
  - The state s is in a rejecting BSCC. In this case, the probability to satisfy the parity objective is 0, and $\mathsf{Val}(\mathbb{M}^\varepsilon_\mathcal{T}, (s, s_I)) \geq \mathsf{Val}(\mathbb{M}, s)$ holds trivially.

  It is therefore enough to select ε_δ small enough that the chance of progressing away from the initial state s_I of the priority tracker before reaching a BSCC in the induced Markov chain happens with a probability below δ.
  Note that the chance of reaching any BSCC *with* the priority tracker still in state s_I is naturally no bigger than the chance of reaching the BSCC *with or without* the

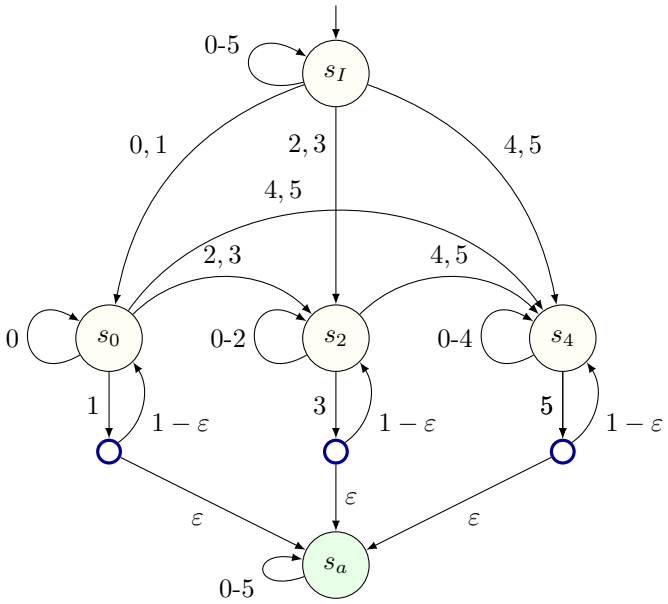


Figure 5 The reduction of the PtB gadget for priorities 0-5 to the reachability problem

priority tracker being in state s_I. Therefore, with the two special cases from above, $\mathsf{Val}(\mathbb{M}^{\varepsilon}_{\mathcal{T}}, (s, s_I)) \geq \mathsf{Val}(\mathbb{M}, s) - \delta$ follows. (ε_δ can, for example, be chosen as δ divided by the expected number of transitions taken before reaching a BSCC.)

Bounding the limit from above. In the second part of the proof we show

$$\mathsf{Val}(\mathbb{M}^{\varepsilon}_{\mathcal{T}}, (s, s_I)) \leq \mathsf{Val}(\mathbb{B}^{\varepsilon}, (s, s_I)) \leq \mathsf{Val}(\mathbb{B}, (s, s_I)) + \delta = \mathsf{Val}(\mathbb{M}, s) + \delta.$$

For the first inequality, note that $\mathbb{B}^{\varepsilon}$ is similar to $\mathbb{M}^{\varepsilon}_{\mathcal{T}}$, except in the nondeterministic vs. probabilistic transitions from state s_I. Therefore, the priority tracker gadget can be interpreted as what one gets when the player uses a particular positional randomized strategy to resolve the nondeterminism in $\mathbb{B}^{\varepsilon}$. As this is one possible strategy to resolve the nondeterminism, the inequality follows.
The equation follows from the fact that the PtB is good for MDPs [18], because it is a suitable limit-deterministic Büchi automaton [16, 31] that recognizes the words of priorities, where the highest priority that occurs infinitely often is odd.
To establish the middle inequality, consider the gadget $\mathbb{B}^{\varepsilon}$ that is obtained by composing the gadget in Figure 5 with the MDP. Hahn et al. [17, Lemma 2] showed for such SLDBA that, for every δ', there exists ε_0 such that, for all $\varepsilon < \varepsilon_0$, we have that

$$\mathsf{Val}(\mathbb{B}, (s, s_I)) - \delta' \leq \mathsf{Val}(\mathbb{B}^{\varepsilon}, (s, s_I)) \leq \mathsf{Val}(\mathbb{B}, (s, s_I)) + \delta' \tag{3}$$

holds; this provides the second inequality. ◀

While the previous theorem proves that the priority-tracker works in the case of Max-MDPs, unfortunately this reduction cannot be used for general stochastic games, or even Min-MDPs, where (just as the nondeterminism in the MDP and the PtB gadget are resolved by different players) Min can gain an unfair advantage from knowing the state of the priority tracker gadget, as demonstrated by the following lemma.

▶ **Lemma 10.** *The priority tracker construction is incorrect for stochastic parity games, even when restricted to Min-MDPs.*

Proof. The game of Figure 6 shows that the priority-tracker should not be used for Min-MDPs (or generally for $2\frac{1}{2}$-player games). The example game is a Min-MDP: Min is the only player who makes moves. Yet, Max wins because the highest recurring priority is either 3 or 1. If, however, Min chooses b from q_0 until the priority-tracker enters state s_2, and then switches to action a, the highest recurring priority is intuitively deemed to be even (as it is not 3, while the priority tracker is in the component that intuitively checks if it is 2 or 3) and Min is adjudicated the game, because Max cannot reach the reachability target s_a. ◀

5 Experimental Results

Our reduction from stochastic parity games to stochastic reachability games can be done on the fly because the augmentation of the game graph only requires knowledge of the current priority. This enables us to use model-free RL to solve the stochastic reachability game that results from our reduction. We use minimax Q-learning for alternating Markov Games [26] to compute strategies. By assigning an undiscounted reward of +1 for reaching s_a (and 0 reward otherwise), the values of state-action pairs computed by Q-learning are direct estimates of the probability of reaching s_a, and hence of the probability to satisfy the property.

The models we use to test our reduction [27] are in Table 1. This table presents the name of the game, the number of states of the game, the priorities in the game, the probability to satisfy the property by player Max when both players play optimally, and the probability as estimated by Q-learning. As a verifying step, we fixed Max player's strategy after learning and solved the resulting stochastic parity game with MUNGOJERRIE [17]. The resulting probability to satisfy the property is presented in the table. We also show the time (in seconds) that it took for Q-learning to compute the strategies, and the value of ϵ used in learning. Each episode was run until it terminated by a transition to a sink.

In `coprobActive` [1], a cop and a robber take turns, deterministically moving to adjacent vertices on a house-shaped graph. Cop is player Max and robber is player Min. The ω-regular property for our game is to eventually reach a state where the cop and robber are at the same vertex. Both players select their starting vertex in the first move of the game, starting with the cop's selection. In this game, the cop has a winning strategy. `coprobPassive` is identical except the robber has an additional action where she does not move. In this variant, the robber has a winning strategy. `coprobActiveP` and `coprobPassiveP` are identical to the prior two models except that moves are only successful with probability 1/2. The state will remain unchanged if the move is unsuccessful. The cop has a winning strategy in both of these models. The `randomME` model is a randomized mutual exclusion protocol [3, p. 836], modified to allow simultaneous requests by the two clients. Player Max controls the arbiter, while player Min controls both clients. The objective of the game is to guarantee absence of

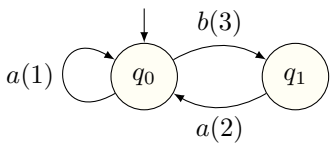


Figure 6 A parity game. Both states are controlled by player Min.

Table 1 Q-learning results for games. Estimated probabilities, verifying probabilities, and times are the average of three runs. We tuned hyperparameters individually for each experiment.

Name	states	priorities	prob.	estim. prob.	verify prob.	time (s)	ϵ
`coprobActive`	104	0,1	1	0.99	1	1.13	0.05
`coprobPassive`	105	0,1	0	0	0	0.46	0.05
`coprobActiveP`	105	0,1	1	0.99	1	2.97	0.03
`coprobPassiveP`	105	0,1	1	0.99	1	4.01	0.03
`coprobSafe`	148	0,1	1	0.99	1	3.50	0.03
`coprobSafeP`	150	0,1	13/15	0.85	0.86	13.57	0.03
`randomME`	30	1,2,3	1	0.95	1	4.43	0.04
`harding`	6	0,1,2	1	0.96	1	2.88	0.04
`smg1`	8	0,1	1	0.97	1	2.91	0.02
`difference`	99241	0,1	1	0.92	1	12.23	0.1
`ttt`	6321	0,1	1	1	1	1.57	0.07
`coins`	38200	0,1	1	0.97	1	2.92	0.05
`penney`	1745	0,1	1/3	0.33	0.33	0.28	0.1
`robots`	45784	0,1	1	1	0.94	1031.29	0.003

starvation for one client. The `harding` example [20] shows that the use of nondeterministic automata to express ω-regular objectives for games with two strategic players may lead to incorrect results. In `smg1` messages are exchanged between a server and a client [24]. In `difference` [39], the Max player chooses digits and the Min players assigns them to places in two two-digit numbers, x and y. The goal of Player Max is to guarantee $x - y \geq 40$. Example `ttt` is a model of the tic-tac-toe game. In `coins` [39], the two players remove in turn a coin from one end of a row of 4 coins. Player Max tries to collect coins worth at least as much as the coins collected by Player Min. In `penney` [28], each player chooses a sequence of three heads and tails. A fair coin is then tossed repeatedly, and the first player whose sequence turn up wins. In `robots` [19], two robots, each controlled by one player, navigate a grid world. Table 1 shows a strong correlation between the value of ε needed to reliably learn an optimal strategy and the learning time.

The reduction of Section 4 from parity objectives to reachability objectives for Markov decision processes can be done on the fly because it only requires knowledge of the current priority. As before, we can use model-free RL to solve the resulting reachability objective.

In Table 2 we compare 3 methods that use Q-learning to learn a strategy that maximizes the probability of satisfying a parity objective in a MDP. In Method 1, we translate the parity objective into a SLDBA objective and use the reduction from [17]. In Method 2, we treat our MDP as a stochastic game (with only one player) and utilize the reduction from Section 3. In Method 3, we use the reduction introduced in Section 4. We tuned the hyperparameters to minimize time subject to the following constraints. First, the strategies produced, as verified by the model checker, satisfied the property with the maximum probability. Second, since each method produces an estimate of the probability of the satisfaction of the property, the estimated probability of satisfaction was within 10% of the true value.

The examples `deferred` and `chocolates` have properties where the learner has many opportunties to transition the SLDBA to its final accepting region. The difficulty here is that the learner must learn to wait to make this transition, which happens with low probability during the initial phase of Q-learning where the learner explores randomly. Methods 2 and 3 perform better in these examples because there is no additional choice for the learner to

Table 2 Q-learning results for MDPs. The Q-table for each experiment was initialized to zero. The number of states with the SLDBA objective is listed first, followed by the parity objective.

Name	states	priorities	Meth. 1 time (s)	Meth. 2 time (s)	Meth. 3 time (s)
`deferred`	74,25	1,2	3.63	2.73	0.52
`trafficNtk`	392,773	0,1,2	0.45	1.87	1.88
`chocolates`	7168,1034	1,2	1523.89	13.21	8.14
`shoot1`	1175,595	0,1,2	0.36	> 2000	33.26
`agridGR2`	252,216	0-5	25.59	58.26	13.97

learn. In `shoot1` and `trafficNtk`, all methods are able to produce strategies that satisfy the property with the maximum probability relatively easily. However, Methods 2 and 3 require small values of ϵ in order for the estimated probabilites to be close to their true values, increasing the learning time. In `agridGR2`, the large number of priorities is harmful to Method 2's performance due to the increasing powers of ϵ. Throughout each of these experiments, Method 3 outperforms Method 2 and is competitive with Method 1.

6 Conclusion

We have presented a reduction from stochastic parity games to stochastic reachability games that allows one to apply model-free reinforcement learning to the computation of the game values and optimal strategies. We have also described a translation that, while only suitable for $1\frac{1}{2}$-player games – more precisely, for Max-MDPs – requires shorter training episodes than the more general reduction. Initial experiments show that the proposed approach allows an off-the-shelf reinforcement learning algorithm like minimax Q-learning to compute optimal strategies for games of moderate size.

References

1 M. Aigner and M. Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8:1–12, 1984.
2 D. Andersson and Miltersen P. B. The complexity of solving stochastic games on graphs. In *Algorithms and Computation*, pages 112–121, 2009.
3 C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.
4 V. S. Borkar and S. P. Meyn. The ODE method for convergence of stochastic approximation and reinforcement learning. *SIAM Journal on Control and Optimization*, 38(2):447–469, 2000.
5 K. Chatterjee and N. Fijalkow. A reduction from parity games to simple stochastic games. In *Games, Automata, Logics and Formal Verification, GandALF*, pages 74–86, June 2011.
6 K. Chatterjee and T. A. Henzinger. Reduction of stochastic parity to stochastic mean-payoff games. *Inf. Process. Lett.*, 106(1):1–7, 2008.
7 K. Chatterjee, M. Jurdziński, and T. A. Henzinger. Simple stochastic parity games. In *Computer Science Logic (CSL)*, pages 100–113, 2003.
8 K. Chatterjee, M. Jurdziński, and T. A. Henzinger. Quantitative stochastic parity games. In *Symposium on Discrete Algorithms, SODA*, pages 121–130, 2004.
9 A. Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992.
10 C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, July 1995.
11 L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1998.
12 J. Fu and U. Topcu. Probably approximately correct MDP learning and control with temporal logic constraints. In *Robotics: Science and Systems*, July 2014.
13 I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

14 C. M. Grinstead and J. L. Snell. *Introduction to Probability*. Amer. Math. Soc., 1997.

15 A. Guez et al. An investigation of model-free planning. *CoRR*, abs/1901.03559, 2019. `arXiv:1901.03559`.

16 E. M. Hahn, G. Li, S. Schewe, A. Turrini, and L. Zhang. Lazy probabilistic model checking without determinisation. In *Concurrency Theory, (CONCUR)*, pages 354–367, 2015.

17 E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak. Omega-regular objectives in model-free reinforcement learning. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 395–412, 2019. LNCS 11427.

18 E. M. Hahn, M. Perez, S. Schewe, F. Somenzi, A. Trivedi, and D. Wojtczak. Good-for-MDPs automata for probabilistic analysis and reinforcement learning. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 306–323, 2020. LNCS 12078.

19 E. M. Hahn, S. Schewe, A. Turrini, and L. Zhang. A simple algorithm for solving qualitative probabilistic parity games. In *Computer Aided Verification*, Part II, pages 291–311, 2016. LNCS 9780.

20 A. Harding, M. Ryan, and P.-Y. Schobbens. A new algorithm for strategy synthesis in LTL games. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2005)*, pages 477–492, Edinburgh, UK, 2005. LNCS 3440.

21 T. A. Henzinger and N. Piterman. Solving games without determinization. In *15th Conference on Computer Science Logic*, pages 394–409, Szeged, Hungary, September 2006. LNCS 4207.

22 Alex Irpan. Deep reinforcement learning doesn't work yet. `https://www.alexirpan.com/2018/02/14/rl-hard.html`, 2018.

23 J. G. Kemeny and J. L. Snell. *Finite Markov Chains*. Van Nostrand, 1960.

24 M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification (CAV)*, pages 585–591, July 2011. LNCS 6806.

25 M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 157–163, 1994.

26 M. L. Littman and C. Szepesvári. A generalized reinforcement-learning model: Convergence and applications. In *International Conference on Machine Learning*, pages 310–318, 1996.

27 Stochastic parity game reinforcement learning benchmarks. `https://github.com/cuplv/parityRLBenchmarks`, 2020.

28 W. Penney. Problem 95. Penney-ante. *Journal of Recreational Mathematics*, 2(4):241, 1969.

29 D. Perrin and J.-É. Pin. *Infinite Words: Automata, Semigroups, Logic and Games*. Elsevier, 2004.

30 L. S. Shapley. Stochastic games. *Proc. Nat. Acad. Sci. U.S.A.*, 39:1095–1100, 1953.

31 S. Sickert, J. Esparza, S. Jaax, and J. Křetínský. Limit-deterministic Büchi automata for linear temporal logic. In *Computer Aided Verification (CAV)*, pages 312–332, 2016. LNCS 9780.

32 D. Silver et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, January 2016.

33 A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman. PAC model-free reinforcement learning. In *International Conference on Machine Learning, ICML*, pages 881–888, 2006.

34 R. S. Sutton and A. G. Barto. *Reinforcement Learnging: An Introduction*. MIT Press, second edition, 2018.

35 M. Y. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Foundations of Computer Science*, pages 327–338, 1985.

36 Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.

37 M. Wen and U. Topcu. Probably approximately correct learning in stochastic games with temporal logic specifications. In *IJCAI*, pages 3630–3636, 2016.

38 E. Wiewiora. Reward shaping. In *Encyclopedia of Machine Learning*, pages 863–865. Springer, 2010.

39 P. Winkler. *Mathematical Puzzles: A Connoisseur's Collection*. A K Peters, 2004.

Decidability of Cutpoint Isolation for Probabilistic Finite Automata on Letter-Bounded Inputs

Paul C. Bell
Department of Computer Science, Liverpool John Moores University, UK
p.c.bell@ljmu.ac.uk

Pavel Semukhin
Department of Computer Science, University of Oxford, UK
pavel.semukhin@cs.ox.ac.uk

Abstract

We show the surprising result that the cutpoint isolation problem is decidable for probabilistic finite automata where input words are taken from a letter-bounded context-free language. A context-free language $\mathcal{L}$ is letter-bounded when $\mathcal{L} \subseteq a_1^* a_2^* \cdots a_\ell^*$ for some finite $\ell > 0$ where each letter is distinct. A cutpoint is isolated when it cannot be approached arbitrarily closely. The decidability of this problem is in marked contrast to the situation for the (strict) emptiness problem for PFA which is undecidable under the even more severe restrictions of PFA with polynomial ambiguity, commutative matrices and input over a letter-bounded language as well as to the injectivity problem which is undecidable for PFA over letter-bounded languages. We provide a constructive nondeterministic algorithm to solve the cutpoint isolation problem, which holds even when the PFA is exponentially ambiguous. We also show that the problem is at least NP-hard and use our decision procedure to solve several related problems.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory; Theory of computation → Computability; Theory of computation → Probabilistic computation

Keywords and phrases Probabilistic finite automata, cutpoint isolation problem, letter-bounded context-free languages

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.22

Related Version A full version of the paper is available at `https://arxiv.org/abs/2002.07660`.

Acknowledgements We thank the referees for their careful reading of this manuscript and helpful comments.

1 Introduction

Probabilistic finite automata (PFA) are an extension of classical nondeterministic finite automata (NFA) where transitions, for each state and letter, are represented as probability distributions. The PFA model was first introduced by Rabin [23].

There are a variety of classical problems for PFA. Let $\mathcal{P}$ denote a PFA, Σ an alphabet and $\lambda \in [0, 1]$ a probability. The acceptance probability of $\mathcal{P}$ on a word $w \in \Sigma^*$ is denoted $f_{\mathcal{P}}(w)$. A central question is *(strict) emptiness* of cutpoint languages: does there exist a finite input word w for which $f_{\mathcal{P}}(w) \geq \lambda$ (or $f_{\mathcal{P}}(w) > \lambda$ for strict emptiness). Another important problem is that of *cutpoint isolation* – to determine if λ can be approached arbitrarily closely, i.e., for each $\epsilon > 0$, does there exist a word $w \in \Sigma$ such that $|f_{\mathcal{P}}(w) - \lambda| < \epsilon$ (or the converse, does there exist $\delta > 0$ such that $|f_{\mathcal{P}}(w) - \lambda| \geq \delta$ for all $w \in \Sigma^*$)? The *value*-1 problem is a special case of the cutpoint isolation when $\lambda = 1$ [13]. In the *injectivity problem* we must determine if $f_{\mathcal{P}}(w)$ is injective (i.e. do there exist two distinct words with the same acceptance probability?) In the *λ-probability problem* we must determine if there exists $w \in \Sigma^*$ such that $f_{\mathcal{P}}(w) = \lambda$.

The emptiness problem is undecidable for rational matrices [22], even over a binary alphabet when the PFA has dimension 46 [6], later improved to dimension 25 [18]. The injectivity problem for PFA is undecidable [2], even for polynomially ambiguous PFA [4].

The main focus of this paper is the cutpoint isolation problem. The authors of [5] show that the problem of determining if a given cutpoint is isolated (resp. if a PFA has any isolated cutpoint) is undecidable and this was shown to hold even for PFA with 420 (resp. 2354) states over a binary alphabet [6]. The cutpoint isolation problem, in the special case where $\lambda = 1$ (the value-1 problem), is also known to be undecidable [13]. The problem is especially interesting given the seminal result of Rabin that if a cutpoint λ is isolated, then the cutpoint language associated with λ is necessarily regular [23].

Most problems are undecidable for PFA and there exist very few algorithmic solutions [13]. Various classes of restrictions on PFA are possible, related to the number of states, the alphabet size and whether one defines the PFA over the algebraic reals or the rationals. Recent work has studied PFA with finite, polynomial or exponential ambiguity (in terms of the underlying NFA) [10], PFA defined for restricted input words (e.g. those coming from bounded or letter-bounded languages) [2, 3], commutative PFA, where all transition matrices commute, for which cutpoint languages and non-free languages generated by such automata become commutative [4] or other structural restrictions on the PFA such as #-acyclic automata, for which some problems become decidable [13], including the value-1 problem. Such #-acyclic automata impose a restriction on the structure of the PFA (as we shall see, we only restrict the input words).

A natural restriction on PFA was studied in [3], where input words of the PFA are restricted to be from a letter-bounded language (also known as a *letter-monotonic language*) of the form $\mathcal{L} = a_1^* a_2^* \cdots a_\ell^*$ with distinct letters $a_i \in \Sigma$. This is analogous to a 1.5-way PFA, whose read head may "stay put" on an input letter but never moves left. This may model a situation where we have some finite number of probabilistic events and we know that there is a fixed order and number of transitions between them, but with each event being applied an arbitrary number of times. The model is also related to "promise problems" whereby we restrict the decision question to a subset of possible inputs [16]. Letter-bounded languages allow a natural and substantial extension to decision questions on a unary alphabet.

The emptiness and λ-probability problems for PFA on letter-bounded languages were shown to be undecidable for high (finite) dimensional matrices via an encoding of Hilbert's tenth problem on the solvability of Diophantine equations and Turakainen's method to transform weighted integer automata to probabilistic automata [25]. These undecidability results also hold for polynomially ambiguous PFA with commutative matrices [4].

The authors of [10] studied decision problems for PFA of various degrees of ambiguity. The degree of ambiguity (finite, polynomial or exponential) of a PFA is a structural property, giving an indication of the number of accepting runs for a given input word. The degree of ambiguity of automata is a well-known and well-studied property in automata theory [26]. The authors of [10] show that the emptiness problem for PFA remains undecidable even for polynomially ambiguous automata (quadratic ambiguity), show **PSPACE**-hardness results for finitely ambiguous PFA and that emptiness is in **NP** for the class of k-ambiguous PFA for every $k > 0$. The emptiness problem for PFA was later shown to be undecidable for linearly ambiguous automata [9].

1.1 Our Contributions

It is natural to consider the decidability of the cutpoint isolation problem for polynomially ambiguous PFA on letter-bounded or commutative languages, given that the (strict) emptiness problems for such automata are undecidable [4]. In the present paper we prove the surprising

result that the cutpoint isolation problem is in fact *decidable*, even if the PFA is exponentially ambiguous, matrices are non-commutative, and the input language is not just the letter-bounded language $a_1^* \cdots a_\ell^*$ but instead a more general letter-bounded context-free language. The results are shown in Table 1.

Table 1 The decidability of problems under different restrictions on the PFA. The main result of this paper is shown in **boldface**. Symbol $\Longrightarrow$ denotes that decidability is implied by the decidability of the more general model; $\Longleftarrow$ denotes that undecidability is implied by the more restricted model.

Problem	Polynomial ambiguity	Letter-bounded CFL input; Exponential ambiguity	Polynomial ambiguity; letter-bounded input; commutative matrices
(Strict) Emptiness	Undecidable [9, 10, 22] $\Longleftarrow$	$\Longleftarrow$	Undecidable [4]
Cutpoint isolation	Undecidable [5, 10]	**Decidable**	$\Longrightarrow$

The result is surprising since in order to solve the cutpoint isolation problem, we must solve two subproblems. Either the cutpoint λ can be reached exactly (the λ-probability problem), or else it can only be approximated arbitrarily closely and is only reached exactly in some limit. As mentioned, the emptiness problem for cutpoint languages is undecidable for polynomially ambiguous PFA on letter-bounded languages, even when all matrices commute [4]. The proof of this result shows a construction of a PFA for which determining if a given $\lambda \in [0, 1]$ is ever reached (i.e., the λ-probability problem) is undecidable. This may at first seem to contradict the results of this paper, since the λ-probability problem is one of the two subproblems to be solved for cutpoint isolation. Why is there no contradiction then? It comes from the fact that as the powers of matrices used in the PFA constructed in [4] increase, the PFA valuation tends towards the limit value λ. Therefore, this λ is always non-isolated and hence the cutpoint isolation problem for such constructed PFA and λ is decidable. However, determining if the PFA ever *exactly* reaches λ is undecidable. So, there is no contradiction with the results of this paper. Our main result is stated as follows.

▶ **Theorem 1.** *The cutpoint isolation problem for probabilistic finite automata where inputs are constrained to a given letter-bounded context-free language is decidable. Moreover, if the cutpoint is isolated, then a separation bound $\epsilon > 0$ can be computed such that no input word's acceptance probability lies within ϵ of the cutpoint.*

The proof of Theorem 1 is found in Section 3. Our proof technique for showing the decidability of cutpoint isolation for PFA on letter-bounded languages uses the following crucial facts. If a PFA over a letter-bounded context-free language can approach some given cutpoint λ arbitrarily closely, then the PFA can reach λ *exactly* if we allow a subset of the matrices to be taken to one of their "limiting powers". We use the property that each limiting power (of which there may be finitely many) of a stochastic matrix can be computed in polynomial time (see Lemma 6), as well as a crucial property from linear algebra that dominant eigenvalues (those of strictly largest magnitude) of a stochastic matrix are necessarily of magnitude 1, roots of unity and they have equal geometric and algebraic multiplicities (see Lemma 5). Since the input words of the PFA come from a letter-bounded CFL, we also use the fact that a letter-bounded language is context-free if and only if its Parikh image is a stratified semilinear set (see Proposition 3).

The combination of these ideas allows us to derive Algorithm 1, which works as follows. We initially set all variables as free (rather than fixed), and compute the Parikh image $p(\mathcal{L})$ of the given letter-bounded CFL $\mathcal{L}$. Using the fact that $p(\mathcal{L})$ is a semilinear set, we compute which letters can be taken to arbitrarily high powers and which letters have fixed finite values.

We then use the technical Proposition 7 which states that if we can reach λ then we can either do so by setting all free variables to an infinite power (which we denote by ω), or else we can compute an integer C such that the value of one of free variables must be less than C. We then either set all free variables as ω in the first case, or nondeterministically choose one of the free variables and assign it a value less than C in the latter case. In the second case we also update the semilinear set and repeat the above procedure until no free variables remain. Finally, we verify that the PFA has exactly the value λ for the chosen values of the variables.

The crucial Proposition 7 is somewhat technical, but relies on splitting a product of stochastic matrices into a summation involving dominant and subdominant eigenvalues (a subdominant eigenvalue being one with magnitude strictly less than 1) and then applying the spectral decomposition or Jordan normal form of each stochastic matrix in order to derive the constant C which bounds the value of one of free variables.

Combining our proof technique with a result of Rabin [23], we derive the following result.

▶ **Corollary 9.** *The emptiness problem is decidable for probabilistic finite automata on letter-bounded context-free languages when the cutpoint is isolated.*

The undecidability of the emptiness problem for PFA over letter-bounded inputs shown in [4] therefore only applies when the cutpoint is non-isolated.

The provided algorithm is nondeterministic in nature although we do not have an upper bound on its complexity. We can however provide the following lower bound via an adaptation of a proof technique from [4] which proved the NP-hardness of the injectivity problem for linearly ambiguous three-state probabilistic finite automata over letter-bounded languages.

▶ **Theorem 12.** *Cutpoint isolation is NP-hard for* 3*-state PFA on letter-bounded inputs.*

Our procedure also allows us to answer some equivalent problems (in Section 5), for example: given a PFA, $\lambda \in [0,1]$ and a maximum number $k \in \mathbb{N}$ of alternations between input letters, determine if λ is isolated. We also prove the value-1 problem is decidable over letter-bounded context-free language inputs.

2 Preliminaries

2.1 Probabilistic Finite Automata on Letter-Bounded Inputs

We denote by $\mathbb{F}^{n\times n}$ the set of all $n \times n$ matrices over some field $\mathbb{F}$. We will primarily be interested in rational matrices. We use a nonstandard form of *Dirac bra-ket notation* in several calculations, to simplify the notation in some complex formulae. If $u = (u_1, \ldots, u_n)^\top \in \mathbb{C}^n$ is a column vector, then we write $|u\rangle = u$ and $\langle u| = u^\top$ where $u^\top$ denotes the transpose of u, i.e., $\langle u| = (u_1, \ldots, u_n)$. Note that Dirac bra-ket notation ordinarily defines that $\langle u| = u^*$ where u^* denotes the *conjugate* transpose of u, however we will not use this notion at any point. Note that $|u\rangle \langle v|$ is just a rank 1 matrix $u^\top v$. We use $\langle e_i|$ and $|e_i\rangle$ to denote the i'th basis row/column vector respectively.

A PFA $\mathcal{P}$ with n states over an alphabet Σ is defined as $\mathcal{A} = (\langle u|, \{M_a | a \in \Sigma\}, |v\rangle)$ where $\langle u| \in \mathbb{R}^n$ is the initial probability distribution; $|v\rangle \in \{0,1\}^n$ is the final state vector and each $M_a \in \mathbb{R}^{n\times n}$ is a (row) stochastic matrix. For a word $w = w_1 w_2 \cdots w_k \in \Sigma^*$, we define the acceptance probability $f_\mathcal{P} : \Sigma^* \to \mathbb{R}$ of $\mathcal{P}$ as:

$$f_\mathcal{P}(w) = \langle u| M_{w_1} M_{w_2} \cdots M_{w_k} |v\rangle,$$

which denotes the acceptance probability of w.[1]

[1] Some authors interchange the order of u and v and use column stochastic matrices, although the two definitions are trivially isomorphic.

For any $\lambda \in [0,1]$ and PFA $\mathcal{P}$ over alphabet Σ, we define a cutpoint language to be: $L_{\geq\lambda}(\mathcal{P}) = \{w \in \Sigma^* | f_{\mathcal{P}}(w) \geq \lambda\}$, and a strict cutpoint language $L_{>\lambda}(\mathcal{P})$ by replacing $\geq$ with $>$. The (strict) emptiness problem for a cutpoint language is to determine if $L_{\geq\lambda}(\mathcal{P}) = \emptyset$ (resp. $L_{>\lambda}(\mathcal{P}) = \emptyset$). Our main focus is on the *cutpoint isolation problem*, now defined.

▶ **Problem 2** (Cutpoint isolation). *Given a PFA $\mathcal{P}$ and cutpoint $\lambda \in [0,1]$, determine if for each $\epsilon > 0$ there exists some $w \in \Sigma^*$ such that $|f_{\mathcal{P}}(w) - \lambda| < \epsilon$.*

Let $\Sigma = \{a_1, a_2, \ldots, a_\ell\}$ be an alphabet with $\ell > 0$ distinct letters. A language $\mathcal{L}$ is called *letter-bounded* if $\mathcal{L} \subseteq a_1^* a_2^* \cdots a_\ell^*$. If $\mathcal{L}$ is letter-bounded and also a context-free language, then it is called a letter-bounded context-free language. We are interested in cutpoint isolation for PFA whose inputs come from a given letter-bounded context-free language.

For a letter-bounded language $\mathcal{L} \subseteq a_1^* a_2^* \cdots a_\ell^*$, define its *Parikh image*[2] as

$$p(\mathcal{L}) = \{ (k_1, \ldots, k_\ell) \ : \ a_1^{k_1} a_2^{k_2} \cdots a_\ell^{k_\ell} \in \mathcal{L} \}.$$

Recall that a subset $Q \subseteq \mathbb{N}^\ell$ is called *linear* if there are vectors $q_0, q_1, \ldots, q_r \in \mathbb{N}^\ell$ such that

$$Q = \{ q_0 + t_1 q_1 + \cdots + t_r q_r \ : \ t_1, \ldots, t_r \in \mathbb{N} \}.$$

We say that a linear set Q is *stratified* if for each $i \geq 1$ the vector q_i has at most two nonzero coordinates, and for any $i, j \geq 1$ if both q_i and q_j have two nonzero coordinates, $i_1 < i_2$ and $j_1 < j_2$, respectively, then their order is *not* $i_1 < j_1 < i_2 < j_2$, i.e., they are not interlaced. A finite union of linear sets is called a *semilinear set*, and a finite union of stratified linear sets is called a *stratified semilinear set*.

We will need the following classical fact about context-free languages.

▶ **Proposition 3.** *If $\mathcal{L}$ is a context-free language, then its Parikh image $p(\mathcal{L})$ is a semilinear set that can be effectively constructed from the definition of $\mathcal{L}$ [21].*

▶ Remark 4. There is a nice characterization of the letter-bounded context-free languages. Namely, a letter-bounded language $\mathcal{L} \subseteq a_1^* a_2^* \cdots a_\ell^*$ is context-free if and only if $p(\mathcal{L})$ is a stratified semilinear set [14, 15].

Let $A_1, \ldots, A_\ell \in \mathbb{Q}^{n \times n}$ be row stochastic matrices. Let $u \in \mathbb{Q}^n$ be a stochastic vector (the initial vector) and $v \in \{0,1\}^n$ (the final state vector). Let $\mathcal{L} \subseteq a_1^* a_2^* \cdots a_\ell^*$ be a letter-bounded context-free language, and let $\lambda \in [0,1]$ be a cutpoint for which we want to decide if it is isolated or not, that is, whether λ belongs to the closure of $\{ \langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle \ : \ a_1^{k_1} a_2^{k_2} \cdots a_\ell^{k_\ell} \in \mathcal{L} \}$.

If λ is not isolated, then there are two scenarios: either there exists $k_1, k_2, \ldots, k_\ell \in \mathbb{N}$ such that $\langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle = \lambda$, or else λ is never reached but only approached arbitrarily closely. In the second case there is a sequence of tuples $\{(k_1^m, k_2^m, \ldots, k_\ell^m)\}_{m=1}^\infty$ such that

$$\lambda = \lim_{m \to \infty} \langle u| A_1^{k_1^m} A_2^{k_2^m} \cdots A_\ell^{k_\ell^m} |v\rangle$$

and, furthermore, for every $t \in \{1, \ldots, \ell\}$, either $k_t^m = k_t^1$ for all $m \geq 1$, i.e. k_t^m is fixed, or k_t^m is strictly increasing and $A_t^{k_t^m}$ converges to a limit as $m \to \infty$.

[2] In general, the Parikh image of $\mathcal{L} \subseteq \Sigma^*$ is defined as $p(\mathcal{L}) = \{ (|w|_{a_1}, \ldots, |w|_{a_\ell}) \ : \ w \in \mathcal{L} \}$ where $|w|_{a_i}$ denotes the number of occurrences of letter a_i in word w.

We will use the notation A^ω to denote the set of all limits of the sequence $\{A^k\}_{k=1}^\infty$ (see Lemma 6 below for a detailed explanation). It follows that if λ is not isolated, then there exists a choice of variables $k_1, k_2, \ldots, k_\ell \in \mathbb{N} \cup \{\omega\}$ such that

$$\lambda \in \langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle .$$

Note that if $k_t = \omega$, then A_t^ω is a finite *set*. In this case we substitute all limits of A_t^ω in the above formula, and so $\langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle$ also becomes a finite set.[3]

2.2 Algebraic numbers

A complex number α is *algebraic* if it is a root of a polynomial $p \in \mathbb{Z}[x]$. The *defining polynomial* $p_\alpha \in \mathbb{Z}[x]$ for α is the unique polynomial of least degree with positive leading coefficient such that the coefficients of p_α do not have a common factor and $p_\alpha(\alpha) = 0$. The *degree* and *height* of α are defined to be that of p_α.

In order to do computations with algebraic numbers we use their standard representations. Namely, an algebraic number can be represented by its defining polynomial and a sufficiently good complex rational approximation. More precisely, α will be represented by a tuple (p_α, a, b, r), where $p_\alpha \in \mathbb{Z}[x]$ is the defining polynomial for α and $a, b, r \in \mathbb{Q}$ are such that α is the unique root of p_α inside the circle in $\mathbb{C}$ with centre $a + bi$ and radius r. As shown in [19], if $\alpha \neq \beta$ are roots of $p \in \mathbb{Z}[x]$, then $|\alpha - \beta| > \frac{\sqrt{6}}{d^{(d+1)/2} H^{d-1}}$, where d and H are the degree and height of p, respectively. So, if we require r to be smaller than half of this bound, the above representation is well-defined.

Let $||\alpha||$ be the size of the standard representation of α, that is, the total bit size of a, b, r and the coefficients of p_α. It is well-known fact that for given algebraic numbers α and β, one can compute $1/\alpha$, $\bar{\alpha}$ and $|\alpha|$ in time polynomial in $||\alpha||$, and one can compute $\alpha + \beta$ and $\alpha\beta$ and decide whether $\alpha = \beta$ in time polynomial in $||\alpha|| + ||\beta||$. Moreover, for a *real* algebraic α, deciding whether $\alpha > 0$ can be done in time polynomial in $||\alpha||$. Finally, there is a polynomial time algorithm that for a given $p \in \mathbb{Z}[x]$ computes the standard representations of all roots of p. For more information on efficient algorithmic computations with algebraic numbers the reader is referred to [1, 8, 17, 20].

2.3 Spectral decomposition and Jordan normal forms

We define the spectrum (set of eigenvalues) of $A \in \mathbb{R}^{n \times n}$ as $\sigma(A) = \{\lambda_1, \ldots, \lambda_n\}$ arranged in monotonically nonincreasing order, i.e. $|\lambda_i| \geq |\lambda_j|$ for all $1 \leq i < j \leq n$ and we define $\hat{\sigma}(A) \subseteq \sigma(A)$ as the set of eigenvalues of A of absolute value 1. We call eigenvalues $\hat{\sigma}(A)$ *dominant eigenvalues* and eigenvalues $\sigma(A) \setminus \hat{\sigma}(A)$ *subdominant eigenvalues*.

Given $A = (a_{ij}) \in \mathbb{F}^{m \times m}$ and $B \in \mathbb{F}^{n \times n}$, we define the direct sum $A \oplus B$ of A and B by: $A \oplus B = \left[\begin{array}{c|c} A & \mathbf{0}_{m,n} \\ \hline \mathbf{0}_{n,m} & B \end{array}\right]$, where $\mathbf{0}_{n,m}$ is the $n \times m$ zero matrix.

We will use both the *spectral decomposition theorem* and the *Jordan normal form* of stochastic matrices in later proofs. For background, see [11].

[3] If $\{A^k\}_{k=1}^\infty$ has a unique limit A', then we will identify the set $A^\omega = \{A'\}$ with the matrix A' and write $A^\omega = A'$. Also, if all k_t's are finite or if all limits are unique, we identify the number $\langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle$ with the one element set $\{\langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle\}$.

Let $A_i \in \mathbb{Q}^{n\times n}$ be a matrix (we use notation A_i since it will prove useful in the proof of Proposition 7), and let $\{\lambda_{i,1},\ldots,\lambda_{i,n_i}\}$ be the eigenvalues of A_i listed according to their *geometric* multiplicities[4]. Then A_i can be written in *Jordan normal form* $A_i = S_i^{-1}(J_{\ell_{i,1}}(\lambda_{i,1}) \oplus \cdots \oplus J_{\ell_{i,n_i}}(\lambda_{i,n_i}))S_i$, where S_i is an invertible matrix ($\det(S_i) \neq 0$) and $J_{\ell_{i,j}}(\lambda_{i,j})$ is a $\ell_{i,j} \times \ell_{i,j}$ *Jordan block* for $1 \leq j \leq n_i \leq n$, with n_i the number of Jordan blocks of A_i and $\ell_{i,j}$ the size of the Jordan block corresponding to eigenvalue $\lambda_{i,j}$, such that $\ell_{i,1} + \cdots + \ell_{i,n_i} = n$. Jordan block $J_{\ell_{i,j}}(\lambda_{i,j})$ corresponds to the j^{th} eigenvalue $\lambda_{i,j}$ of A_i and has the form:

$$J_{\ell_{i,j}}(\lambda_{i,j}) = \begin{pmatrix} \lambda_{i,j} & 1 & 0 & \cdots & 0 \\ 0 & \lambda_{i,j} & 1 & \cdots & 0 \\ 0 & 0 & \lambda_{i,j} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_{i,j} \end{pmatrix} \in \mathbb{C}^{\ell_{i,j}\times\ell_{i,j}}$$

The matrix S_i contains the *generalised* eigenvectors of A_i. Noting that $\binom{x}{y} = 0$ if $y > x$, we now see that

$$J_{\ell_{i,j}}(\lambda_{i,j})^{k_i} = \begin{pmatrix} \lambda_{i,j}^{k_i} & \binom{k_i}{1}\lambda_{i,j}^{k_i-1} & \binom{k_i}{2}\lambda_{i,j}^{k_i-2} & \cdots & \binom{k_i}{\ell_{i,j}-1}\lambda_{i,j}^{k_i-(\ell_{i,j}-1)} \\ 0 & \lambda_{i,j}^{k_i} & \binom{k_i}{1}\lambda_{i,j}^{k_i-1} & \cdots & \binom{k_i}{\ell_{i,j}-2}\lambda_{i,j}^{k_i-(\ell_{i,j}-2)} \\ 0 & 0 & \lambda_{i,j}^{k_i} & \cdots & \binom{k_i}{\ell_{i,j}-3}\lambda_{i,j}^{k_i-(\ell_{i,j}-3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_{i,j}^{k_i} \end{pmatrix} \in \mathbb{C}^{\ell_{i,j}\times\ell_{i,j}}$$

$$= \sum_{0\leq m\leq \ell_{i,j}-1} \lambda_{i,j}^{k_i-m}\binom{k_i}{m}\left(\sum_{1\leq p\leq \ell_{i,j}-m} |e_p\rangle\langle e_{m+p}|\right) \tag{1}$$

The *spectral decomposition* of a matrix is a special case of the Jordan normal form. Namely, any *diagonalizable* matrix $A_i \in \mathbb{Q}^{n\times n}$ can be written as

$$A_i = S_i^{-1}(\lambda_{i,1} \oplus \cdots \oplus \lambda_{i,n})S_i = \sum_{j=1}^{n} \lambda_{i,j} |v_{i,j}\rangle\langle u_{i,j}|, \tag{2}$$

where $\sigma(M) = \{\lambda_{i,1},\ldots,\lambda_{i,n}\}$ is the set of eigenvalues of A_i, $|v_{i,j}\rangle$ is the j'th column of S_i^{-1} and $\langle u_{i,j}|$ is the j'th row of S_i. Thus we have $A_i^k = \sum_{j=1}^{n} \lambda_{i,j}^k |v_{i,j}\rangle\langle u_{i,j}|$.

We will also require the following technical lemma concerning the dominant eigenvalues of stochastic matrices.

▶ **Lemma 5** ([11, Theorem 6.5.3]). *Let λ be a dominant eigenvalue of a stochastic matrix $A \in \mathbb{R}^{n\times n}$. Then λ is a root of unity of order no more than n. Moreover, the geometric multiplicity of λ is equal to its algebraic multiplicity. In other words, the Jordan blocks that correspond to λ have size 1×1.*

We also require the following lemma.

[4] Note that n_i is the number of linearly independent eigenvectors of A_i or the number of Jordan blocks in the Jordan normal form of A_i. The matrix A_i is diagonalizable if and only if $n_i = n$. Jordan normal forms are unique up to permutations of the Jordan blocks.

▶ **Lemma 6.** *For any stochastic matrix A, the sequence $\{A^k\}_{k=1}^{\infty}$ has a finite number of limits. Namely, there exist a* computable *constant d such that, for each $r = 0, \ldots, d-1$, the subsequence $\{A^{dm+r}\}_{m=1}^{\infty}$ converges to a limit, and this limit can be computed in polynomial time given d and r.*

Proof. Let A be a stochastic matrix. As shown in [7], we can compute in polynomial time the Jordan normal form of A and a transformation matrix S such that $A = S^{-1}JS$. Note that A may have complex eigenvalues, so all computations are done using standard representations of algebraic numbers as explained in Section 2.2.

By Lemma 5, all dominant eigenvalues of A are roots of unity of orders no more than n, and their Jordan blocks have size 1×1. If λ is a root of unity of order p, then $\{\lambda^k\}_{k=1}^{\infty}$ is a periodic sequence with period p. On the other hand, if $J_\ell(\lambda)$ is a Jordan block corresponding to an eigenvalue λ such that $|\lambda| < 1$, then $\lim_{k\to\infty} J_\ell(\lambda)^k$ is equal to the zero matrix.

Let d be the least common multiple of the orders of the roots of unity among the eigenvalues of A. Now if λ is a dominant eigenvalue of A, then the values of $\lambda^{dm+r} = \lambda^r$ do not depend on m, where $r = 0, \ldots, d-1$. Hence J^{dm+r} converges to a limit when $m \to \infty$. This limit is equal to a matrix J' obtained from J by replacing all dominant λ with λ^r and all Jordan blocks corresponding to subdominant eigenvalues with zero matrices. So, $\lim_{m\to\infty} A^{dm+r} = S^{-1}J'S$.

This shows that $\{A^k\}_{k=1}^{\infty}$ has at most d limits. Finally, we note that d may be exponential in the dimension of A. However, if $\{A^k\}_{k=1}^{\infty}$ has a single limit, then this limit can be computed in polynomial time. ◀

3 Decidability of Cutpoint Isolation

In this section we will give a proof of Theorem 1 which is our main result. The crucial ingredient of our proof is the following technical proposition which will be proven in Section 4.

▶ **Proposition 7.** *Let $J = \{1, 2, \ldots, \ell\}$ be indices, $\lambda \in [0, 1]$ a cutpoint, and let $J_F \subseteq J$ be such that k_t is a free variable, for $t \in J_F$, and k_t is assigned a fixed finite value, for $t \in J \setminus J_F$. Then*

- *either $\lambda \in \langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle$, where $k_t = \omega$ for all $t \in J_F$,*
- *or else there exists a constant $C > 0$ such that $\lambda \in \langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle$ implies $k_t < C$ for at least one $t \in J_F$.*

Moreover, we can decide whether the first case holds and compute the constant C in the second case.

Below we give a high-level description of the main algorithm (Algorithm 1), which gives a formal proof of Theorem 1, and explain how Proposition 7 is used there.

Let $\mathcal{L} \subseteq a_1^* a_2^* \cdots a_\ell^*$ be a given letter-bounded CFL. We start by considering all indices $J = \{1, \ldots, \ell\}$ as *free* (i.e. their value is not fixed and they will later be given a fixed value from $\mathbb{N} \cup \{\omega\}$) and iteratively fix them until no free indices remain. We first use Parikh and Ginsburg's results (Proposition 3) to compute the Parikh image $p(\mathcal{L})$. Then we nondeterministically choose a linear subset Q and use it to determine the indices which can be taken to arbitrary high values while staying within Q. These indices will correspond to the "free variables" in the algorithm.

Let J_F be a set of such indices (which will be called R in Algorithm 1). We then set k_t for $t \in J \setminus J_F$ to appropriate finite values, while k_t with $t \in J_F$ remain free variables. We wish to determine if there is a choice of $k_t \in \mathbb{N} \cup \{\omega\}$ for $t \in J_F$ such that

$$\lambda \in \langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle ,$$

that is, whether λ can be reached by setting each free variable k_t either to some finite value or else to ω, an "infinite" power. Proposition 7 then tells us that either all free variables should be set at ω in order to reach λ (and this is decidable), or else there exists a computable constant C such that *if* we can reach λ by some choice of these free variables, then some $k_t < C$ for an index $t \in J_F$.

In the first case, we set all free variables to ω. In the second case, we nondeterministically choose some free variable, fix its value in the range $[0, C)$ and then update our linear set Q to satisfy a new constraint. The procedure repeats iteratively until all free variables have been assigned a fixed value. The algorithm then verifies if this choice of variables gives a solution.

Algorithm 1 Nondeterministic algorithm deciding whether a given cutpoint is isolated.

Stage one (Nondeterministic iterative fixing of free variables):

Let $T = J = \{1, 2, \ldots, \ell\}$.

Compute the Parikh image $p(\mathcal{L})$ and nondeterministically choose one of its finitely many linear subsets $Q = \{\, q_0 + t_1 q_1 + \cdots + t_r q_r \;:\; t_1, \ldots, t_r \in \mathbb{N} \,\} \subseteq p(\mathcal{L})$.[5]

while $T \neq \emptyset$ **do**

Let R be the set of indices $j \in T$ such that at least one q_i with $i \geq 1$ has a nonzero jth coordinate.[6]

For each $j \in T \setminus R$, the jth coordinate of all vectors from Q is equal to the jth coordinate of q_0.[7] So, we set k_j for $j \in T \setminus R$ to be the jth coordinate of q_0.

Then, for $j \in R$, compute the limits A_j^ω with indices respecting set Q (see Remark 8 below for details).

Check whether $\lambda \in \langle u|\, A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell}\, |v\rangle$, where $k_j = \omega$ for all $j \in R$.

If yes, return True and stop.

Otherwise, assuming all indices k_j for $j \in J \setminus R$ are fixed and R is the set of free variables, use Proposition 7 to compute the constant $C > 0$ such that $\lambda \in \langle u|\, A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell}\, |v\rangle$ implies $k_j < C$ for at least one $j \in R$.

Then nondeterministically choose $j \in R$, fix $k_j \in [0, C)$ and set $T \leftarrow R \setminus \{j\}$.

Next, for the chosen index j, find those indices i in $\{1, \ldots, r\}$ for which q_i has a nonzero jth coordinate. Without loss of generality, suppose $\{1, \ldots, s\}$ are these indices.

Fixing $k_j \in [0, C)$, restricts the parameters $t_1, \ldots, t_s$ in Q to a finite set of possible values since the vector $q_0 + t_1 q_1 + \cdots + t_s q_s$ must have k_j in its jth coordinate. Nondeterministically choose one of these values for $t_1, \ldots, t_s$ or return False and stop, if such a choice is impossible.

Let $Q \leftarrow \{\, (q_0 + t_1 q_1 + \cdots + t_s q_s) + t_{s+1} q_{s+1} + \cdots + t_r q_r \;:\; t_{s+1}, \ldots, t_r \in \mathbb{N} \,\}$.[8]

Stage two (Verifying the computation):

At this stage we have fixed all variables $k_1, k_2, \ldots, k_\ell$ to some finite values.

Compute $\langle u|\, A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell}\, |v\rangle$ for the obtained values of $k_1, \ldots, k_\ell \in \mathbb{N}$.

Return True if $\lambda = \langle u|\, A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell}\, |v\rangle$ or False, otherwise.

End.

▶ Remark 8. To compute the limits A_j^ω with indices respecting set Q, note that the projection of Q on the jth coordinate is equal to $\{\, q_{0,j} + t_1 q_{1,j} + \cdots + t_r q_{r,j} \;:\; t_1, \ldots, t_r \in \mathbb{N} \,\}$, where $q_{i,j}$ is the jth coordinate of q_i. The set $\langle q_{1,j}, \ldots, q_{r,j} \rangle = \{t_1 q_{1,j} + \cdots + t_r q_{r,j} \;:\; t_1, \ldots, t_r \in \mathbb{N}\}$ is a finitely generated subsemigroups of $(\mathbb{N}, +)$. Let $d = \gcd(q_{1,j}, \ldots, q_{r,j})$, then there is a number $s > 0$ such that for any $t \geq s$, we have $t \in \langle q_{1,j}, \ldots, q_{r,j} \rangle$ if and only if d divides t.

This is a well-known property of the subsemigroups of $(\mathbb{N}, +)$ [24]. Thus the limits A_j^ω with indices respecting Q are equal to $A_j^\omega = A_j^{q_{0,j}}(A_j^d)^\omega$, where the limits $(A_j^d)^\omega$ are computed using Lemma 6.

It remains to prove that we can compute a separation bound $\epsilon > 0$ between λ and the closest acceptance probability of $\mathcal{P}$ for any input word $w \in \mathcal{L}$. Algorithm 1 has two stopping conditions, either by returning True in **Stage one** (which we discount since it implies the cutpoint is not isolated), or else after **Stage two**.

Algorithm 1 has two sources of nondeterminism in **Stage one**: in the choice of linear subset Q and then during the while loop in the choice of $j \in R$ and $k_j \in [0, C)$. We will evaluate every choice of linear subset Q and every choice of j and k_j to cover all possible cases, updating a global variable ϵ at the end of every nondeterministic branch. Initially, we set $\epsilon \leftarrow \infty$, and let ϵ_1, ϵ_2 be additional global variables that are set $\epsilon_1 \leftarrow \epsilon_2 \leftarrow \infty$ at the beginning of every nondeterministic branch.

During the execution of **Stage one** we use Proposition 7 to compute C such that if all free variables are above C then we are at least some $\epsilon' > 0$ away from λ. Note that ϵ' is less than half of the distance between λ and some limit values. For each iteration of the while loop, we set $\epsilon_1 \leftarrow \min\{\epsilon_1, \epsilon'\}$ to keep track of the minimal value. During **Stage two**, all variables have a fixed finite value, and we set $\epsilon_2 \leftarrow |\langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle - \lambda|$ which is greater than zero assuming λ is isolated. Finally, we set $\epsilon \leftarrow \min\{\epsilon, \epsilon_1, \epsilon_2\} > 0$.

After inspecting all possible nondeterministic runs of the algorithm, the obtained value of ϵ gives us the separation bound. Indeed, during the execution of the above procedure, ϵ is updated to the minimum of ϵ_1 and ϵ_2, where ϵ_1 is less than half of the distance between λ and some limit values and ϵ_2 keeps track of the distance between λ and the values $\langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle$ when each index k_j is less than the corresponding constant C.

4 Proof of Proposition 7

We begin with a proof sketch. Since each A_i is stochastic, $\hat{\sigma}(A_i)$ contains at least one eigenvalue 1 and all other eigenvalues in $\hat{\sigma}(A_i)$ are roots of unity by Lemma 5. All eigenvalues in $\sigma(A_i) \setminus \hat{\sigma}(A_i)$ have absolute value strictly smaller than 1. Our approach is to rewrite the expression

$$\langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle \tag{3}$$

into the sum of two terms (which will be denoted by S_0 and S_1) such that S_0 determines the limit behaviour as all free variables tend towards infinity, since they control only dominant eigenvalues, while S_1 is vanishing, since at least one free variable controls a subdominant eigenvalue. We can then reason that if all free variables simultaneously become larger, then Eqn (3) tends towards a set of computable limits with some vanishing terms. Therefore we can determine either that we can reach λ when all free variables are ω, or else we can prove that Eqn (3) is within any $\epsilon > 0$ of a limit value once all free variables are sufficiently large, which proves the proposition (by setting ϵ as less than the smallest difference from a limit value and λ). We now proceed with the formal details.

First, we consider the simpler case when all matrices are diagonalizable and then show how to extend this argument to the general case.

Diagonalizable matrices. Let us first assume that all matrices are diagonalizable. By the spectral decomposition theorem (see Eqn (2)), we may write a matrix $A_i^{k_i}$ as:

$$A_i^{k_i} \quad = \quad \sum_{j=1}^{n} \lambda_{i,j}^{k_i} |v_{i,j}\rangle \langle u_{i,j}| , \tag{4}$$

where $\{\lambda_{i,1}, \ldots, \lambda_{i,n}\}$ are the eigenvalues of A_i repeated according to their multiplicities, and the vectors $|v_{i,j}\rangle$ and $\langle u_{i,j}|$, for $1 \leq j \leq n$, are related to the eigenvectors of A_i. Now, we can write:

$$\langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle = \langle u| \left(\prod_{i=1}^{\ell} \left(\sum_{j=1}^{n} \lambda_{i,j}^{k_i} |v_{i,j}\rangle \langle u_{i,j}| \right) \right) |v\rangle$$

$$= \sum_{j_1,\ldots,j_\ell \in [1,n]} \lambda_{1,j_1}^{k_1} \lambda_{2,j_2}^{k_2} \cdots \lambda_{\ell,j_\ell}^{k_\ell} \langle u|v_{1,j_1}\rangle \langle u_{1,j_1}|v_{2,j_2}\rangle \langle u_{2,j_2}| \cdots |v_{\ell,j_\ell}\rangle \langle u_{\ell,j_\ell}|v\rangle$$

Let us thus define $\Theta_{j_1,\ldots,j_\ell} = \langle u|v_{1,j_1}\rangle \langle u_{1,j_1}|v_{2,j_2}\rangle \langle u_{2,j_2}| \cdots |v_{\ell,j_\ell}\rangle \langle u_{\ell,j_\ell}|v\rangle$. The above sum can be split in two: the first summand containing terms where only dominant eigenvalues are to the power of free variables, and the second containing terms with at least one subdominant eigenvalue to the power of a free variable (these two terms are labelled S_0 and S_1 below). This is a useful decomposition since any term which contains a subdominant eigenvalue taken to the power of a free variable will tend towards zero as the values of all free variables (simultaneously) increase. Thus we can write $\langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle = S_0 + S_1$, where

$$S_0 = \sum_{\substack{j_1,\ldots,j_\ell \in [1,n] \\ \forall t \in J_F \,:\, |\lambda_{t,j_t}|=1}} \lambda_{1,j_1}^{k_1} \lambda_{2,j_2}^{k_2} \cdots \lambda_{\ell,j_\ell}^{k_\ell} \Theta_{j_1,\ldots,j_\ell},$$

$$S_1 = \sum_{\substack{j_1,\ldots,j_\ell \in [1,n] \\ \exists t \in J_F \,:\, |\lambda_{t,j_t}|<1}} \lambda_{1,j_1}^{k_1} \lambda_{2,j_2}^{k_2} \cdots \lambda_{\ell,j_\ell}^{k_\ell} \Theta_{j_1,\ldots,j_\ell}.$$

By Lemma 5 the dominant eigenvalues are roots of unity, and so S_0 assumes only finitely many different values as k_t with $t \in J_F$ vary, while k_t with $t \in J \setminus J_F$ are fixed.

Suppose S_1 in not an empty sum since otherwise $S_1 = 0$. Then there exists $t \in J_F$ and $j_t \in [1, n]$ such that $|\lambda_{t,j_t}| < 1$. Let ρ be the maximum among such values, that is,

$$\rho = \max\{\, |\lambda_{t,j}| \;:\; t \in J_F,\ j \in [1,n] \text{ and } |\lambda_{t,j}| < 1 \,\}.$$

Suppose $k_t \geq C$ for $t \in J_F$, where C is some constant to be chosen later. Then S_1 can be estimated as follows: since for every choice of $j_1, \ldots, j_\ell$ in the summation S_1 there is $t \in J_F$ with $|\lambda_{t,j_t}| \leq \rho < 1$ and $|\lambda_{i,j}| \leq 1$ for all other $\lambda_{i,j}$, we have

$$|S_1| \leq C_1 \rho^C, \quad \text{where} \quad C_1 = \sum_{\substack{j_1,\ldots,j_\ell \in [1,n] \\ \exists t \in J_F \,:\, |\lambda_{t,j_t}|<1}} |\Theta_{j_1,\ldots,j_\ell}|.$$

Notice that for any rational $\epsilon > 0$, we can compute $C \in \mathbb{N}$ such that $|S_1| \leq C_1 \rho^C < \epsilon$. Now, S_0 gives a finite number of limit values for $\langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle$. If λ is not equal to any of them, then choose $\epsilon > 0$ to be less than half the minimal distance between λ and those limit values. Using this ϵ, we compute C as above. By definition of C, if all $k_t \geq C$ for $t \in J_F$, then the distance between $\langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle$ and one of the limit values of S_0 is less than ϵ. Thus $\langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle$ cannot be equal to λ when all $k_t \geq C$ for $t \in J_F$. Hence if $\lambda = \langle u| A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell} |v\rangle$, then there is $t \in J_F$ such that $k_t < C$.

The general case. We now show how to extend the proof to the case when some matrices are non-diagonalizable.

Let $a_{i,j} = \sum_{s=1}^{j-1} \ell_{i,s}$ be the sum of the sizes of the first $j-1$ Jordan blocks of matrix A_i, so that $a_{i,1} = 0, a_{i,2} = \ell_{i,1}, a_{i,3} = \ell_{i,1} + \ell_{i,2}$ etc. Then we see that by using Eqn (1), $A_i^{k_i} = S_i^{-1}(J_{\ell_{i,1}}(\lambda_{i,1})^{k_i} \oplus \cdots \oplus J_{\ell_{i,n_i}}(\lambda_{i,n_i})^{k_i})S_i$ has the form

$$
\begin{aligned}
& S_i^{-1}\left(\sum_{1\le j\le n_i}\ \sum_{0\le m\le \ell_{i,j}-1} \lambda_{i,j}^{k_i-m}\binom{k_i}{m}\left(\sum_{1\le p\le \ell_{i,j}-m} |e_{a_{i,j}+p}\rangle\langle e_{a_{i,j}+m+p}|\right)\right) S_i \\
= & \sum_{1\le j\le n_i}\ \sum_{0\le m\le \ell_{i,j}-1} \lambda_{i,j}^{k_i-m}\binom{k_i}{m}\left(\sum_{1\le p\le \ell_{i,j}-m} S_i^{-1}|e_{a_{i,j}+p}\rangle\langle e_{a_{i,j}+m+p}| S_i\right) \\
= & \sum_{1\le j\le n_i}\ \sum_{0\le m\le \ell_{i,j}-1} \lambda_{i,j}^{k_i-m}\binom{k_i}{m}\left(\sum_{1\le p\le \ell_{i,j}-m} |v_{i,a_{i,j}+p}\rangle\langle u_{i,a_{i,j}+m+p}|\right),
\end{aligned}
$$

where $S_i = \sum_{q=1}^{n} |e_q\rangle\langle u_{i,q}|$ and $S_i^{-1} = \sum_{q=1}^{n} |v_{i,q}\rangle\langle e_q|$, with e_q the q^{th} basis vector. Here we used the property that $\langle e_i|e_j\rangle = 0$ for any $i \neq j$. We may now compute that:

$$
\begin{aligned}
& \langle u| A_1^{k_1} A_2^{k_2}\cdots A_\ell^{k_\ell} |v\rangle \\
= & \langle u|\prod_{i=1}^{\ell}\left(\sum_{1\le j\le n_i}\ \sum_{0\le m\le \ell_{i,j}-1} \lambda_{i,j}^{k_i-m}\binom{k_i}{m}\left(\sum_{1\le p\le \ell_{i,j}-m} |v_{i,a_{i,j}+p}\rangle\langle u_{i,a_{i,j}+m+p}|\right)\right)|v\rangle \\
= & \langle u|\prod_{i=1}^{\ell}\left(\sum_{1\le j\le n_i}\ \sum_{0\le m\le \ell_{i,j}-1} \lambda_{i,j}^{k_i-m}\binom{k_i}{m}\Psi_{i,j,m}\right)|v\rangle \\
= & \sum_{\substack{j_1,\ldots,j_\ell \ \mid\ j_q\in[1,n_q] \\ m_1,\ldots,m_\ell \ \mid\ m_q\in[0,\ell_{q,j_q}-1]}}\left(\prod_{1\le t\le \ell}\lambda_{t,j_t}^{k_t-m_t}\binom{k_t}{m_t}\right)\langle u|\,\Psi_{j_1,\ldots,j_\ell}^{m_1,\ldots,m_\ell}|v\rangle,
\end{aligned}
$$

where $\Psi_{i,j,m} = \sum_{1\le p\le \ell_{i,j}-m} |v_{i,a_{i,j}+p}\rangle\langle u_{i,a_{i,j}+m+p}|$ and

$$\Psi_{j_1,\ldots,j_\ell}^{m_1,\ldots,m_\ell} = \Psi_{1,j_1,m_1}\Psi_{2,j_2,m_2}\cdots\Psi_{\ell,j_\ell,m_\ell}.$$

We may split the above summation, as before, into two parts corresponding to products containing only dominant eigenvalues to powers of free variables and those containing at least one subdominant eigenvalue to the power of a free variables. By Lemma 5, Jordan blocks corresponding to dominant eigenvalues have size 1×1, that is, if $|\lambda_{t,j_t}| = 1$ for $t \in J_F$, then $\ell_{t,j_t} = 1$ and hence $m_t = 0$. So, we can write

$$\langle u| A_1^{k_1} A_2^{k_2}\cdots A_\ell^{k_\ell}|v\rangle = S_0 + S_1,$$

where

$$
\begin{aligned}
S_0 &= \sum_{\substack{j_1,\ldots,j_\ell \ \mid\ j_q\in[1,n_q] \\ m_1,\ldots,m_\ell \ \mid\ m_q\in[0,\ell_{q,j_q}-1] \\ \forall t\in J_F\ :\ |\lambda_{t,j_t}|=1}}\left(\prod_{t\in J_F}\lambda_{t,j_t}^{k_t}\right)\Theta_{j_1,\ldots,j_\ell}^{m_1,\ldots,m_\ell}, \\
S_1 &= \sum_{\substack{j_1,\ldots,j_\ell \ \mid\ j_q\in[1,n_q] \\ m_1,\ldots,m_\ell \ \mid\ m_q\in[0,\ell_{q,j_q}-1] \\ \exists t\in J_F\ :\ |\lambda_{t,j_t}|<1}}\left(\prod_{t\in J_F}\lambda_{t,j_t}^{k_t-m_t}\binom{k_t}{m_t}\right)\Theta_{j_1,\ldots,j_\ell}^{m_1,\ldots,m_\ell} \qquad \text{and} \qquad (5) \\
\Theta_{j_1,\ldots,j_\ell}^{m_1,\ldots,m_\ell} &= \left(\prod_{t\in J\setminus J_F}\lambda_{t,j_t}^{k_t-m_t}\binom{k_t}{m_t}\right)\langle u|\,\Psi_{j_1,\ldots,j_\ell}^{m_1,\ldots,m_\ell}|v\rangle.
\end{aligned}
$$

Note that k_t's in the formula for $\Theta^{m_1,\ldots,m_\ell}_{j_1,\ldots,j_\ell}$ are fixed since $t \notin J_F$ and so A_t is not a free matrix. In other words, $\Theta^{m_1,\ldots,m_\ell}_{j_1,\ldots,j_\ell}$ does not depend on free variables k_i for $t \in J_F$. This also implies that S_0 assumes only finitely many different values as k_t with $t \in J_F$ vary since by Lemma 5 the dominant eigenvalues are roots of unity.

Again using the fact that Jordan blocks corresponding to the dominant eigenvalues have size 1×1, we can rewrite the product inside the formula for S_1 from Eqn (5) as follows

$$\prod_{t\in J_F} \lambda_{t,j_t}^{k_t-m_t}\binom{k_t}{m_t} = \prod_{\substack{t\in J_F\\ |\lambda_{t,j_t}|=1}} \lambda_{t,j_t}^{k_t} \cdot \prod_{\substack{t\in J_F\\ |\lambda_{t,j_t}|<1}} \lambda_{t,j_t}^{k_t-m_t}\binom{k_t}{m_t}.$$

Suppose S_1 in not an empty sum since otherwise $S_1 = 0$. Then there exists $t \in J_F$ and $j_t \in [1, n_t]$ such that $|\lambda_{t,j_t}| < 1$. Let ρ be the maximum among such values, that is,

$$\rho = \max\{\, |\lambda_{t,j}| \;:\; t \in J_F,\ j \in [1, n_t] \text{ and } |\lambda_{t,j}| < 1 \,\}.$$

Notice that every summand in S_1 has at least one $|\lambda_{t,j_t}| \le \rho < 1$ with $t \in J_F$, and $|\lambda_{i,j}| \le 1$ for all other $\lambda_{i,j}$. Also, $\binom{k_t}{m_t} \le k_t^{m_t} \le k_t^n$ since $m_t \le n$. So every summand in S_1 can be estimated by the expression

$$C_1 \cdot \prod_{\substack{t\in J_F\\ |\lambda_{t,j_t}|<1}} \rho^{k_t} k_t^n, \qquad \text{where } C_1 \text{ is a computable constant.}$$

We have $\rho^k k^n \to 0$ when $k \to \infty$, and for any rational $\delta > 0$ we can compute C such that $\rho^k k^n < \delta$ for $k \ge C$. If in addition we assume that $0 < \delta < 1$ and that $k_t \ge C$ for all $t \in J_F$, then $|S_1| \le C_1 n^{2\ell}\delta$.

Now, S_0 gives a finite number of limit values for $\langle u|\, A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell}\, |v\rangle$. If λ is not equal to any of them, then choose a rational $0 < \delta < 1$ such that $\epsilon = C_1 n^{2\ell}\delta$ is less than half the minimal distance between λ and those limit values. Using this δ, we compute C as before. By definition of C, if all $k_t \ge C$ for $t \in J_F$, then the distance between $\langle u|\, A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell}\, |v\rangle$ and one of the limit values of S_0 is less than ϵ. Thus $\langle u|\, A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell}\, |v\rangle$ cannot be equal to λ when all $k_t \ge C$ for $t \in J_F$. Hence if $\lambda = \langle u|\, A_1^{k_1} A_2^{k_2} \cdots A_\ell^{k_\ell}\, |v\rangle$, then there is $t \in J_F$ such that $k_t < C$.

5 Other decidability results and NP-hardness

In this section we utilise Theorem 1 to obtain some related decidability results. The first of these combines Theorem 1 with a seminal result of Rabin and allows us to use our decidability result for cutpoint isolation to solve the emptiness problem for PFA on letter-bounded context-free languages when the cutpoint is isolated. We again highlight here that the emptiness problem is undecidable in general on letter-bounded languages, even when all matrices commute and the PFA is polynomially ambiguous [4].

▶ **Corollary 9.** *The emptiness problem is decidable for probabilistic finite automata on letter-bounded context-free languages when the cutpoint is isolated.*

Proof. A seminal result of Rabin [23] showed that given a n-state PFA $\mathcal{P}$ acting on an alphabet Σ and *isolated* cutpoint $\lambda \in [0, 1]$ such that λ is isolated by $\epsilon > 0$ (i.e. $|\mathcal{P}(w) - \lambda| > \epsilon$ for all $w \in \Sigma^*$), then there exists a DFA $\mathcal{D}$ such that $L_{<\lambda}(\mathcal{P}) = L(\mathcal{D})$, where $L(\mathcal{D})$ denotes the language accepted by the DFA $\mathcal{D}$. Moreover, Rabin showed that the number of states of D is no more than $\left(1 + \frac{|F|}{\epsilon}\right)^{n-1}$ where F is the set of final states of $\mathcal{P}$.

We note that the proof of Theorem 1 not only determines if a cutpoint is isolated but also determines an "isolation bound" $\epsilon > 0$ if it is isolated. In this case we can use Rabin's result to construct an equivalent DFA $\mathcal{D}_<$ recognising $L_{<\lambda}(\mathcal{P})$. By inverting final and non final states of $\mathcal{D}_<$, we can construct $\mathcal{D}_\geq$ which recognises $L_{\geq\lambda}(\mathcal{P})$. Finally we note that if λ is isolated then $L_{<\lambda}(\mathcal{P}) = L_{\leq\lambda}(\mathcal{P})$ and thus $\mathcal{D}_<$ and $\mathcal{D}_\geq$ recognise the same languages as $L_{\leq\lambda}(\mathcal{P})$ and $L_{>\lambda}(\mathcal{P})$, respectively. Hence the emptiness problem is decidable. ◀

We now show that the value-1 problem for PFA on letter-bounded CFL inputs is decidable. This problem is undecidable for standard PFA but decidable for #-cyclic automata [13].

▶ **Corollary 10.** *The value-1 is decidable for probabilistic finite automata on letter-bounded context-free languages.*

Proof. This is trivial since the value-1 problem is equivalent to the isolation of the cutpoint 1 for a PFA [13]. ◀

Finally we note that Theorem 1 trivially allows us to determine if a given cutpoint is isolated for a PFA which is allowed a fixed maximum number of alternations between input letters (in any order), where each letter may be taken to an arbitrarily high power.

▶ **Corollary 11.** *Given a probabilistic finite automaton $\mathcal{P}$ on alphabet $\Sigma = \{a_1, \ldots, a_\ell\}$, cutpoint $\lambda \in [0, 1]$ and maximum number $k > 0$ of alternations between input letters, then determining if the cutpoint is isolated is decidable.*

Proof. We may apply Algorithm 1 on $\mathcal{P}$ and λ with each language from the following (finite) set of letter-bounded languages $\Lambda = \{w_1^* w_2^* \ldots w_k^* | w_i \in \Sigma\}$.

This defines the set of inputs where we alternate between the input letters a maximum of k times (analogous to how counter automata models are often studied with a maximum number of alternations between increasing and decreasing the counters). If any $L \in \Lambda$ on Algorithm 1 returns that the cutpoint is not isolated then λ is not isolated for $\mathcal{P}$ with a maximum number of alternations k, otherwise the cutpoint is isolated. ◀

In the remainder of this section, we give a lower bound on the complexity of the cutpoint isolation problem for 3-state PFA on letter-bounded inputs (noting that the encoded PFA are polynomially rather than exponentially ambiguous).

▶ **Theorem 12.** *Cutpoint isolation is NP-hard for 3-state PFA on letter-bounded inputs.*

Proof. We use a reduction from the subset sum problem, defined thus: given a set of positive integers $S = \{x_1, x_2, \ldots, x_k\} \subseteq \mathbb{N}$ and a natural number $T \in \mathbb{N}$, does there exist a subset $S' \subseteq S$ such that $\sum_{\ell \in S'} \ell = T$? This problem is well known to be NP-complete [12]. We define the set of matrices $M = \{A_i, B_i | 1 \leq i \leq k\} \subseteq \mathbb{Q}^{3\times 3}$ in the following way:

$$A_i = \frac{1}{x_i + 1} \begin{pmatrix} 1 & x_i & 0 \\ 0 & 1 & x_i \\ 0 & 0 & x_i + 1 \end{pmatrix}, \quad B_i = \frac{1}{x_i + 1} \begin{pmatrix} 1 & 0 & x_i \\ 0 & 1 & x_i \\ 0 & 0 & x_i + 1 \end{pmatrix}$$

Note that A_i and B_i are thus row stochastic. Let $u = (1, 0, 0)^\top$ be the initial probability distribution, $v = (0, 1, 0)^\top$ be the final state vector and let $\mathcal{P} = (\langle u| , \{A_i, B_i\}, |v\rangle)$ be our PFA. We define the cutpoint $\lambda = \frac{T}{y}$, where $y = \sum_{j=1}^{k}(x_j + 1)$. Define letter-bounded language $\mathcal{L} = (a_1|b_1)(a_2|b_2)\cdots(a_k|b_k) \subseteq a_1^* b_1^* a_2^* b_2^* \cdots a_k^* b_k^*$ (thus $\mathcal{L}$ is letter monotonic) and define a morphism $\varphi : \{a_i, b_i | 1 \leq i \leq k\}^* \to \{A_i, B_i | 1 \leq i \leq k\}^*$ in the natural way (e.g. the morphism induced by $\varphi(a_i) = A_i$ and $\varphi(b_i) = B_i$). Now, for a word $w = w_1 w_2 \cdots w_k \in \mathcal{L}$,

note that $w_j \in \{a_j, b_j\}$ for $1 \leq j \leq k$. Define that $\mathfrak{v}(a_i) = x_i$ and $\mathfrak{v}(b_i) = 0$ and inductively extend to $\mathfrak{v} : \Sigma^* \to \mathbb{N}$ by defining $\mathfrak{v}(w_1 w_2 \cdots w_k) = \mathfrak{v}(w_1) + \mathfrak{v}(w_2 \cdots w_k)$ with $\mathfrak{v}(\varepsilon) = 0$. In this case, we see that (due to the structure of A_i and B_i):

$$\langle u| \varphi(w_1 w_2 \cdots w_k) |v\rangle = \frac{\mathfrak{v}(w)}{\prod_{j=1}^{k}(x_j + 1)}$$

Note of course that the factor $\frac{1}{\prod_{j=1}^{k}(x_j+1)}$ is the same for any $w \in \mathcal{L}$.

Assume that there exists a solution to the subset sum problem, i.e., there exists $S' \subseteq S$ such that $\sum_{\ell \in S'} \ell = T$. Then consider word $w = w_1 w_2 \cdots w_k$ such that $w_j = a_j$ if $x_j \in S'$ and $w_j = b_j$ otherwise. In this case, $\sum_{i \in S'}^{k} x_i = \mathfrak{v}(w)$ and thus $\langle u| \varphi(w_1 w_2 \cdots w_k) |v\rangle = \frac{T}{y} = \lambda$. If no solution exists, then for any word $w = w_1 w_2 \cdots w_k$, $|\mathfrak{v}(w) - T| \geq 1$, and so $|\langle u| \varphi(w_1 w_2 \cdots w_k) |v\rangle - \lambda| > \frac{1}{\prod_{j=1}^{k}(x_j+1)}$ and thus λ cannot be arbitrarily approximated.

Clearly the representation size of the PFA $\mathcal{P}$ and λ are polynomial in the representation size of the subset sum problem instance and therefore we are done. ◀

6 Conclusion

In this work we showed that the cutpoint isolation problem is decidable for PFA when the input words are constrained to come from a letter-bounded context-free language, even for exponentially ambiguous PFA. This is in contrast to the situation for the (strict) emptiness problem and the injectivity problem, which are *undecidable* even over more restricted PFA for which all matrices commute, the PFA is polynomially ambiguous and the input words are over a simple letter-bounded language $a_1^* \cdots a_\ell^*$. We show that if the cutpoint is isolated for words over the input language, then the emptiness problem becomes decidable. We also show that the value-1 problem is decidable for these restricted input words.

It would be interesting to determine the complexity of the cutpoint isolation problem more precisely. We show an NP-hard lower bound in Theorem 12. The algorithm we provide may belong to NP, however there are some issues with showing this upper bound, namely that the number of limits of a stochastic matrix may be exponential in its dimension and the value of constant C from Proposition 7 may be exponential in terms of the bit size of the matrices, making the verification stage of an NP algorithm difficult to achieve. Extending the results to more general bounded languages would also be an interesting future work.

References

1 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in real algebraic geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin, second edition, 2006.

2 P. C. Bell, S. Chen, and L. M. Jackson. Freeness properties of weighted and probabilistic automata over bounded languages. *Information and Computation*, 269, 2019.

3 P. C. Bell, V. Halava, and M. Hirvensalo. Decision problems for probabilistic finite automata on bounded languages. *Fundamenta Informaticae*, 123(1):1–14, 2012.

4 Paul C. Bell. Polynomially ambiguous probabilistic automata on restricted languages. In *International Colloquium on Automata, Languages, and Programming (ICALP'19)*, volume 132 of *LIPIcs*, pages 105:1–105:14, 2019. doi:10.4230/LIPIcs.ICALP.2019.105.

5 A. Bertoni, G. Mauri, and M. Torelli. Some recursively unsolvable problems relating to isolated cutpoints in probabilistic automata. In *Automata, Languages and Programming*, volume 52, pages 87–94, 1977.

6 V. Blondel and V. Canterini. Undecidable problems for probabilistic automata of fixed dimension. *Theory of Computing Systems*, 36:231–245, 2003.

7 Jin-yi Cai. Computing Jordan normal forms exactly for commuting matrices in polynomial time. *Int. J. Found. Comput. Sci.*, 5(3/4):293–302, 1994. doi:10.1142/S0129054194000165.

8 Henri Cohen. *A course in computational algebraic number theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 1993. doi:10.1007/978-3-662-02945-9.

9 L. Daviaud, M. Jurdzinski, R. Lazic, F. Mazowiecki, G. A. Pérez, and J. Worrell. When is containment decidable for probabilistic automata? In *International Colloquium on Automata, Languages, and Programming (ICALP'18)*, volume 107 of *LIPIcs*, pages 121:1–121:14, 2018. doi:10.4230/LIPIcs.ICALP.2018.121.

10 N. Fijalkow, C. Riveros, and J. Worrell. Probabilistic automata of bounded ambiguity. In *28th International Conference on Concurrency Theory (CONCUR)*, pages 19:1–19:14, 2017.

11 S. Friedland. *Matrices: Algebra, Analysis and Applications*. World Scientific Publishing Company Pte Limited, 2015. URL: https://books.google.co.uk/books?id=y8fACwAAQBAJ.

12 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co. New York, NY, USA, 1979.

13 H. Gimbert and Y. Oualhadj. Probabilistic automata on finite words: decidable and undecidable problems. In *International Colloquium on Automata, Languages and Programming (ICALP'10)*, volume 2, pages 527–538, 2010.

14 S. Ginsburg. *The Mathematical Theory of Context Free Languages*. McGraw-Hill, 1966.

15 Seymour Ginsburg and Edwin Spanier. Semigroups, Presburger formulas, and languages. *Pacific journal of Mathematics*, 16(2):285–296, 1966.

16 O. Goldreich. On promise problems: a survey. In *Essays in Memory of Shimon Even*, volume 3895 of *Lecture Notes in Computer Science*, pages 254–290. Springer-Verlag, 2006.

17 V. Halava, T. Harju, M. Hirvensalo, and J. Karhumäki. Skolem's problem — on the border between decidability and undecidability. In *TUCS Technical Report Number 683*, 2005.

18 M. Hirvensalo. Improved undecidability results on the emptiness problem of probabilistic and quantum cut-point languages. *SOFSEM 2007: Theory and Practice of Computer Science, Lecture Notes in Computer Science*, 4362:309–319, 2007.

19 M. Mignotte. Some useful bounds. In *Computer algebra*, pages 259–263. Springer, Vienna, 1983.

20 V. Y. Pan. Optimal and nearly optimal algorithms for approximating polynomial zeros. *Comput. Math. Appl.*, 31(12):97–138, 1996. doi:10.1016/0898-1221(96)00080-6.

21 R. J. Parikh. On context-free languages. *Journal of the ACM (JACM)*, 13(4):570–581, 1966.

22 A. Paz. *Introduction to Probabilistic Automata*. Academic Press, 1971.

23 M. O. Rabin. Probabilistic automata. *Information and Control*, 6:230–245, 1963.

24 J. C. Rosales and P. A. García-Sánchez. *Numerical semigroups*, volume 20 of *Developments in Mathematics*. Springer, New York, 2009. doi:10.1007/978-1-4419-0160-6.

25 P. Turakainen. Generalized automata and stochastic languages. *Proceedings of the American Mathematical Society*, 21:303–309, 1969.

26 A. Weber and H. Seidl. On the degree of ambiguity of finite automata. *Theoretical Computer Science*, 88(2):325–349, 1991.

Multi-Dimensional Long-Run Average Problems for Vector Addition Systems with States

Krishnendu Chatterjee
IST Austria, Klosterneuburg, Austria
https://pub.ist.ac.at/~kchatterjee/
krish.chat@ist.ac.at

Thomas A. Henzinger
IST Austria, Klosterneuburg, Austria
http://pub.ist.ac.at/~tah/
tah@ist.ac.at

Jan Otop
University of Wrocław, Poland
https://sites.google.com/a/cs.uni.wroc.pl/jan-otop/
jotop@cs.uni.wroc.pl

Abstract

A vector addition system with states (VASS) consists of a finite set of states and counters. A transition changes the current state to the next state, and every counter is either incremented, or decremented, or left unchanged. A state and value for each counter is a configuration; and a computation is an infinite sequence of configurations with transitions between successive configurations. A probabilistic VASS consists of a VASS along with a probability distribution over the transitions for each state. Qualitative properties such as state and configuration reachability have been widely studied for VASS. In this work we consider multi-dimensional long-run average objectives for VASS and probabilistic VASS. For a counter, the cost of a configuration is the value of the counter; and the long-run average value of a computation for the counter is the long-run average of the costs of the configurations in the computation. The multi-dimensional long-run average problem given a VASS and a threshold value for each counter, asks whether there is a computation such that for each counter the long-run average value for the counter does not exceed the respective threshold. For probabilistic VASS, instead of the existence of a computation, we consider whether the expected long-run average value for each counter does not exceed the respective threshold. Our main results are as follows: we show that the multi-dimensional long-run average problem (a) is NP-complete for integer-valued VASS; (b) is undecidable for natural-valued VASS (i.e., nonnegative counters); and (c) can be solved in polynomial time for probabilistic integer-valued VASS, and probabilistic natural-valued VASS when all computations are non-terminating.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Quantitative automata

Keywords and phrases vector addition systems, mean-payoff, multidimension, probabilistic semantics

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.23

Related Version A full version of the paper is available at http://arxiv.org/abs/2007.08917.

Funding *Krishnendu Chatterjee*: The Austrian Science Fund (FWF) NFN grant S11407-N23 (RiSE/SHiNE).
Thomas A. Henzinger: The Austrian Science Fund (FWF) grants S11402-N23 (RiSE/ShiNE) and Z211-N23 (Wittgenstein Award).
Jan Otop: The National Science Centre (NCN), Poland under grant 2017/27/B/ST6/00299.

Acknowledgements We want to thank the anonymous reviewers for their helpful comments, which contributed to the final version of this paper.

1 Introduction

Vector Addition System with States (VASS) and probabilistic VASS. *Vector Addition Systems (VASs)* provide a powerful framework for analysis of parallel processes [16]. They are equivalent to the well-studied model of Petri Nets [25]. The generalization of VASs with a finite-state transition system gives *Vector Addition Systems with States (VASS)*. The model of VASS is as follows: there is a finite set of control states with transitions between them, and a set of k counters, where at every transition between the control states each counter is either incremented, decremented, or remains unchanged. For a VASS, a *configuration* is a control state and a valuation of each counter, and the transitions of the VASS determines the transitions between the configurations. Thus a VASS is a finite description of an infinite-state transition system between the configurations. The class of VASS where the counters can hold all possible integer values, are referred to as integer-valued VASS; and the class of VASS where the counters can hold only non-negative values, are referred to as natural-valued VASS. A probabilistic VASS consists of a VASS along with probability distribution over the transitions for every state.

VASS Framework in Verification. VASS are an elegant mathematical framework for concurrent processes [16], and have been widely studied in performance analysis of concurrent processes [14, 20, 23, 24]. They have also been used in several other contexts, such as: (a) analysis of parametrized systems [3], (b) abstract models for programs for bounds analysis [34], (c) interactions between components of an API in component-based synthesis [18]. The probabilistic VASS provide a natural model for problems mentioned above with stochasticity in the system [6]. Thus VASS and probabilistic VASS provide a rich framework for many problems in verification and program analysis.

Previous results for VASS. A computation (run) in a VASS is an infinite sequence of configurations with transitions between successive configurations. The classical problems studied for VASS are as follows: (a) *control-state reachability* where given a set of target control states a computation satisfies the objective if a target state is reached; (b) *configuration reachability* where given a set of target configurations a computation satisfies the objective if a target configuration reached. For natural-valued VASS, (a) the control-state reachability problem is ExpSpace-complete: the ExpSpace-hardness is shown in [15, 30] and the upper bound follows from [33]; and (b) the configuration reachability problem is decidable [26, 27, 28, 31], and a recent breakthrough result establishes non-elementary hardness [13]. For integer-valued VASS, (a) the control-state reachability problem is NLogSpace-complete (by reduction to graph reachability); (b) the configuration reachability problem is NP-complete. In probabilistic VASS, for the natural-valued class, even defining the probability measure over infinite computations is a challenging and complex problem [6], as computations that violate the non-negativity condition terminate as finite computations.

Long-run average objective and multi-dimensional long-run average problem. The classical problems for VASS consider qualitative (or Boolean) objectives where each computation is either satisfactory or not. In this work we consider multi-dimensional long-run average objective. For a counter, we consider the cost of a configuration as the value of the counter. For a computation, the long-run average of the costs of the configurations of the computation is the long-run average value for the respective counter. The multi-dimensional long-run average problem given a VASS and a threshold value for each counter, asks whether there is

a computation such that for each counter the long-run average value for the counter does not exceed the respective threshold. For integer-valued probabilistic VASS, instead of the existence of a computation, we consider whether the expected long-run average value for each counter does not exceed the respective threshold. For natural-valued probabilistic VASS, the presence of terminating runs makes even defining the probability measure complex. We consider two variants: (a) *strict semantics* that require all computations to be non-terminating; (b) *relaxed semantics* where we consider the conditional probability with respect to non-terminating runs.

Motivating examples. We present some motivating examples for the problems we consider. First, consider a VASS where the counters represent different queue lengths, and each queue consumes a resource type (e.g., energy or memory or time delay) proportional to its length. The multi-dimensional long-run average problems asks that the average consumption of each resource does not exceed a desired threshold. Second, consider a system that uses two different batteries, and the counters represent the charge levels. At different states, different batteries are used, and we are interested in the long-run average charge of each battery. This is again modeled as the multi-dimensional long-run average problem.

Our contributions. Our main contributions are as follows:

1. For non-probabilistic VASS we show that the multi-dimensional long-run average problem (a) is NP-complete for integer-valued VASS, and (b) is undecidable for natural-valued VASS.
2. For probabilistic integer-valued VASS, we show that the multi-dimensional long-run average problem can be solved in polynomial time. For natural-valued VASS, we show that the multi-dimensional problem can be solved in polynomial-time for (a) the strict semantics, and (b) the relaxed semantics for strongly connected VASS such that the expected multi-dimensional long-run average is finite. For the relaxed semantics and general natural-valued VASS, we show ExpSpace-hardness, and the exact decidability and complexity remain open.

Related works. For probabilistic VASS the long-run average behavior problem has been studied [6], as well as for other infinite-state models such as pushdown automata and games [1, 11, 12]. However, these works consider that costs are associated with the transitions of the finite-state system and do not depend on the counter values; moreover, they do not consider the multi-dimensional problem. In contrast, we consider costs that depend on the counter values, and hence on the configurations. Costs based on configurations, specifically the content of the stack in pushdown automata, have been considered in [32]. Quantitative asymptotic bounds for polynomial-time termination in VASS have also been studied [5, 29], however, these works do not consider long-run average property. Finally, a related model of automata with monitor counters with long-run average property have been considered in [7, 8]. However, there is a crucial difference: in automata with monitor counters, counters are reset once the value is used. Moreover, the complexity results for automata with monitor counters are quite different from the results we establish. Finally a recent work considers long-run average problem for VASS [9]. However, the cost is always single-dimensional with a linear combination of the counter values, and moreover, probabilistic VASS have not been considered in [9].

2 Preliminaries

For a sequence w, we define $w[i]$ as the $(i+1)$-th element of w (we start with 0) and $w[i,j]$ as the subsequence $w[i]w[i+1]\dots w[j]$. We allow j to be ∞ for infinite sequences. For a finite sequence w, we denote by $|w|$ its length; and for an infinite sequence the length is ∞. We use the same notation for vectors. For a vector $\vec{x} \in \mathbb{R}^k$ (resp., $\mathbb{Q}^k$, $\mathbb{Z}^k$ or $\mathbb{N}^k$), we define $x[i]$ as the i-th component of $\vec{x}$.

2.1 Vector addition systems with states (VASS)

A k-dimensional vector addition system with states (VASS) over $\mathbb{Z}$ (resp., over $\mathbb{N}$), referred to as VASS$(\mathbb{Z},k)$ (resp., VASS$(\mathbb{N},k)$), is a tuple $\mathcal{A} = \langle Q, Q_0, \delta\rangle$, where (1) Q is a finite set of states, (2) $Q_0 \subseteq Q$ is a set of initial states, and (3) $\delta \subseteq Q \times Q \times \mathbb{Z}^k$ is a transition relation. In a transition $(q,q',\vec{y})$, the vector $\vec{y}$ is called a *counter update* as we refer to k dimensions of a VASS as *counters*. We often omit the dimension in VASS and write VASS$(\mathbb{Z})$, VASS$(\mathbb{N})$ if a definition or an argument is uniform w.r.t. the dimension.

We define the size of a VASS in a standard way assuming binary encoding of counter updates. Formally, the size of a VASS $\langle Q, Q_0, \delta\rangle$ is defined as $|Q| + \sum_{(q,q',\vec{y})\in\delta} \mathrm{len}(\vec{y})$, where $\mathrm{len}(\vec{y})$ is the length of the binary representation of $\vec{y}$.

Configurations and computations. A *configuration* of a VASS$(\mathbb{Z},k)$ $\mathcal{A}$ is a pair from $Q \times \mathbb{Z}^k$, which consists of a state and a valuation of the counters. A *computation* of $\mathcal{A}$ is an infinite sequence π of configurations such that (a) $\pi[0] \in Q_0 \times \{\vec{0}\}$, and (b) for every $i \geq 0$, there exists $(q,q',\vec{y}) \in \delta$ such that $\pi[i] = (q,\vec{x})$ and $\pi[i+1] = (q', \vec{x}+\vec{y})$. Note that, without loss of generality, we assume that the initial counter valuation is $\vec{0}$. We can encode any initial configuration in the VASS itself.

A computation of a VASS$(\mathbb{N},k)$ $\mathcal{A}$ is a computation π of $\mathcal{A}$ considered as a VASS$(\mathbb{Z},k)$ such that the values of all counters are non-negative, i.e., for all i we have $\pi[i] \in Q \times \mathbb{N}^k$. Transitions of a VASS$(\mathbb{N})$ that make the value of some counter negative are disabled.

We call a finite sequence ρ a *subcomputation* of a VASS$(\mathbb{Z},k)$ (resp., VASS$(\mathbb{N},k)$) $\mathcal{A}$, if it satisfies condition (b), i.e., all configurations are consistent with some transitions of $\mathcal{A}$, and all configurations belong to $Q \times \mathbb{Z}^k$ (resp., $Q \times \mathbb{N}^k$).

Paths and cycles. A path $\mathbf{p} = (q_0,q_0',\vec{y}_0),(q_1,q_1',\vec{y}_1),\dots$ in a VASS$(\mathbb{Z})$ (resp., VASS$(\mathbb{N})$) $\mathcal{A}$ is a (finite or infinite) sequence of transitions (from δ) such that for all $0 \leq i < |\mathbf{p}|$ we have $q_i' = q_{i+1}$. A finite path $\mathbf{p}$ is a cycle if $\mathbf{p} = (q_0,q_0',\vec{y}_0),\dots,(q_m,q_m',\vec{y}_m)$ and $q_0 = q_m'$. Every computation in a VASS$(\mathbb{Z})$ (resp., VASS$(\mathbb{N})$) corresponds to the unique infinite path. Conversely, every infinite path in a VASS$(\mathbb{Z})$ $\mathcal{A}$ starting with $q_0 \in Q_0$ defines a computation in $\mathcal{A}$. However, if $\mathcal{A}$ is a VASS$(\mathbb{N},k)$, some paths do not correspond to valid computations due to non-negativity restriction posed on the counters.

Cycle characteristics. For a path $\mathbf{p}$ we define $\textsc{Gain}(\mathbf{p})$ as the vector of total counter change upon $\mathbf{p}$. Formally, for $\mathbf{p}$ of length n with counter updates $\vec{y}_1,\dots,\vec{y}_n$ we define $\textsc{Gain}(\mathbf{p}) = \sum_{i=1}^{n} \vec{y}_i$.

2.2 Probabilistic semantics

Markov chains. A *Markov chain* is a tuple $\langle \Sigma, Q, Q_0, \delta, P, \mu \rangle$ such that (1) Σ is a (finite) set of labels, (2) Q is a (finite) set of states, (3) Q_0 is a set of initial states, (4) $\delta \subseteq Q \times Q \times \Sigma$ is a transition relation, (5) $P \colon \delta \to (0, 1]$ is a probability distribution over transitions such that for every $s \in S$ we have $\sum_{s' \in S, a \in \Sigma} p(s, s', a) = 1$, and (6) $\mu \colon Q_0 \to [0, 1]$ is an initial distribution such that $\sum_{q \in Q_0} \mu(q) = 1$.

Probability measures defined by Markov chains. For a finite path $\mathbf{p}$ in a Markov chain $\mathcal{M}$, we define the probability of $\mathbf{p}$, denoted by $\mathbb{P}_{\mathcal{M}}(\mathbf{p})$, as the product of probabilities of transitions along $\mathbf{p}$. For any $n > 0$, the probability $\mathbb{P}_{\mathcal{M}}(\cdot)$ is indeed a probability measure over paths of length n. We extend this probability measure to infinite paths in the standard fashion. Let X be the set of all infinite paths in $\mathcal{M}$. For a basic open set $\mathbf{p} \cdot X$, which is the set of all paths with the common prefix $\mathbf{p}$, we define $\mathbb{P}_{\mathcal{M}}(\mathbf{p} \cdot X) = \mathbb{P}_{\mathcal{M}}(\mathbf{p})$, and then the probability measure over infinite paths defined by $\mathcal{M}$ is the unique extension of the above measure (by Carathéodory's extension theorem [17]). We will denote the unique probability measure defined by $\mathcal{M}$ as $\mathbb{P}_{\mathcal{M}}$.

Probabilistic VASS. Probabilistic VASS generalize both VASS and Markov chains. A probabilistic VASS is a VASS, in which transitions are labeled with probabilities. It can be also considered to be an infinite-state Markov chain over the set of states $Q \times \mathbb{Z}^k$ (resp., $Q \times \mathbb{N}^k$) and Σ is a singleton. Formally, a probabilistic VASS is a tuple $\mathcal{A} = \langle Q, Q_0, \delta, P, \mu \rangle$ such that (1) $\langle Q, Q_0, \delta \rangle$ is a VASS (VASS($\mathbb{Z}$) or VASS($\mathbb{N}$)), (2) $P \colon \delta \to (0, 1]$ is the probability distribution over transitions, which for every $q \in Q$ satisfies $\sum_{(q, q', \vec{y}) \in \delta} P(q, q', \vec{y}) = 1$, and (3) $\mu \colon Q_0 \to [0, 1]$ is the initial distribution, which satisfies $\sum_{q \in Q_0} \mu(q) = 1$.

Probability measures defined by probabilistic VASS. A probabilistic VASS($\mathbb{Z}$) (resp. VASS($\mathbb{N}$)) defines the probability measure over its computations. First, a probabilistic VASS($\mathbb{Z}$) (resp., VASS($\mathbb{N}$)) $\mathcal{A}$ defines the probability measure over its infinite paths in the same way as a Markov chain does. In VASS($\mathbb{Z}$), every path corresponds to a computation and hence the probability measure over infinite paths carries over to computations.

- We define $\mathbb{P}_{\mathcal{A}}$ as the probability measure on computations carried over from infinite paths.

However, in VASS($\mathbb{N}$) some paths may not correspond to valid computations. For that reason, defining the probability measure over computations poses difficulties [6]. We consider two possible solutions: the *strict* and the *relaxed* semantics.

- Under the strict semantics, we require that all paths correspond to valid computations and then we define the probability measure $\mathbb{P}^s_{\mathcal{A}}$ over computations as in the VASS($\mathbb{Z}$) case.
- Under the relaxed semantics, we require the set of paths corresponding to valid computations to have a non-zero probability, and we define the probability measure $\mathbb{P}^r_{\mathcal{A}}$ over computations as the conditional probability under the condition being the set of all paths that correspond to valid computations.

Random computations. To indicate that we consider a computation picked at random, we denote by ξ computations considered as random events.

▶ Remark 1. Under the strict semantics we require that every path corresponds to a valid computation, i.e., no counter gets a negative value. Note that relaxing *all* to *almost all* (i.e., with probability 1) gives us the same notion. Being a valid computation is a safety property and hence if the set of paths corresponding to valid computations has probability 1, then it is the set of all paths.

3 Problems

In this section, we define *the multi-dimensional average problem* and *the expected multi-dimensional average problem*, which we study in this paper. We define the averages over selected positions; averages are parametrized by a set of states S, called *selected states*, which determines meaningful configurations over which we compute the average, while skipping other configurations. This allows us to specify properties based on desired events (from S) rather than steps.

Averages and limit-averages over selecting states. Let $\mathcal{A} = \langle Q, Q_0, \delta \rangle$ be a VASS$(\mathbb{Z}, k)$ (resp., VASS$(\mathbb{N}, k)$) and $S \subseteq Q$ be a set of *selecting states*. Fix a counter $i \in \{1, \ldots, k\}$. For a finite subcomputation ρ of $\mathcal{A}$, which contains at least one configuration from $S \times \mathbb{Z}^k$ (resp., $S \times \mathbb{N}^k$), we define the *average value of counter i (over S)*, denoted by $\textsc{Avg}^i_S(\rho)$, as the average over values of counter i over configurations with the state belonging to S, i.e., we first pick a subsequence $(s_1, \vec{x}_1), \ldots, (s_m, \vec{x}_m)$ consisting of all configurations $(s, \vec{x})$ such that $s \in S$, and then take the average of the values of counter i: $\textsc{Avg}^i_S(\rho) = \frac{1}{m} \sum_{j=1}^{m} \vec{x}_j[i]$. If ρ has no configurations with states from S, then $\textsc{Avg}^i_S(\rho)$ is undefined. For an infinite sequence π of configurations, which contains infinitely many configurations from $S \times \mathbb{Z}^k$ (resp., $S \times \mathbb{N}^k$), we define the *limit-average value of the counter i (over S)*, denoted by $\textsc{LimAvg}^i_S(\pi)$, as $\textsc{LimAvg}^i_S(\pi) = \liminf_{k \to \infty} \textsc{Avg}^i_S(\pi[0, k-1])$. If π does not contain infinitely many configurations from $S \times \mathbb{Z}^k$ (resp., $S \times \mathbb{N}^k$), then $\textsc{LimAvg}^i_S(\pi)$ is undefined.

Multi-dimensional averages and limit-averages over selecting states. We extend averages and limit-averages to multiple dimensions. Let $\vec{S} = (S[1], \ldots, S[k])$ be a k-tuple of the subsets of Q. For a (finite) subcomputation ρ and an (infinite) computation π, we define

$$\textsc{Avg}_{\vec{S}}(\rho) = (\textsc{Avg}^1_{S[1]}(\rho), \ldots, \textsc{Avg}^k_{S[k]}(\rho))$$
$$\textsc{LimAvg}_{\vec{S}}(\pi) = (\textsc{LimAvg}^1_{S[1]}(\pi), \ldots, \textsc{LimAvg}^k_{S[k]}(\pi))$$

if all their components are defined. If any component of $\textsc{Avg}_{\vec{S}}(\rho)$ (resp., $\textsc{LimAvg}_{\vec{S}}(\pi)$) is undefined, the whole vector is undefined.

▶ **Definition 2** (The multi-dimensional average problem for VASS). *Given a VASS$(\mathbb{N}, k)$ (resp., VASS$(\mathbb{Z}, k)$) $\mathcal{A}$, $\vec{S} \in (2^Q)^k$ and $\vec{\lambda} \in \mathbb{Q}^k$, the **(multi-dimensional) average** problem asks whether there exists a computation π such that $\textsc{LimAvg}_{\vec{S}}(\pi)$ is defined and $\textsc{LimAvg}_{\vec{S}}(\pi) \leq \vec{\lambda}$, i.e., the limit-averages of counter values over $\vec{S}$ are component-wise bounded by $\vec{\lambda}$.*

Expected limit-averages over selecting states. Consider a probabilistic VASS$(\mathbb{Z}, k)$ (resp., VASS$(\mathbb{N}, k)$), which defines a probability measure $\mathbb{P}_{\mathcal{A}}$ (resp., $\mathbb{P}^s_{\mathcal{A}}$ or $\mathbb{P}^r_{\mathcal{A}}$) over its computations. Let $\vec{S} = (S[1], \ldots, S[k])$ be a k-tuple of the subsets of Q. The function $\xi \mapsto \textsc{LimAvg}^i_{S[i]}(\xi)$ is a random variable w.r.t. $\mathbb{P}_{\mathcal{A}}$ (resp., $\mathbb{P}^s_{\mathcal{A}}$ or $\mathbb{P}^r_{\mathcal{A}}$) and we define $\mathbb{E}_{\mathcal{A}}(\textsc{LimAvg}^i_{S[i]})$ as the expected value of this random variable. If the set of computations ξ, at which $\textsc{LimAvg}^i_{S[i]}(\xi)$ is undefined, has a non-zero probability, then the expected value is undefined as well. We extend the expectation to vectors and define the expected multi-dimensional limit-average as

$$\mathbb{E}_{\mathcal{A}}(\textsc{LimAvg}_{\vec{S}}) = (\mathbb{E}_{\mathcal{A}}(\textsc{LimAvg}^1_{S[1]}), \ldots, \mathbb{E}_{\mathcal{A}}(\textsc{LimAvg}^k_{S[k]})).$$

As above, the expected value $\mathbb{E}_{\mathcal{A}}(\textsc{LimAvg}_{\vec{S}})$ is defined only if all components are defined.

▶ **Definition 3** (The expected (multi-dimensional) average problem for VASS). *Given a probabilistic VASS $\mathcal{A}$, $\vec{S} \in (2^Q)^k$, the* ***expected multi-dimensional average*** *problem asks to compute the expected limit-averages over $\vec{S}$, i.e., $\mathbb{E}_{\mathcal{A}}(\textsc{LimAvg}_{\vec{S}})$.*

▶ Remark 4. In all complexity results for the multi-dimensional average and the expected multi-dimensional average problems, we consider VASS where the counter updates are encoded in binary.

4 Results on integer-valued VASS

4.1 The multi-dimension average problem

Consider a VASS$(\mathbb{Z}, k)$ $\mathcal{A} = \langle Q, Q_0, \delta\rangle$, a vector $\vec{S} \in (2^Q)^k$ and thresholds $\vec{\lambda} \in \mathbb{Q}^k$. For simplicity, we assume that $Q_0 = \{q_0\}$ and hence $(q_0, \vec{0})$ is the initial configuration.

We present sufficient and necessary conditions for the existence of a computation π with $\textsc{LimAvg}_{\vec{S}}(\pi) \leq \vec{\lambda}$. These conditions are expressed in terms of simple cycles in $\mathcal{A}$, i.e., they stipulate that for each counter i there exist (a) a simple cycle that can be *iterated* to ensure that limit average infimum is consistent with the threshold $\vec{\lambda}[i]$, and (b) a path to *access* this cycle, and then to switch back to another cycle. These conditions can be check in NP. We present main ideas assuming that $\mathcal{A}$ is strongly connected.

Assume that $\mathcal{A}$ is strongly connected, i.e., it is strongly connected as a labeled graph. We distinguish two types of counters based on their behavior in $\mathcal{A}$: *bounded* and *unbounded*. We first assume that for every counter i there is a cycle $\mathbf{c}_i$ such that iterating this cycle decreases this counter's value, i.e., $\textsc{Gain}(\mathbf{c}_i)[i] < 0$. In such a case all counters are *unbounded* and for any $\vec{\lambda}$ there exists a computation π such that $\textsc{LimAvg}_{\vec{S}}(\pi) \leq \vec{\lambda}$.

The all-unbounded case. We assume that all counters are unbounded. Fix some $\vec{\lambda} \in \mathbb{Q}^k$. We construct π such that $\textsc{LimAvg}_{\vec{S}}(\pi) \leq \vec{\lambda}$ by interleaving strategies for each counter i to make its partial average below $\vec{\lambda}[i]$. More precisely, we define the path $\mathbf{p}$ of the form

$$\mathbf{p} = \mathbf{s}_1^1 \mathbf{s}_2^1 \dots \mathbf{s}_k^1 \mathbf{s}_1^2 \dots \mathbf{s}_k^2 \dots$$

such that for every prefix $\mathbf{s}_1^1 \dots \mathbf{s}_i^j$ of $\mathbf{p}$, the subcomputation ρ_i^j corresponding to that prefix satisfies $\textsc{Avg}_{S[i]}(\rho_i^j) \leq \vec{\lambda}[i]$, i.e., the partial average over $S[i]$ is bounded by $\vec{\lambda}[i]$. We can construct such $\mathbf{p}$ as follows. Suppose that a prefix of $\mathbf{p}$ has been defined as above, and we need to construct $\mathbf{s}_i^j$. There are two cases: if the cycle $\mathbf{c}_i$ with $\textsc{Gain}(\mathbf{c}_i)[i] < 0$ contains a selecting state from $S[i]$, then $\mathbf{s}_i^j = (\mathbf{c}_i)^m$ for some large m, i.e., we iterate $\mathbf{c}_i$ long enough such that the average of the whole prefix computation is below $\vec{\lambda}[i]$. If $\mathbf{c}_i$ does not contain any state from $S[i]$, then there exists $\mathbf{c}_i'$ that contains a selecting state and $\textsc{Gain}(\mathbf{c}_i')[i] < 0$. Indeed, let $\mathbf{d}$ be a cycle from the initial state of $\mathbf{c}_i$ to itself that contains a selecting state. Then, $\mathbf{d} \cdot \mathbf{c}_i^N$ contains a selecting state and $\textsc{Gain}(\mathbf{d} \cdot \mathbf{c}_i^N)[i] < 0$ for some N.

Now, let π be the computation corresponding to $\mathbf{p}$. For every counter i there are infinitely many positions at which the partial average over $S[i]$ at most $\vec{\lambda}[i]$ and hence $\textsc{LimAvg}_{\vec{S}}(\pi) \leq \vec{\lambda}$.

The some-bounded case. Assume that for a counter j, there is no cycle such that iterating it decreases the value of counter j. In other words, for all cycles c we have $\textsc{Gain}(c)[j] \geq 0$. We classify such a counter as *bounded*. It is clearly lower bounded and for the limit average to be finite its has to be upper bounded. In consequence, in any computation π with finite limit-average, all cycles c that occur infinitely often satisfy $\textsc{Gain}(c)[j] = 0$. This in turn restricts the set of cycles that can appear infinitely often in the considered paths, which

makes other counters *bounded*. We iterate this process until we reach a fixed point $\mathbf{B}$, which is the set of all bounded counters. The complement of $\mathbf{B}$, denoted by $\mathbf{U}$, is the set of *unbounded* counters.

Note that for each unbounded counter $i \in \mathbf{U}$ there is a cycle $\mathbf{c}_i$ such that:

(U1) we have $\textsc{Gain}(\mathbf{c}_i)[i] < 0$ and it contains a selecting state, and

(U2) for each bounded counter $j \in \mathbf{B}$, we have $\textsc{Gain}(\mathbf{c}_i)[j] = 0$.

It follows that similarly to the *all-unbounded* case, we can make sure that the partial averages of unbounded counters are arbitrarily low.

The limit average of a bounded counter depends on its initial value. Indeed, in the extreme case, if the value of a counter i does not change in any transition, then it is bounded and in every computation the limit average of counter i is precisely its initial value. However, to characterize cycles that witness low limit-averages of bounded counters it is more convenient to refer to a configuration that occurs infinitely often rather than the initial configuration. Therefore, we consider a *recurring configuration* $(s_0, \vec{x})$ that is: (a) reachable from the initial configuration $(q_0, \vec{0})$, (b) there are infinitely many configurations $(s_0, \vec{y})$ such that $\vec{y}$ and $\vec{x}$ agree on bounded counters. We now drop the strongly-connected assumption on $\mathcal{A}$.

Observe that switching between cycles for different (bounded or unbounded) counters may affect values of bounded counters. Therefore, for a bounded counter $i \in \mathbf{B}$ we require that there is a cycle $\mathbf{c}_i$, which (a) can be *accessed* with an appropriate path, and (b) its average together with the initial value are bounded by $\vec{\lambda}[i]$. To make it more precise: there exist a cycle $\mathbf{c}_i$ and paths $\mathbf{in}_i, \mathbf{out}_i$ such that

(B1) we have $\textsc{Avg}_{S[i]}(\rho) \leq \vec{\lambda}[i]$, where ρ is the subcomputation corresponding to the cycle $\mathbf{c}_i$ starting from the configuration reached from $(s_0, \vec{x})$ over the path $\mathbf{in}_i$, and

(B2) $\mathbf{in}_i, \mathbf{out}_i$ are from s_0 to some $s \in \mathbf{c}_i$ and from the same s to s_0 respectively, and for each bounded counter $j \in \mathbf{B}$, we have $\textsc{Gain}(\mathbf{c}_i)[j] = 0$ and $\textsc{Gain}(\mathbf{in}_i\mathbf{out}_i)[j] = 0$.

Finally, we require that for all unbounded counters $i \in U$ there exist access paths $\mathbf{in}_i, \mathbf{out}_i$ as in condition **(B2)**, i.e., paths $\mathbf{in}_i, \mathbf{out}_i$ satisfy:

(U3) $\mathbf{in}_i, \mathbf{out}_i$ are from s_0 to some $s \in \mathbf{c}_i$ and from the same s to s_0 respectively, and for each bounded counter $j \in B$, we have $\textsc{Gain}(\mathbf{in}_i\mathbf{out}_i)[j] = 0$.

Condition **(U3)** is necessary as otherwise, switching between cycles for unbounded cycles and bounded cycles could change values of bounded counters. Observe that conditions **(U2)** and **(U3)** together are the same as **(B2)**. We unify these conditions into a single one denoted **(BU)**.

A witness for $\mathbf{LimAvg}_{\vec{S}} \leq \vec{\lambda}$. A *witness for* $\textsc{LimAvg}_{\vec{S}} \leq \vec{\lambda}$ is a tuple consisting of (a) a (recurring) configuration $(s_0, \vec{x})$ reachable from the initial configuration $(q_0, \vec{0})$, (b) a partition of counters into $\mathbf{B}$ and $\mathbf{U}$, and (c) cycles $\mathbf{c}_i$ and access paths $\mathbf{in}_i, \mathbf{out}_i$, for all i, which all satisfy conditions **(U1)**, **(B1)** and **(BU)**.

First, we show that the existence of a witness for $\textsc{LimAvg}_{\vec{S}} \leq \vec{\lambda}$ is sufficient for the existence of a computation π with $\textsc{LimAvg}_{\vec{S}}(\pi) \leq \vec{\lambda}$.

Key ideas. Using a witness, we construct a computation π satisfying $\textsc{LimAvg}_{\vec{S}}(\pi) \leq \vec{\lambda}$ in a similar way as in the *all-unbounded* case. The only difference here is that we use access paths to switch between cycles for different counters so that we switch between the counters in the state s_0, where the values of bounded counters are the same as in the configuration $(s_0, \vec{x})$. Due to condition **(BU)**, we do not require $\mathcal{A}$ to be strongly connected. The full proof is relegated to the appendix. In consequence, we have the following:

▶ **Lemma 5.** *Let $\mathcal{A}$ be a VASS($\mathbb{Z}$). If it has a witness for $\textsc{LimAvg}_{\vec{S}} \leq \vec{\lambda}$, then there exists a computation π such that $\textsc{LimAvg}_{\vec{S}}(\pi) \leq \vec{\lambda}$.*

We show that the existence of a witness for $\textsc{LimAvg}_{\vec{S}} \leq \vec{\lambda}$ is necessary for the existence of a computation π with $\textsc{LimAvg}_{\vec{S}}(\pi) \leq \vec{\lambda}$. We present key ideas, while the full proof is relegated to the appendix.

Key ideas. Consider a computation π such that $\textsc{LimAvg}_{\vec{S}}(\pi) \leq \vec{\lambda}$ and let $\mathbf{p}$ be the infinite path corresponding to π. Let s_0 be a state that occurs infinitely often. We can decompose $\mathbf{p}$ into simple cycles greedily always picking the first occurring simple cycle. Now, consider all simple cycles $D = \{\mathbf{d}_1, \ldots, \mathbf{d}_m\}$ that occur infinitely often. Based on these cycles we define $\mathbf{B}$ as the set of counters j such that for all cycles $\mathbf{d} \in D$ we have $\textsc{Gain}(\mathbf{d})[j] = 0$, and $\mathbf{U} = \{1, \ldots, k\} \setminus \mathbf{B}$. It follows that there is a configuration $(s_0, \vec{x})$ in π such that for all $(s_0, \vec{y})$ past that configuration $\vec{x}$ and $\vec{y}$ agree on the values of bounded counters. This follows from the fact that eventually all cycles from s_0 to itself in $\mathbf{p}$ can be decomposed into cycles from D, which have zero gain for all bounded counters. We pick $(s_0, \vec{x})$ as the recurrent configuration in the witness.

Observe that for the limit-average to be finite, for every $i \in \mathbf{U}$ there has to be a simple cycle $\mathbf{c}_i$ with $\textsc{Gain}(\mathbf{c}_i)[i] < 0$. We can also enforce that $\mathbf{c}_i$ contains a selecting state; such a cycle with a selecting state is of the form $\mathbf{f}\mathbf{c}_i^N$, where $\mathbf{f}, \mathbf{c}_i$ are simple cycles, but $N = \textsc{Gain}(\mathbf{f})[i]$ can be exponential. Still, this cycle can be represented polynomially by the pair $(\mathbf{f}, \mathbf{c}_i)$. Therefore, condition **(U1)** is satisfied. For counter $i \in \mathbf{B}$ there is a subsequence on which the limit average of i converges to a value less or equal to $\vec{\lambda}[i]$. It follows that there has to be a cycle from D satisfying **(B1)**. Otherwise, the limit averages of all computations exceed $\vec{\lambda}[i]$.

Each cycle $\mathbf{d}$ from D is contained in some cycle $\tilde{\mathbf{d}}$ from s_0 to itself, i.e., $\tilde{\mathbf{d}} = \mathbf{in} \cdot \mathbf{d} \cdot \mathbf{out}$. We can pick such $\tilde{\mathbf{d}}$ that occurs infinitely often and hence can be decomposed into cycles $\mathbf{e}_1, \ldots, \mathbf{e}_l$ from D, which by the definition of $\mathbf{B}$, satisfy $\textsc{Gain}(\mathbf{e}_p)[j] = 0$ for each $j \in \mathbf{B}$. Therefore, paths $\mathbf{in}_i, \mathbf{out}_i$ for $\mathbf{c}_i$ satisfy condition **(BU)**.

Finally, we need to show that if there is a witness, then there is a witness of polynomial size. Cycles $\mathbf{c}_i$ (for $i \in \mathbf{B}$) as well as paths $\mathbf{in}_i, \mathbf{out}_i$ (for all i) can be picked to have polynomial length and for $i \in \mathbf{U}$ cycles $\mathbf{c}_i$ can be picked to have polynomial length representation. Now, for the recurrent configuration $(s_0, \vec{x})$, observe that the components of $\vec{x}$ are relevant only for bounded counters, whose average is bounded by α, which has the binary representation of polynomial length in $|\mathcal{A}|$. Therefore, we need to find any reachable $(s_0, \vec{x})$ such that for all bounded i we have $\vec{x}[i]$ is less than $-\alpha + \vec{\lambda}[i]$. Furthermore, reachable configurations in VASS($\mathbb{Z}$) are semilinear sets [4] defined by equations of polynomial size in $|\mathcal{A}|$ and hence if there is any configuration $(s_0, \vec{x})$ then there is also one with the binary representation of polynomial length in $|\mathcal{A}| + |\vec{\lambda}|$. In consequence, we have:

▶ **Lemma 6.** *For all VASS($\mathbb{Z}, k$) $\mathcal{A}$ and $\vec{\lambda} \in \mathbb{Q}^k$ the following holds: if there is a computation π such that $\textsc{LimAvg}_{\vec{S}}(\pi) \leq \vec{\lambda}$, then there exists a witness for $\textsc{LimAvg}_{\vec{S}} \leq \vec{\lambda}$, which has a polynomial size in $|\mathcal{A}| + |\vec{\lambda}|$.*

Finally, a polynomial-size witness for $\textsc{LimAvg}_{\vec{S}}(\pi) \leq \vec{\lambda}$ can be non-deterministically picked and verified in polynomial time. More precisely, in the definition of a witness for $\textsc{LimAvg}_{\vec{S}} \leq \vec{\lambda}$ condition (a) can be checked in NP as reachability for VASS($\mathbb{Z}$) is NP-complete [4], and conditions (b) and (c) can be check in polynomial time. In consequence, we have:

▶ **Lemma 7.** *The multi-dimensional average problem for VASS($\mathbb{Z}$) is in NP.*

For hardness of the multi-dimensional average problem, consider configuration-reachability for VASS($\mathbb{Z}$), which is NP-complete. Configuration-reachability is mutually reducible to coverability for VASS($\mathbb{Z}$) [21], which in turn is equivalent to dual coverability, i.e, the problem, given a VASS($\mathbb{Z}$) and two configurations $(s, \vec{0})$ and $(t, \vec{x})$, decide whether there is a (finite) subcomputation from $(s, \vec{0})$ to some $(t, \vec{y})$, where $\vec{y} \leq \vec{x}$. The dual coverability straightforwardly reduces to the multi-dimensional average problem as follows. We construct $\mathcal{A}'$ from $\mathcal{A}$ by adding a fresh state t^* and two transitions labeled with $\vec{0}$: from t to t^* and a self-loop over t^*. Observe that there is a subcomputation from $(s, \vec{0})$ to $(t, \vec{y})$ where $\vec{y} \leq \vec{x}$ in $\mathcal{A}$ if and only if there is a computation from $(s, \vec{0})$ that eventually reaches t^* and the multi-dimensional limit-averages are bounded by $\vec{x}$. To enforce that a computation eventually reaches t^*, we use an additional counter that is 0 in the configurations of $\mathcal{A}$ and it changes to -1 upon moving to t^*. Requirement that the limit average of this counter is less or equal to -1 forces the computation to move to t^*. In consequence, the dual coverability for VASS($\mathbb{Z}$) reduces to the multi-dimensional average problem for VASS($\mathbb{Z}$) and hence the latter problem is NP-complete.

▶ **Theorem 8.** *The multi-dimensional average problem for VASS($\mathbb{Z}$) is NP-complete.*

4.2 The expected average problem

Observe that the expected average problem for probabilistic VASS($\mathbb{Z}$) is modular and each dimension can be considered separately. This follows from the fact that each path in a VASS($\mathbb{Z}$) corresponds to a computation, which is not the case for VASS($\mathbb{N}$). Furthermore, in this problem we compute the expected value over all computations and hence it can be considered for each dimension separately. Therefore, we consider VASS that are single-dimensional. We first discuss the strongly-connected case and then generalize our results to all VASS($\mathbb{Z}$).

4.2.1 The strongly-connected case

Let $\mathcal{A}$ be a single-dimensional probabilistic VASS($\mathbb{Z}, 1$), which is strongly connected as a labeled graph. We additionally assume that it has a single initial configuration $(q_0, 0)$. The case of any initial distribution follows easily. First, we define the expected gain of $\mathcal{A}$, which corresponds to the expected trend of the counter.

The expected gain. The graph of $\mathcal{A}$ can be considered as a Markov chain and using standard methods we compute for each state q its long-run frequency x_q [2, 19]. More precisely, the *frequency* of q in a subcomputation $\xi[1, n]$ is the number of configurations with the state q in $\xi[1, n]$ divided by n. The Ergodic Theorem for Markov chains implies that with probability 1 over a random computation ξ, for every state q, the frequency of q in $\xi[1, n]$ converges to x_q as n tends to infinity. Based on frequencies x_q we define *the expected gain* $\mathbb{E}(Gain)$ as the expected counter update provided that the initial state q is picked at random according to the frequencies x_q and the outgoing transition is picked at random according to the distribution at q, that is:

$$\mathbb{E}(Gain) = \sum_{(q,q',y)\in\delta} x_q \cdot P(q, q', y) \cdot y$$

The classification based on $\mathbb{E}(Gain)$. We show that if $\mathbb{E}(Gain)$ is positive (resp., negative), then the limit-average is infinite (resp. minus infinity). However, if the expected gain is zero there are two cases based on boundedness of configurations. Either the gain of every cycle is actually zero, or cycles with a positive gain balance cycles with a negative gain so that the expected gain is zero. We discuss these cases below.

We say that a VASS$(\mathbb{Z},1)$ is *totally bounded* if the gain of each cycle is zero. This property does not depend on the probability distribution over transitions and we extend it straightforwardly to probabilistic VASS$(\mathbb{Z},1)$. Note that if a VASS$(\mathbb{Z},1)$ is strongly connected and totally bounded, then in each reachable configuration, the state uniquely determines the counter's value. Otherwise, there exists a cycle with a non-zero gain. This observation allows us to reduce the expected limit-average problem for such VASS to computing the expected long-run reward for Markov chains [2, Chapter 10.5].

Consider a VASS$(\mathbb{Z},1)$ with $\mathbb{E}(Gain)$ being zero and at least one cycle with a non-zero gain. Observe that $\mathbb{E}(Gain)$ being 0 implies that there is at least on cycle with a positive gain and a cycle with a negative gain. Let us consider the simplest probabilistic VASS, which has a single state q_0 and two self loops labeled with 1 and -1, both with probability 0.5. The distribution of the counter's gain in n transitions, denoted S_n, is related to the binomial distribution $B(n,0.5)$ in the following way: $S_n \sim 2 \cdot B(n,0.5) - n$. It follows that with probability 1 over a random computation ξ, the counter in ξ is neither lower nor upper bounded. Furthermore, we show that with probability 1, a random computation ξ has two subsequences such that the averages on one sequence tend to ∞, and on the other tend to $-\infty$. To state this formally we define:

$$\textsc{LimAvgInf}_S(\pi) = \liminf_{k\to\infty} \textsc{Avg}_S(\rho[1,k])$$
$$\textsc{LimAvgSup}_S(\pi) = \limsup_{k\to\infty} \textsc{Avg}_S(\rho[1,k])$$

Now, we present the lemma summarizing the above discussion.

▶ **Lemma 9.** *Let $\mathcal{A}$ be a strongly-connected probabilistic VASS$(\mathbb{Z},1)$. One of the following conditions holds:*

(1) $\mathbb{E}(Gain) > 0$, *and* $\textsc{LimAvgInf}_S(\xi) = \textsc{LimAvgSup}_S(\xi) = \infty$ *with probability* 1 *(over ξ),*
(2) $\mathbb{E}(Gain) < 0$, *and* $\textsc{LimAvgInf}_S(\xi) = \textsc{LimAvgSup}_S(\xi) = -\infty$ *with probability* 1,
(3) *$\mathcal{A}$ is totally bounded, and for some $x \in \mathbb{Q}$, with probability* 1 *over ξ we have* $\textsc{LimAvgInf}_S(\xi) = \textsc{LimAvgSup}_S(\xi) = x$, *and*
(4) $\mathbb{E}(Gain) = 0$, *$\mathcal{A}$ is not totally bounded, and with probability* 1 *over ξ we have* $\textsc{LimAvgInf}_S(\xi) = -\infty$ *and* $\textsc{LimAvgSup}_S(\xi) = \infty$.

Proof's sketch. To show conditions (1) and (2) observe that the Ergodic Theorem for Markov chains implies that with probability 1 (over ξ) the value of the counter in configuration $\xi[n]$ divided by n converges to $\mathbb{E}(Gain)$ as n tends to infinity and hence the average counter value converges to ∞ if $\mathbb{E}(Gain) > 0$ and it converges to $-\infty$ if $\mathbb{E}(Gain) < 0$.

Condition (3) follows from the above discussion and the fact that in a strongly-connected Markov chain there is a value x such that the long-run reward is almost surely x.

We present the full proof of conditions (1), (2) and (3) in the appendix. Now, we focus on condition (4). We only show that $\textsc{LimAvgInf}_S(\xi) = -\infty$ holds with probability 1 over ξ, as the proof of $\textsc{LimAvgSup}_S(\xi) = \infty$ is symmetric. Observe that in a strongly-connected VASS$(\mathbb{Z})$, the event $\textsc{LimAvgInf}_S(\xi) = -\infty$ is a tail event. Therefore, due to Kolmogorov's 0-1 law [17] it has either probability 0 or 1. In consequence, it suffices to show that it has a positive probability.

First, we show that with a positive probability $\textsc{LimAvgInf}_S(\xi)$ is upper bounded.

▶ **Lemma 10.** *Consider a probabilistic VASS($\mathbb{Z}$) $\mathcal{A}$ as in (4) of Lemma 9. There exist $c \in \mathbb{Q}$ and $\delta > 0$ such that $\textsc{LimAvgInf}_S(\xi) < c$ holds with probability greater than δ.*

Proof's ideas. We observe that $\textsc{LimAvgInf}_S(\xi) = \infty$ implies that the values of the counter converge to infinity, which implies $\mathbb{E}(Gain) > 0$. The full proof is relegated to the appendix. ◀

For $c \in \mathbb{Q}$, we define X_c as the set of computations π such that $\textsc{LimAvgInf}_S(\pi) < c$. Lemma 10 states that there are $c \in \mathbb{Q}$ and $\delta > 0$ such that $\mathbb{P}(X_c) = \delta$. We show that for every $d \in \mathbb{Q}$, $\textsc{LimAvgInf}_S(\xi) < d$ holds with probability at least δ.

▶ **Lemma 11.** *Consider a probabilistic VASS($\mathbb{Z}$) $\mathcal{A}$ as in (4) of Lemma 9. Assume that $\mathbb{P}(X_c) = \delta > 0$. Then, for every d we have $\mathbb{P}(\textsc{LimAvgInf}_S(\xi) < d) \geq \delta$.*

Proof's ideas. The main idea is to prepend to computations from X_c a subcomputation that decreases the initial counter's value to a. Then, the limit infimum of averages is $c + a$. Furthermore, we show that the set of such finite paths has probability 1.

More precisely, consider a subcomputation ρ from $(q_0, 0)$ to (q_0, a) and $\pi \in X_c$. We define the *join* of ρ and π, denoted by $\rho \bowtie \pi$, as the computation consisting of first ρ and then $\pi[1, \infty]$ (π with the first configuration removed) with a added to the counter of all following configurations of π. Observe that the join of ρ and π is indeed a computation. Moreover, the influence of the average of ρ on the whole computation diminishes and hence $\textsc{LimAvgInf}_S(\rho \bowtie \pi) = \textsc{LimAvgInf}_S(\pi) + a < c + a$.

Let Y be the set of (finite) subcomputations that start in $(q_0, 0)$ and terminate once they reach some configuration (q_0, b) where $b < d - c$. Since $\mathcal{A}$ is strongly connected and not totally bounded, almost surely a random computation ξ reaches a configuration (q_0, b) where $b < d - c$. It follows that the set of all computations extending some subcomputation from Y has probability 1. Therefore, the set of all joins of subcomputations from Y with computations from X_c has probability at least δ and all such computations $\rho \bowtie \pi$ satisfy $\textsc{LimAvgInf}_S(\rho \bowtie \pi) < d$, and hence Lemma 11 follows. ◀

Lemma 11 implies that the set of computations ξ such that $\textsc{LimAvgInf}_S(\xi) = -\infty$ has a positive probability. Since $\textsc{LimAvgInf}_S(\xi) = -\infty$ is a tail event in a strongly-connected VASS($\mathbb{Z}$), Kolmogorov's 0-1 law [17] implies that its probability is 1, which concludes the proof of Lemma 9. ◀

Lemma 9 implies the following:

▶ **Lemma 12.** *The expected average problem for strongly-connected probabilistic VASS($\mathbb{Z}, 1$) can be solved in polynomial time.*

Proof's ideas. Consider a strongly-connected probabilistic VASS($\mathbb{Z}, 1$) $\mathcal{A}$. We can compute frequencies x_q of states of $\mathcal{A}$ in polynomial time using standard methods [2, Chapter 10.5]. Having frequencies x_q, we can compute the expected gain $\mathbb{E}(Gain)$ of $\mathcal{A}$ in polynomial time from the definition.

Assume that $\mathbb{E}(Gain) = 0$. We can check whether $\mathcal{A}$ is not totally bounded by checking whether it has a cycle with a non-zero gain, which can be done in polynomial time. Finally, if it is totally bounded, then each state of $\mathcal{A}$ uniquely determines the value of each counter, and we can eliminate the counters and label states with counter values. Therefore, the problem of computing the long-run average of almost all computations, denoted by x, reduces to computing the expected long-run reward a Markov chain with rewards, which can be done in polynomial time [2, Chapter 10.5]. In consequence, we can check all the conditions of Lemma 9 in polynomial time and hence the result follows. ◀

4.2.2 The general case

Let $\mathcal{A}$ be a probabilistic VASS$(\mathbb{Z}, 1)$. We show how to compute its expected limit-average in polynomial time. We identify all *bottom* SCCs (BSCCs) of $\mathcal{A}$ (where an SCC B is *bottom* if all states reachable from B belong to B). If there is a BSCC that does not contain a state from S, then the expected limit-average is undefined. Assume that every BSSC contains a state from S and consider the following cases:

- If there are two BSCCs: (a) one with a positive expected gain, and (b) the other with a negative expected gain, then the expected limit average is undefined. The expected value is undefined for random variables that attain $+\infty$ and $-\infty$ with a positive probability [17].
- If there is a BSCC with a positive gain and every BSCC has (a) a positive gain, or (b) it has the zero gain and it is totally bounded, then the expected limit-average is ∞.
- If there is a BSCC with (a) a negative gain, or (b) the zero gain and not totally bounded, and every BSCC has a non-positive gain, then the expected limit-average is $-\infty$.
- If all BSCCs have the zero gain and are totally bounded, the expected limit-average is finite and we discuss below how to compute it.

First, we compute all BSCCs $B_1, \ldots, B_m$ of $\mathcal{A}$. We pick in each of these components an initial state q_0^i. For each BSCC B_i with its initial configuration $(q_0^i, 0)$, we compute x_i, which is the expected limit-average in B_i. As we observed before, if we join a subcomputation from $(q_0, 0)$ to (q_0^i, y_i) and some computation from $(q_0^i, 0)$ with the limit-average x_i, then the limit-average of the resulting computation is $x_i + y_i$. Therefore, for each state q_0^i we compute the probability of reaching that state from the initial distribution, denoted p_i, and the expected counter's value y_i upon reaching q_0^i, i.e., the conditional expected counter's value under the condition that the state q_0^i is reached. Probabilities p_i can be computed using standard methods for Markov chains [2, Chapter 10.1]. The values y_i can be computed as well using standard methods for Markov chains with rewards [2, Chapter 10.5]. Observe that the expected limit-average of $\mathcal{A}$ is given by the following formula:

$$\mathbb{E}_{\mathcal{A}}(\textsc{LimAvg}_S) = \sum_{i=1}^{m} p_i \cdot (y_i + x_i)$$

Finally, as we discussed above, we can compute the expected value for each counter separately. In consequence we have the following:

▶ **Theorem 13.** *The expected average problem for probabilistic VASS$(\mathbb{Z})$ can be solved in polynomial time.*

5 Results on natural-valued VASS

5.1 The average problem in a single dimension

We first study the average problem for single-dimensional VASS$(\mathbb{N}, 1)$. For the lower bound observe that the reachability problem for VASS$(\mathbb{N}, 1)$, which is NP-complete [22], reduces to the average problem for VASS$(\mathbb{N}, 1)$. The reduction is straightforward and hence we omit it. To show the NP upper bound, we show the following:

▶ **Lemma 14.** *For all VASS$(\mathbb{N}, 1)$ $\mathcal{A}$ the following holds: there exists a computation π with $\textsc{LimAvg}_S(\pi) \leq \lambda$ if and only if there exist subcomputations ρ_0, ρ_c such that:*

- *ρ_0 is from $(q_0, 0)$ to (s, x), where $x \leq \lambda$, and*
- *ρ_c is a cycle from (s, x) to itself satisfying the following conditions:*

(a) $\mathrm{AVG}_S(\rho_c) \leq \lambda$,
(b) *the number of configurations with selecting states in* ρ_c*, i.e., configurations from* $S \times \mathbb{N}$*, is* $O(|S| \cdot |\lambda|^2)$*, and*
(c) *the value of the counter in each configuration of* ρ_c *from* $S \times \mathbb{N}$ *is* $O(|S| \cdot |\lambda|^2)$.

We present the proof's sketch, while the full proof is relegated to the appendix.

Proof's sketch. Observe that having ρ_0, ρ_c as above, the computation $\pi = \rho_0(\rho_c)^\infty$ is a valid computation and it satisfies $\mathrm{LIMAVG}_S(\pi) \leq \lambda$. Conversely, we observe that there has to be a cycle ρ in π with the average $\mathrm{AVG}_S(\rho) \leq \lambda$. Furthermore, we pick a cycle of minimal length and observe that it satisfies all the conditions of the lemma, i.e., if some condition is violated, then the cycle can be shortened. ◀

We can check in NP whether there exist ρ_0, ρ_c satisfying the conditions from Lemma 14.

Key ideas. We non-deterministically pick all selecting configurations $(s_1, x_1), \ldots, (s_m, x_m)$ from ρ_c. Then, we check reachability from $(q_0, 0)$ to (s_1, x_1), and for each $i < m$ reachability over non-selecting configurations from (s_i, x_i) to (s_{i+1}, x_{i+1}), and from (s_m, x_m) to (s_1, x_1). All these reachability checks can be done in NP. Finally, we check $\frac{1}{m}\sum_{i=1}^{m} x_i \leq \lambda$. All these checks can be done is NP. The number of configurations m as well as the size of each configuration is polynomially bounded due to Lemma 14. In consequence, we have:

▶ **Theorem 15.** *The average problem for VASS*$(\mathbb{N}, 1)$ *is NP-complete.*

5.2 The multi-dimension average problem

We show that the (decision variant of the) multi-dimensional average problem for VASS($\mathbb{N}$) is undecidable. A related problem, called the average-value problem, has been studied in [9]. In that problem, the values of all counters in each configuration $(q, \vec{x})$ are aggregated into a single number, called *the cost*, by computing dot-product of $\vec{x}$ and a cost vector $\vec{c}_q \in N^k$. A cost vector $\vec{c}_q$ depends on the state q in the configuration. The average-value problem asks whether there exists a computation such that the limit average of costs is less or equal to a threshold λ. The problem for VASS($\mathbb{N}$) with threshold 0 is undecidable [9, Theorem 24]. Threshold 0 in VASS($\mathbb{N}$) means that whenever a cost vector is non-zero at component i (i.e, $\vec{c}_q[i] \neq 0$), then counter's i value should be 0. This constraint is expressed in the multi-dimensional average problem and hence we have:

▶ **Theorem 16.** *The decision variant of the multi-dimensional average problem for VASS*$(\mathbb{N})$ *is undecidable.*

5.3 The expected multi-dimensional average problem

We first study probabilistic VASS($\mathbb{N}$) under the strict semantics and give the precise complexity. Next, we consider the relaxed semantics, where we have the exact complexity in the strongly-connected case and a hardness result in the general case.

5.3.1 Probabilistic natural-valued VASS under the strict semantics

Consider a probabilistic VASS($\mathbb{N}$) $\mathcal{A}$ under the strict semantics. To check whether every path of $\mathcal{A}$ corresponds to a valid computation, we examine each counter i separately and check whether it can reach a negative value from some initial configuration. This can be done in polynomial time with the standard reachability analysis. If a negative value for some

counter is reachable, then the expected limit-average is undefined under the strict semantics. Otherwise, every path in $\mathcal{A}$ corresponds to a valid computation and we can consider $\mathcal{A}$ as a VASS($\mathbb{Z}$) as the non-negativity restriction is vacuous for $\mathcal{A}$. Therefore, we apply Theorem 13 and compute the expected value for $\mathcal{A}$. In consequence, we have:

▶ **Theorem 17.** *The expected average problem for probabilistic VASS($\mathbb{N}$) under the strict semantics can be solved in polynomial time.*

5.3.2 Probabilistic natural-valued VASS under the relaxed semantics

The finite strongly-connected case. Consider a probabilistic VASS($\mathbb{N}$) $\mathcal{A}$, which is strongly connected. We show that if the expected limit-average is finite, then the strict and the relaxed semantics coincide. We first assume that $\mathcal{A}$ is single-dimensional. Using the classification from Lemma 9 applied to $\mathcal{A}$ considered as a VASS($\mathbb{Z}, 1$), we observe that:

- If $\mathbb{E}(Gain) < 0$ or $\mathbb{E}(Gain) = 0$ and $\mathcal{A}$ is not totally bounded, then a random computation ξ (under the VASS($\mathbb{Z}$) semantics) satisfies $\textsc{LimAvgInf}_S(\xi) = -\infty$, and hence it is not a valid computation of the VASS($\mathbb{N}$). Therefore, the expected limit-average under the relaxed semantics is undefined for $\mathcal{A}$.
- If $\mathbb{E}(Gain) > 0$, then $\textsc{LimAvgInf}_S(\xi) = \infty$. Therefore, if the set of random computations ξ (under the VASS($\mathbb{Z}$) semantics) that are also valid computations under the VASS($\mathbb{N}$) semantics has a positive probability, then the expected limit-average under the relaxed semantics is defined and infinite. Otherwise, it is undefined.
- If $\mathbb{E}(Gain) = 0$ and $\mathcal{A}$ is totally bounded, then (as we observe in Section 4.2) in each configuration, the state uniquely determines the counters value. Therefore, we can check whether counter values in all states are non-negative. If this is the case, then all paths correspond to valid computations, the expected limit-average is defined and finite, and we can compute it with Theorem 17. Otherwise, observe that in a strongly-connected $\mathcal{A}$ every state is visited with probability 1 and hence the expected value is undefined.

Therefore, for the expected limit-average under the relaxed semantics to be defined and finite, the expected gain w.r.t. every counter has to be 0 and it has to be totally bounded. Furthermore, we check for each counter independently whether every path corresponds to a valid computation in VASS($\mathbb{N}$). Since we consider all paths, we can make these checks independently for all counters. In consequence we have the following:

▶ **Theorem 18.** *Deciding whether the expected limit-average is defined and finite over strongly-connected probabilistic VASS($\mathbb{N}$) under the relaxed semantics can be solved in polynomial time. Furthermore, it if is it can be computed in polynomial time.*

The general case. We present only a hardness result. The coverability problem for VASS($\mathbb{N}$), which is ExpSpace-complete [30], reduces to (the decision version of) the expected limit-average problem for VASS($\mathbb{N}$). The reduction is rather straightforward with minor technical difficulties (we need to ensure that the expected value of each counter is finite). In consequence, we have:

▶ **Theorem 19.** *The problem, given a probabilistic VASS($\mathbb{N}, k$) $\mathcal{A}$ under the relaxed semantics, $S \subseteq Q$ and $\vec{x} \in \mathbb{Q}^k$, decide whether $\mathbb{E}_{\mathcal{A}}(\textsc{LimAvg}_{\vec{S}}) < \vec{x}$ is ExpSpace-hard.*

The proof has been relegated to the appendix.

▶ Remark 20. The decidability of the problem from Theorem 19 is open.

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Piotr Hofman, Richard Mayr, K. Narayan Kumar, and Patrick Totzke. Infinite-state energy games. In *CSL-LICS 2014*, pages 7:1–7:10, 2014. doi:10.1145/2603088.2603100.

2 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

3 Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. Decidability in parameterized verification. *SIGACT News*, 47(2):53–64, 2016. doi:10.1145/2951860.2951873.

4 Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in two-dimensional vector addition systems with states is pspace-complete. In *LICS 2015*, pages 32–43. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.14.

5 Tomás Brázdil, Krishnendu Chatterjee, Antonín Kucera, Petr Novotný, Dominik Velan, and Florian Zuleger. Efficient algorithms for asymptotic bounds on termination time in VASS. In *LICS 2018*, pages 185–194, 2018. doi:10.1145/3209108.3209191.

6 Tomás Brázdil, Stefan Kiefer, Antonín Kucera, and Petr Novotný. Long-run average behaviour of probabilistic vector addition systems. In *LICS 2015*, pages 44–55, 2015. doi:10.1109/LICS.2015.15.

7 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Nested weighted limit-average automata of bounded width. In *MFCS 2016*, pages 24:1–24:14, 2016. doi:10.4230/LIPIcs.MFCS.2016.24.

8 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Quantitative monitor automata. In *SAS 2016*, pages 23–38, 2016. doi:10.1007/978-3-662-53413-7_2.

9 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Long-run average behavior of vector addition systems with states. In *CONCUR 2019*, pages 27:1–27:16, 2019. doi:10.4230/LIPIcs.CONCUR.2019.27.

10 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Quantitative automata under probabilistic semantics. *Logical Methods in Computer Science*, 15(3), 2019. doi:10.23638/LMCS-15(3:16)2019.

11 Krishnendu Chatterjee and Yaron Velner. The complexity of mean-payoff pushdown games. *J. ACM*, 64(5):34:1–34:49, 2017. doi:10.1145/3121408.

12 Krishnendu Chatterjee and Yaron Velner. Hyperplane separation technique for multidimensional mean-payoff games. *J. Comput. Syst. Sci.*, 88:236–259, 2017. doi:10.1016/j.jcss.2017.04.005.

13 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for petri nets is not elementary. In *STOC 2019*, pages 24–33, 2019. doi:10.1145/3313276.3316369.

14 Emanuele D'Osualdo, Jonathan Kochems, and C.-H. Luke Ong. Automatic verification of erlang-style concurrency. In *SAS 2013*, pages 454–476, 2013. doi:10.1007/978-3-642-38856-9_24.

15 Javier Esparza. Decidability and complexity of petri net problems—an introduction. *Lectures on Petri nets I: Basic models*, pages 374–428, 1998.

16 Javier Esparza and Mogens Nielsen. Decidability issues for petri nets - a survey. *Bull. EATCS*, 52:244–262, 1994.

17 W. Feller. *An introduction to probability theory and its applications*. Wiley, 1971.

18 Yu Feng, Ruben Martins, Yuepeng Wang, Isil Dillig, and Thomas W. Reps. Component-based synthesis for complex apis. In *POPL 2017*, pages 599–612, New York, NY, USA, 2017. ACM. doi:10.1145/3009837.3009851.

19 Jerzy Filar and Koos Vrieze. *Competitive Markov decision processes*. Springer, 1996.

20 Pierre Ganty and Rupak Majumdar. Algorithmic verification of asynchronous programs. *ACM Trans. Program. Lang. Syst.*, 34(1):6:1–6:48, May 2012. doi:10.1145/2160910.2160915.

21 Christoph Haase and Simon Halfon. Integer vector addition systems with states. In *RP 2014*, pages 112–124, 2014. doi:10.1007/978-3-319-11439-2_9.

22 Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In *CONCUR 2009*, pages 369–383, 2009. doi:10.1007/978-3-642-04081-8_25.

23 Alexander Kaiser, Daniel Kroening, and Thomas Wahl. Dynamic cutoff detection in parameterized concurrent programs. In *CAV 2010*, pages 645–659, 2010. doi:10.1007/978-3-642-14295-6_55.

24 Alexander Kaiser, Daniel Kroening, and Thomas Wahl. Efficient coverability analysis by proof minimization. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR 2012*, pages 500–515, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-32940-1_35.

25 Richard M. Karp and Raymond E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969. doi:10.1016/S0022-0000(69)80011-5.

26 S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 267–281, 1982. doi:10.1145/800070.802201.

27 Jean-Luc Lambert. A structure to decide reachability in petri nets. *Theoretical Computer Science*, 99(1):79–104, 1992. doi:10.1016/0304-3975(92)90173-D.

28 Jérôme Leroux. Vector addition systems reachability problem (A simpler solution). In *Turing-100 - The Alan Turing Centenary, Manchester, UK, June 22-25, 2012*, pages 214–228, 2012. URL: https://easychair.org/publications/paper/Blr.

29 Jérôme Leroux. Polynomial vector addition systems with states. In *ICALP 2018*, pages 134:1–134:13, 2018. doi:10.4230/LIPIcs.ICALP.2018.134.

30 Richard Lipton. The reachability problem is exponential-space hard. *Department of Computer Science, Yale University, Tech. Rep*, 62, 1976.

31 Ernst W. Mayr. An Algorithm for the General Petri Net Reachability Problem. In *STOC 1981*, pages 238–246, 1981. doi:10.1145/800076.802477.

32 Jakub Michaliszyn and Jan Otop. Average stack cost of büchi pushdown automata. In *FSTTCS 2017*, pages 42:1–42:13, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.42.

33 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

34 Moritz Sinn, Florian Zuleger, and Helmut Veith. A simple and scalable static analysis for bound analysis and amortized complexity analysis. In *CAV 2014*, pages 745–761, 2014. doi:10.1007/978-3-319-08867-9_50.

A Appendix

▶ **Lemma 5.** *Let $\mathcal{A}$ be a VASS($\mathbb{Z}$). If it has a witness for $\textsc{LimAvg}_{\vec{S}} \leq \vec{\lambda}$, then there exists a computation π such that $\textsc{LimAvg}_{\vec{S}}(\pi) \leq \vec{\lambda}$.*

Proof. Let $(q_0, \vec{0})$ be the initial configuration and $(s_0, \vec{x})$ be the recurrent configuration of the witness. We assume that $\mathbf{B}$ is non-empty. Otherwise, the construction presented in the all-bounded case essentially works. The only difference is that we use paths $\mathbf{in}_i, \mathbf{out}_i$ to switch between cycles.

We define the path $\mathbf{p}$ of the form

$$\mathbf{p} = \mathbf{s}_0\mathbf{s}_1^1\mathbf{s}_2^1 \dots \mathbf{s}_k^1\mathbf{s}_1^2 \dots \mathbf{s}_k^2 \dots$$

such that for every prefix $\mathbf{s}_0\mathbf{s}_1^1 \dots \mathbf{s}_i^j$ of $\mathbf{p}$, the precomputation ρ_i^j corresponding to that prefix satisfies:

(a) $\textsc{Avg}_{S[i]}(\rho_i^j) \leq \vec{\lambda}[i] + \frac{1}{j}$, and

(b) ρ_i^j terminates in $(s_0, \vec{y})$, where for every $i \in \mathbf{B}$ we have $\vec{y}[i] = \vec{x}[i]$.

Having such a path $\mathbf{p}$, consider the computation π that corresponds to $\mathbf{p}$. Observe that for every counter i, condition (a) implies that for every $\epsilon > 0$ there are infinitely many positions k such that the average $\textsc{Avg}_{S[i]}(\pi[1,k])$ is less than $\vec{\lambda}[i] + \epsilon$ and hence $\textsc{LimAvg}_{S[i]}(\pi) \leq \vec{\lambda}[i]$. It follows that $\textsc{LimAvg}_{\vec{S}}(\pi) \leq \vec{\lambda}$.

Now, we discuss how to construct such $\mathbf{p}$. First, $\mathbf{s}_0$ is a path that corresponds to a computation from $(q_0, \vec{0})$ to $(s_0, \vec{x})$. Second, suppose that a prefix $\mathbf{p}_1$ of $\mathbf{p}$ has been defined as above, and we need to construct $\mathbf{s}_i^j$. Observe that the already constructed subcomputation ends in $(s_0, \vec{y})$ such that for every $k \in \mathbf{B}$ we have $\vec{y}[i] = \vec{x}[i]$.

There exist paths $\mathbf{in}_i, \mathbf{out}_i$ and a cycle $\mathbf{c}_i$ satisfying **(BU)**, and **(U1)** (if $i \in \mathbf{U}$) or **(B1)** (if $i \in \mathbf{B}$). Consider $\mathbf{s}_i^j$ of the form $\mathbf{in}_i \mathbf{c}_i^N \mathbf{out}_i$ from some $N > 0$ fixed later. First, due to condition **(BU)**, for every $k \in \mathbf{B}$ we have $\textsc{Gain}(\mathbf{in}_i \mathbf{out}_i) = 0$ and $\textsc{Gain}(\mathbf{c}_i) = 0$, and hence $\textsc{Gain}(\mathbf{s}_i^j) = 0$. It follows that **(b)** holds.

Now, to see that **(a)** holds for N big enough we consider two cases. If counter i is bounded, then **(B1)** implies that for ρ_N being the computation corresponding to $\mathbf{p}_1 \mathbf{in}_i \mathbf{c}_i^N \mathbf{out}_i$, the average of the part corresponding to $\mathbf{c}_i^N$ is x, which is less or equal to $\vec{\lambda}[i]$. Therefore, $\textsc{Avg}_{S[i]}(\rho_N)$ tends to x as $N \to \infty$ and hence there is N such that $\textsc{Avg}_{S[i]}(\rho_N) \leq \vec{\lambda}[i] + \frac{1}{j}$.

If counter i is unbounded, then $\textsc{Gain}(\mathbf{c}_i)[i] < 0$ and $\mathbf{c}_i$ contains a selecting state. It follows that the values of counter i tend to $-\infty$, and hence $\textsc{Avg}_{S[i]}(\rho_N)$ tends to $-\infty$ as $N \to \infty$. Therefore, there exists N such that $\textsc{Avg}_{S[i]}(\rho_N) \leq \vec{\lambda}[i] + \frac{1}{j}$. ◀

▶ **Lemma 6.** *For all VASS$(\mathbb{Z}, k)$ $\mathcal{A}$ and $\vec{\lambda} \in \mathbb{Q}^k$ the following holds: if there is a computation π such that $\textsc{LimAvg}_{\vec{S}}(\pi) \leq \vec{\lambda}$, then there exists a witness for $\textsc{LimAvg}_{\vec{S}} \leq \vec{\lambda}$, which has a polynomial size in $|\mathcal{A}| + |\vec{\lambda}|$.*

Proof. Consider a computation π such that $\textsc{LimAvg}_{\vec{S}}(\pi) \leq \vec{\lambda}$ and let $\mathbf{p}$ be the infinite path corresponding to π. We decompose $\mathbf{p}$ into simple cycles greedily always picking the first occurring simple cycle. Now, consider all simple cycles that occur infinitely often as well as all rotations of these cycles $D = \{\mathbf{d}_1, \ldots, \mathbf{d}_m\}$. Based on these cycles, we define $\mathbf{B}$ as the set of counters j such that for all cycles $\mathbf{d} \in D$ we have $\textsc{Gain}(\mathbf{d})[j] = 0$, and $\mathbf{U} = \{1, \ldots, k\} \setminus \mathbf{B}$.

Let s_0 be a state that occurs infinitely often in $\mathbf{p}$. Note that eventually, past some position K, all transitions belong to cycles from D. Therefore, we pick the first configuration $(s_0, \vec{x})$ past position K and observe that for all successive configurations $(s_0, \vec{y})$, for every counter $i \in \mathbf{B}$, the gain between these configurations is 0 and hence $\vec{x}[i] = \vec{y}[i]$. The length of description of $(s_0, \vec{x})$ is unbounded, but we show at the end of the proof that it can be chosen to be polynomial in $|\mathcal{A}| + |\vec{\lambda}|$. First, we show that there is any witness.

Consider a counter j such that all cycles $\mathbf{d} \in D$ satisfy $\textsc{Gain}(\mathbf{d})[j] \geq 0$. We observe that for all cycles $\mathbf{d} \in D$ we have $\textsc{Gain}(\mathbf{d})[j] = 0$ and hence $j \in \mathbf{B}$. Indeed, if there is a cycle $\mathbf{d} \in D$ that satisfies $\textsc{Gain}(\mathbf{d})[j] > 0$, then values of counter j in π tend to ∞ and hence $\textsc{LimAvg}_{S[j]}(\pi) = \infty > \vec{\lambda}[j]$. It follows that for ever $i \in \mathbf{U}$, there is a cycle $\mathbf{c}_i$ such that $\textsc{Gain}(\mathbf{c}_i)[i] < 0$.

The unbounded-counter case. Consider $i \in \mathbf{U}$. Let $\tilde{\mathbf{c}}_i \in D$ be such that $\textsc{Gain}(\tilde{\mathbf{c}}_i)[i] < 0$ and let q be the first state of $\tilde{\mathbf{c}}_i$. Observe that there exist cycles $\mathbf{f}_1, \mathbf{f}_2 \in D$ such that $\mathbf{f}_1$ is from s_0 to itself and contains q, and $\mathbf{f}_2$ is from q to itself and contains some selecting state from S_i. Indeed, q and s_0 occur infinite often in $\mathbf{p}$. Consider disjoint cycles $\mathbf{e}_1, \mathbf{e}_2, \ldots$ each from s_0 to itself that contains q. For each $\mathbf{e}_l$ we remove from it iteratively simple cycles from D such that the resulting $\mathbf{e}'_l$ does not contain any simple cycle from D. Observe that only finitely many $\mathbf{e}'_l$ are non-empty as otherwise there would be another simple cycle that occurs infinitely often and does not belong to D. Now, let $\mathbf{e}_l$ be a cycle that can be decomposed

into simple cycles from D. Let us remove iteratively simple cycles to leave the ends s_0 of $\mathbf{e}_l$ and a single occurrence of q. The resulting cycle consists of one or two simple cycles from D. The proof for $\mathbf{f}_2$ is similar.

Now, for $N = |\textsc{Gain}(\mathbf{f}_2)[i]| + 1$ we define $\mathbf{c}_i = \mathbf{f}_2\tilde{\mathbf{c}}_i^N$. Then, $\textsc{Gain}(\mathbf{c}_i^N)[i] < 0$ and $\mathbf{c}_i$ contains a selecting state from $S[i]$. Therefore, condition **(U1)** holds.

The cycle $\mathbf{f}_1$ can be decomposed into paths $\mathbf{in}_i, \mathbf{out}_i$, respectively from s_0 to q, and from q to s_0. Furthermore, since $\mathbf{f}_2, \mathbf{c}_i, \mathbf{in}_i\mathbf{out}_i$ can be decomposed into cycles from D, then by definition of $\mathbf{B}$, for all $j \in \mathbf{B}$ we have $\textsc{Gain}(\mathbf{f}_2)[j] = \textsc{Gain}(\mathbf{c}_i)[j] = \textsc{Gain}(\mathbf{in}_i\mathbf{out}_i)[j] = 0$ and hence $\textsc{Gain}(\mathbf{f}_2\mathbf{c}_i^n)[j] = 0$. Therefore, condition **(BU)** holds. Note that $\mathbf{in}_i, \mathbf{out}_i$ have the lengths bounded by $2 \cdot |\mathcal{A}|$ and $\mathbf{c}_i$ can be represented by the pair of cycles: $(\tilde{\mathbf{c}}_i, \mathbf{f}_2)$ of the length at most $2 \cdot |\mathcal{A}|$. Thus, all have polynomial-size representation.

The bounded-counter case. Let $i \in \mathbf{B}$. Since every cycle $\mathbf{d} \in D$ satisfies $\textsc{Gain}(\mathbf{d})[i] = 0$, from some position K onwards the gain of each cycle is 0 and hence the value of counter i on any two positions past K with the same state are the same. Therefore, we associate with each state the value of counter i and eliminate values of counters. Furthermore, we associate with each cycle $\mathbf{d} \in D$ its average value over $S[i]$, which is uniquely defined. Finally, if for all cycles $\mathbf{d} \in D$ the average exceeds $\vec{\lambda}[i]$, then $\textsc{LimAvg}_{S[i]}(\pi) > \vec{\lambda}[i]$. Therefore, there exists a cycle $\mathbf{c}_i \in D$ with the average value less or equal to $\vec{\lambda}[i]$.

Since s_0 occurs infinitely often, in particular it occurs past position K. Therefore, we show as in the unbounded-counter case that there exist $\mathbf{in}_i, \mathbf{out}_i$ such that $\mathbf{in}_i$ leads from s_0 to some state q of $\mathbf{c}_i$ and $\mathbf{out}_i$ from q to s_0 such that $\mathbf{c}_i, \mathbf{in}_i, \mathbf{out}_i$ satisfy **(BU)**. Finally, observe that $\mathbf{c}_i, \mathbf{in}_i, \mathbf{out}_i$ satisfy **(U1)**. Note that $\mathbf{c}_i, \mathbf{in}_i, \mathbf{out}_i$ can be picked to have the lengths at most $2 \cdot |\mathcal{A}|$.

A witness with polynomial recurrent configuration. We have shown that there exists a witness for $\textsc{LimAvg}_{\vec{S}} \leq \vec{\lambda}$ with $(s_0, \vec{x})$. We show that there exists $\vec{z}$ such that (a) the witness for $\textsc{LimAvg}_{\vec{S}} \leq \vec{\lambda}$ with $(s_0, \vec{x})$ replaced by $(s_0, \vec{z})$ remains a witness for $\textsc{LimAvg}_{\vec{S}} \leq \vec{\lambda}$, and (b) $\vec{z}$ has the binary representation of polynomial length in $|\mathcal{A}| + |\vec{\lambda}|$.

For (a) we need to show that (i) **(B1)** is satisfied with $(s_0, \vec{z})$, and (ii) $(s_0, \vec{z})$ is reachable from $(q_0, \vec{0})$. Recall that **(B1)** states that for every $i \in \mathbf{B}$, the subcomputation ρ corresponding to the cycle $\mathbf{c}_i$ starting from the configuration reached from $(s_0, \vec{z})$ over the path $\mathbf{in}_i$ satisfies $\textsc{Avg}_{S[i]}(\rho) \leq \vec{\lambda}[i]$. Note that the lengths of $\mathbf{in}_i, \mathbf{c}_i$ are bounded by $2 \cdot |\mathcal{A}|$ (because $i \in \mathbf{B}$) and hence there exists α_i with the binary representation of polynomial-length in $|\mathcal{A}|$ such that $\textsc{Avg}_{S[i]}(\rho) = \alpha_i + \vec{z}[i]$. Therefore, any $\vec{z}$ such that $\vec{z}[i] < -\alpha_i + \vec{\lambda}[i]$ for all $i \in \mathbf{B}$ satisfies (i). For (ii) observe that reachable configurations in VASS($\mathbb{Z}$) are semilinear sets [4] represented by polynomial-size equations (where coefficients are given in binary). Therefore, we can find a vector $\vec{z}$ satisfying (i) and (ii) whose binary representation has polynomial length in $|\mathcal{A}| + |\vec{\lambda}|$. ◀

▶ **Lemma 9.** *Let $\mathcal{A}$ be a strongly-connected probabilistic VASS($\mathbb{Z}, 1$). One of the following conditions holds:*

(1) $\mathbb{E}(\mathit{Gain}) > 0$, *and* $\textsc{LimAvgInf}_S(\xi) = \textsc{LimAvgSup}_S(\xi) = \infty$ *with probability* 1 *(over ξ),*

(2) $\mathbb{E}(\mathit{Gain}) < 0$, *and* $\textsc{LimAvgInf}_S(\xi) = \textsc{LimAvgSup}_S(\xi) = -\infty$ *with probability* 1,

(3) *$\mathcal{A}$ is totally bounded, and for some $x \in \mathbb{Q}$, with probability* 1 *over ξ we have $\textsc{LimAvgInf}_S(\xi) = \textsc{LimAvgSup}_S(\xi) = x$, and*

(4) $\mathbb{E}(\mathit{Gain}) = 0$, *$\mathcal{A}$ is not totally bounded, and with probability* 1 *over ξ we have $\textsc{LimAvgInf}_S(\xi) = -\infty$ and $\textsc{LimAvgSup}_S(\xi) = \infty$.*

Proof. We prove only (1),(2),(3), as (4) is presented with details in the paper.

We show conditions (1) and (2). Assume that $\mathbb{E}(Gain) \neq 0$. The Ergodic Theorem for Markov chains implies that with probability 1 (over ξ) for every state q, the frequency of configurations with the state q converges to $x[q]$. For every transition (q, q', y), the frequency of this transition converges to $x_q \cdot P(q, q', y)$. Now, we multiply the frequency of each transition (q, q', y) by its update value y and get the value of the counter in $\xi[n]$ divided by n. On the other hand, this value converges to $\mathbb{E}(Gain)$ as n tends to infinity. It follows that with probability 1 over ξ the counter's value at a position n equals $\mathbb{E}(Gain) \cdot n \pm o(n)$. Therefore, if $\mathbb{E}(Gain) > 0$ we have $\textsc{LimAvgInf}_S(\xi) = \textsc{LimAvgSup}_S(\xi) = \infty$. Similarly, if $\mathbb{E}(Gain) < 0$, then $\textsc{LimAvgInf}_S(\xi) = \textsc{LimAvgSup}_S(\xi) = -\infty$ with probability 1.

Now, we show condition (3). Assume that $\mathcal{A}$ is totally bounded. In every computation π if there are two configurations with the same state $(s, x_1), (s, x_2)$, then the counter's value is the same $x_1 = x_2$. To see that, consider a subcomputation from (s, x_1) to (s, x_2) and let $\mathbf{p}$ be the path that corresponds to that subcomputation. Then, $0 = \textsc{Gain}(\mathbf{p}) = x_2 - x_1$. Furthermore, since $\mathcal{A}$ is strongly connected and $(q_0, 0)$ is the initial configuration for all computations, then in all computations the state determines the value of the counter.

It follows that we can eliminate the counter and consider $\mathcal{A}$ as a Markov chain with the limit-average objective with silent moves [10]. In a Markov chain with silent moves, transitions are weighted with rational numbers and a special value $\perp$, which is skipped in the computation of partial averages. Similarly to Markov chains, in strongly-connected Markov chains with silent moves, the expected limit-average in the Markov chain is actually the limit-average of almost all paths and it is our value x. Moreover, the expected value can be computed in polynomial time [10].

More precisely, let $h(q)$ be the value of the counter in the state q. We define a Markov chain $\mathcal{M}$ corresponding to $\mathcal{A}$ as follows. We define $\mathcal{M} = \langle \{a\}, Q, \{q_0\}, \delta', P', \mu' \rangle$ such that $\delta'(q, q', a)$ holds if and only if $(q, q', x) \in \delta$ for some $x \in \mathbb{Z}$, $P'(q, q', a) = \sum_{x \in \mathbb{Z}} P(q, q', x)$, and $\mu'(q_0) = 1$. We consider the weighted Markov chain with silent moves $\langle \mathcal{M}, \mathbf{c} \rangle$ such that $\mathbf{c} \colon \delta' \to \mathbb{Z} \cup \{\perp\}$ is defined as $\mathbf{c}(q, q', a) = h(q)$, if $q \in S$ is a selecting state, and $\mathbf{c}(q, q', a) = \perp$ (is silent) otherwise. Observe that for every computation π and the corresponding $\mathbf{p}$ (without counter updates), we have $\textsc{LimAvg}_S(\pi)$ is precisely the limit average of costs $\mathbf{c}$ of transitions along $\mathbf{p}$. Since for almost all paths $\mathbf{p}$ in $\langle \mathcal{M}, \mathbf{c} \rangle$, the limit average cost of $\mathbf{p}$ is the expected cost x of $\langle \mathcal{M}, \mathbf{c} \rangle$, almost all computations in $\mathcal{A}$ have the limit-average equal to x. ◀

▶ **Lemma 10.** *Consider a probabilistic VASS($\mathbb{Z}$) $\mathcal{A}$ as in (4) of Lemma 9. There exist $c \in \mathbb{Q}$ and $\delta > 0$ such that $\textsc{LimAvgInf}_S(\xi) < c$ holds with probability greater than δ.*

Proof. Let $\mathcal{A}'$ results from $\mathcal{A}$ by assuming that the all states are initial, i.e., $Q_0 = Q$, and the initial distribution over states μ coincides with the long-run frequencies of states, i.e., $\mu(q) = x_q$.

Suppose that $\textsc{LimAvgInf}_S(\xi) = \infty$ with probability 1 w.r.t. $\mathcal{A}$. Then, it also holds with probability 1 w.r.t. $\mathcal{A}'$. Then, the average counter value at the n-th position converges to ∞ ($\lim_{n \to \infty} \textsc{Avg}_S(\xi[1, n]) = \infty$) with probability 1 in $\mathcal{A}'$. Therefore, the expected average counter value up to position n, $\mathbb{E}_{\mathcal{A}'}(\textsc{Avg}_S(\xi[1, n]))$, converges to ∞. However, the expected gain is 0, which implies that in $\mathcal{A}'$, at every position n, the expected value of the counter (in $\mathcal{A}'$) is 0. It follows that $\mathbb{E}_{\mathcal{A}'}(\textsc{Avg}_S(\xi[1, n]))$ is 0. A contradiction. ◀

▶ **Lemma 14.** *For all VASS($\mathbb{N}, 1$) $\mathcal{A}$ the following holds: there exists a computation π with $\textsc{LimAvg}_S(\pi) \leq \lambda$ if and only if there exist subcomputations ρ_0, ρ_c such that:*

- *ρ_0 is from $(q_0, 0)$ to (s, x), where $x \leq \lambda$, and*
- *ρ_c is a cycle from (s, x) to itself satisfying the following conditions:*

(a) $\textsc{Avg}_S(\rho_c) \leq \lambda$,
(b) *the number of configurations with selecting states in* ρ_c*, i.e., configurations from* $S \times \mathbb{N}$*, is* $O(|S| \cdot |\lambda|^2)$*, and*
(c) *the value of the counter in each configuration of* ρ_c *from* $S \times \mathbb{N}$ *is* $O(|S| \cdot |\lambda|^2)$.

Proof. Observe that having ρ_0, ρ_c as above, the computation $\pi = \rho_0(\rho_c)^\infty$ is a valid computation and it satisfies $\textsc{LimAvg}_S(\pi) \leq \lambda$.

Conversely, assume that there is a computation π with $\textsc{LimAvg}_S(\pi) \leq \lambda$. Consider $\epsilon > 0$ and pick a cycle subcomputation ρ from π of the average value at most $\lambda + \epsilon$ of the minimal length (all shorter subcomputations have higher average). Such a cycle exists as there has to be a configuration (s, y) with $y \leq \lambda + \epsilon$ that occurs infinitely often. Otherwise, $\textsc{LimAvg}_S(\pi) \geq \lambda + \epsilon$. Then, we divide π into cycles with ends with configuration (s, y) and there has to be a cycle with the average value at most $\lambda + \epsilon$.

We show that this minimal ρ has few *selecting configurations*, which are configurations with a selecting state. Let L be the number of selecting configurations in ρ with the counter's value at most $\lceil \lambda \rceil$ and let H be the number of selecting configurations with the counters value at least $\lceil \lambda \rceil + 1$. We lower the average if we replace the value of the configurations of the first type by 0 and the second by $\lceil \lambda \rceil + 1$ and get

$$\frac{(\lceil \lambda \rceil + 1)H}{L + H} \leq \lambda + \epsilon$$

and hence $H \leq \frac{\lambda+\epsilon}{1-\epsilon} \cdot L$ Assuming that $\epsilon \leq 0.5$, we can bound $H \leq 2 \cdot (\lceil \lambda \rceil + 1)L$.

Now, we give a bound on L. Due to minimality assumption on ρ, it cannot contain subcycles with the same properties. Suppose it has a subcycle τ. Due to minimality assumption, the subcycle has the average value exceeding $\lambda + \epsilon$. But then, ρ' obtained from ρ by removal of τ has a smaller average and a shorter length. A contradiction. It follows that for every state q and for every value $x \leq \lceil \lambda \rceil$ there is at most one configuration (q, x) in ρ. Therefore, $L \leq (\lceil \lambda \rceil + 1)|S|$. and hence

$$L + H \leq (\lceil \lambda \rceil + 1)|S| + 2 \cdot (\lceil \lambda \rceil + 1)^2|S| \leq 2 \cdot |S| \cdot (\lceil \lambda \rceil + 2)^2.$$

As previously observed the minimal value of a configuration from L is 0 and from H is $\lceil \lambda \rceil + 1$. Suppose that there is a single high value B is ρ and all other values take the minimal possible value. Then, we get:

$$\frac{B + (H - 1)\lceil \lambda \rceil}{L + H} \leq \lambda$$

thus

$$B \leq \lceil \lambda \rceil (L + 1) \leq (\lceil \lambda \rceil + 1)^2|Q|$$

and that is the bound on the maximal value of a selecting configuration. ◀

▶ **Theorem 19.** *The problem, given a probabilistic VASS*$(\mathbb{N}, k)$ $\mathcal{A}$ *under the relaxed semantics,* $S \subseteq Q$ *and* $\vec{x} \in \mathbb{Q}^k$*, decide whether* $\mathbb{E}_{\mathcal{A}}(\textsc{LimAvg}_{\vec{S}}) < \vec{x}$ *is* ExpSpace*-hard.*

Proof. Consider a VASS$(\mathbb{N}, k)$ $\mathcal{A}$, an initial configuration $(s, \vec{x}_1)$, and a target configuration $(t, \vec{x}_2)$. Without loss of generality, we assume that $\vec{x}_1 = \vec{x}_2 = \vec{0}$ and the update of each counter is $-1, 0, 1$. We construct a probabilistic VASS$(\mathbb{N})$ $\mathcal{A}^P$ based on $\mathcal{A}$ by adding an additional counter $k+1$ and a sink state r, which has only a single outgoing transition, which is a self-loop upon which counters do not change values, i.e., $(r, r, \vec{0})$. We add a transition

from t to r labeled with $\vec{0}$ and assign to it some positive probability. To make sure that the expected value of $\mathcal{A}^P$ is defined and finite, we add to every state $\mathcal{A}^P$ a transition to r labeled with $\vec{1}$ with probability $\frac{1}{2}$. We assign positive probabilities to the remaining transitions. Observe that a random computation reaches the sink r with probability 1, and the probability that it happens after more than n steps is bounded by $(\frac{1}{2})^n$. The value of the counter after n steps is at most n. Therefore, the expected limit-average of counters $1, \ldots, k$ is bounded by $\sum_{j=1}^{\infty}(\frac{1}{2})^i \cdot i = 2$. Upon reaching r the counter $k+1$ has value 0, if the previous state was t, and 1 otherwise. Therefore, the expected limit-average is strictly less than $(2+\epsilon, \ldots, 2+\epsilon, 1)$ (for any $\epsilon > 0$) if and only if there is a computation from $(s, \vec{0})$ to $(t, \vec{y})$ (with any $\vec{y}$) in $\mathcal{A}$. ◀

Games Where You Can Play Optimally with Arena-Independent Finite Memory

Patricia Bouyer
LSV – CNRS & ENS Paris-Saclay, Université Paris-Saclay, Saint-Aubin, France

Stéphane Le Roux
LSV – CNRS & ENS Paris-Saclay, Université Paris-Saclay, Saint-Aubin, France

Youssouf Oualhadj
LACL, UPEC, Créteil, France

Mickael Randour
F.R.S.-FNRS & UMONS – Université de Mons, Mons, Belgium

Pierre Vandenhove
F.R.S.-FNRS & UMONS – Université de Mons, Mons, Belgium
LSV – CNRS & ENS Paris-Saclay, Université Paris-Saclay, Saint-Aubin, France

Abstract

For decades, two-player (antagonistic) games on graphs have been a framework of choice for many important problems in theoretical computer science. A notorious one is controller synthesis, which can be rephrased through the game-theoretic metaphor as the quest for a winning strategy of the system in a game against its antagonistic environment. Depending on the specification, *optimal* strategies might be simple or quite complex, for example having to use (possibly infinite) memory. Hence, research strives to understand which settings allow for simple strategies.

In 2005, Gimbert and Zielonka [26] provided a complete characterization of preference relations (a formal framework to model specifications and game objectives) that admit *memoryless* optimal strategies for both players. In the last fifteen years however, practical applications have driven the community toward games with complex or multiple objectives, where memory – finite or infinite – is almost always required. Despite much effort, the exact frontiers of the class of preference relations that admit *finite-memory* optimal strategies still elude us.

In this work, we establish a complete characterization of preference relations that admit optimal strategies using *arena-independent* finite memory, generalizing the work of Gimbert and Zielonka to the finite-memory case. We also prove an equivalent to their celebrated corollary of great practical interest: if both players have optimal (arena-independent-)finite-memory strategies in all one-player games, then it is also the case in all two-player games. Finally, we pinpoint the boundaries of our results with regard to the literature: our work completely covers the case of arena-independent memory (e.g., multiple parity objectives, lower- and upper-bounded energy objectives), and paves the way to the arena-dependent case (e.g., multiple lower-bounded energy objectives).

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases two-player games on graphs, finite-memory determinacy, optimal strategies

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.24

Related Version Full paper on arXiv: `https://arxiv.org/abs/2001.03894`

Funding Research supported by F.R.S.-FNRS under Grant n° F.4520.18 (ManySynth), F.R.S.-FNRS mobility funding for scientific missions (Y. Oualhadj in UMONS, 2018), and ENS Paris-Saclay visiting professorship (M. Randour, 2019). Mickael Randour is an F.R.S.-FNRS Research Associate and Pierre Vandenhove is an F.R.S.-FNRS Research Fellow.

Acknowledgements We extend our warmest thanks to Mathieu Sassolas, for inspiring discussions that were essential in starting this work.

1 Introduction

Controller synthesis through the game-theoretic metaphor. Two-player games on (finite) graphs are studied extensively, in particular for their application to controller synthesis for reactive systems (see, e.g., [28, 35, 7, 2]). The seminal model is *antagonistic* (i.e., *zero-sum* if one chooses a quantitative view): player 1 ($\mathcal{P}_1$) is seen as the system to control, player 2 ($\mathcal{P}_2$) as its antagonistic environment, and the game models their interaction. Each vertex of the game graph (called *arena*) models a *state* of the system and belongs to one of the players. Players take turns moving a pebble from state to state according to the edges, each player choosing the destination whenever the pebble is on one of his states. These choices are made according to the *strategy* of the player, which, in general, might use memory (bounded or not) of the past moves to prescribe the next action.

The resulting infinite sequence of states, called *play*, represents the execution of the system. The objective of $\mathcal{P}_1$ is to enforce a given *specification*, often encoded as a *winning condition* (i.e., a set of winning plays) or as a *payoff function* to maximize (i.e., a quantitative performance to optimize). This paradigm focuses on the *worst-case* performance of the system, hence $\mathcal{P}_2$'s goal is to prevent $\mathcal{P}_1$ from achieving his objective.

The goal of *synthesis* is thus to decide if $\mathcal{P}_1$ has a *winning strategy*, i.e., one ensuring a given winning condition or guaranteeing a given payoff threshold, against all possible strategies of $\mathcal{P}_2$, and to build such a strategy efficiently if it exists.

Winning strategies are *formal blueprints* for controllers to implement in applications. Therefore, their complexity is of tremendous importance: the simpler the strategy, the easier and cheaper it will be to build the controller and maintain it. This explains why a lot of research effort is constantly put in identifying the exact complexity (in terms of memory and/or randomness) of strategies needed to play *optimally* (i.e., to the best of the player's ability) for each specific class of games and objectives (e.g., [26, 17, 14, 40, 22, 10, 1, 4, 39, 5, 11]). Alongside the *practical interest* lies the *theoretical puzzle*: understanding the underlying mechanisms and implicit properties of games that lead to "simple" strategies being sufficient. Given the numerous connections between games and various branches of mathematics and computer science, this fundamental question has interest in its own right.

Preference relations. There are two prominent ways to formalize a game objective in the literature. The first one, dubbed *quantitative* and inspired by games in economics, is to use *payoff functions* mapping plays to numerical values, and to see $\mathcal{P}_1$ as a maximizer player. This is for example the case of mean-payoff games [19]. The second one, called *qualitative*, is to define a *set of winning plays* – called *winning condition* – induced by some property, as in, e.g., parity games [20, 41]. The two formalisms are strongly linked: the classical decision problem for quantitative games is to fix a payoff threshold and ask if $\mathcal{P}_1$ has a strategy to guarantee it, transforming the problem into a qualitative one (where the winning plays are all those with a payoff at least equal to the threshold). To define payoff functions or winning conditions, one often uses weights, priorities, colors, etc, on states or edges of the arena.

In this work, we walk in the footsteps of Gimbert and Zielonka [26]: we associate a *color* to each edge of our arenas, and we adopt the abstract formalism of *preference relations* over infinite sequences of colors (induced by plays). This general formalism permits to encode virtually all classical game objectives, both qualitative and quantitative, and lets us reason in a well-founded framework under minimal assumptions.

Memoryless optimal strategies. Remarkably, several canonical classes of games that have been around for decades and proved their usefulness over and over – e.g., mean-payoff [19], parity [20, 41], or energy games [12] – share a desirable property: they all admit *memoryless optimal strategies for both players*. That is, for every strategy σ_i of $\mathcal{P}_i$, there is a strategy σ_i^{ML} which is *at least as good* (i.e., wins whenever σ_i wins or ensures at least the same payoff) and that uses no memory at all. Such a memoryless strategy always picks the same edge when in the same state, regardless of what happened before in the game.

Memoryless strategies are the simplest kind of strategies one can use. Therefore, it is quite interesting that they suffice for objectives as rich as the ones we just discussed. Following this observation, a lot of effort has been put in understanding which games admit memoryless optimal strategies, and in identifying the exact frontiers of *memoryless determinacy*. Let us mention, non-exhaustively, works by Gimbert and Zielonka [25, 26] (culminating in a complete characterization), Aminof and Rubin [1] (through the prism of first-cycle games), and Kopczyński [31] (half-positional determinacy). All these advances were built by identifying the common underlying mechanisms in ad hoc proofs for specific classes of games, and generalizing them to wide classes (e.g., the first-cycle games of [1] are inspired by the seminal paper of Ehrenfeucht and Mycielski on mean-payoff games [19]).

Gimbert and Zielonka's approach. Arguably, the most important result in this direction is the *complete characterization* of preference relations admitting memoryless optimal strategies, established in [26], fifteen years ago. By complete characterization, we mean sufficient *and* necessary conditions on the preference relations.

It can be stated as follows: a preference relation admits memoryless optimal strategies for both players on all arenas if and only if the relation (used by $\mathcal{P}_1$) and its inverse (used by $\mathcal{P}_2$) are *monotone* and *selective*. These concepts will be defined formally in Section 3, but let us give an intuition here. Roughly, a preference relation is monotone if it is stable under *prefix addition*: given two sequences of colors such that one is *strictly* preferred to the other, it is impossible to reverse this order of preference by adding the same prefix to both sequences. Selectivity is similarly defined with regard to *cycle mixing*: if a preference relation is selective, then, starting from two sequences of colors, it is impossible to create a third one by mixing the first two in such a way that the third one is strictly preferred to the first two. These elegant notions coincide with the natural intuition that memoryless strategies suffice if there is no interest in behaving differently in a state depending on what happened before.

In addition to this complete characterization, Gimbert and Zielonka proved another great result, of high interest in practice [26, Corollary 7]: as a by-product of their approach, they obtain that if memoryless strategies suffice in all one-player games of $\mathcal{P}_1$ and all one-player games of $\mathcal{P}_2$, they also suffice in all two-player games. Such a *lifting corollary* provides a neat and easy way to prove that a preference relation admits memoryless optimal strategies *without proving monotony and selectivity at all*: proving it in the two one-player subcases, which is generally much easier as it boils down to graph reasoning, and then lifting the result to the general two-player case through the corollary.

The rise of memory. The need to model complex specifications has shifted research toward games where multiple (quantitative and qualitative) objectives co-exist and interact, requiring the analysis of *interplay* and *trade-offs* between several objectives. Hence, a lot of effort is put in studying games where objectives are actually conjunctions of objectives, or even richer Boolean combinations. See for example [16] for combinations of parity, [13, 17, 30] for combinations of energy and parity, [40] for combinations of mean-payoff, [5, 4] for combinations of energy and average-energy, [11] for combinations of energy and mean-payoff, [14] for combinations of total-payoff, or [14, 10, 8] for combinations of window objectives.

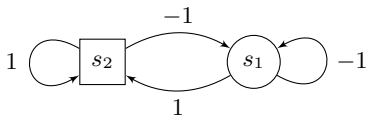


Figure 1 $\mathcal{P}_1$ (circle) needs infinite memory to win.

When considering such rich objectives, *memoryless strategies usually do not suffice*, and one has to use an amount of memory that can quickly become an obstacle to implementation (e.g., exponential memory) or that can prevent it completely (infinite memory). Establishing precise memory bounds for such general combinations of objectives is tricky and sometimes counterintuitive. For example, while energy games and mean-payoff games are inter-reducible in the single-objective setting, exponential-memory strategies are both sufficient and necessary for conjunctions of energy objectives [17, 30] while *infinite-memory* strategies are required for conjunctions of mean-payoff ones [40].

A natural question arises: *which preference relations do admit finite-memory optimal strategies?* Surprisingly, whether an equivalent to Gimbert and Zielonka's characterization could be obtained for finite memory or not has remained an open question up to now. It is worth noticing that such an equivalent could be of tremendous help in practice, especially if a *lifting corollary* also holds: see for example [5, 4, 11], where proving that finite-memory strategies suffice in one-player games was fairly easy, in contrast to the high complexity of the two-player case – a lifting corollary could grant the two-player case for free!

Having said that, one has to hope that the following corollary can be established: "if *finite-memory* strategies suffice in all one-player games of $\mathcal{P}_1$ and all one-player games of $\mathcal{P}_2$, they also suffice in all two-player games." Unfortunately, this hope is but a delusion.

Lifting corollary: a counterexample. Consider games where the colors are integers, and the objective of $\mathcal{P}_1$ is to create a play such that (a) the running sum of weights grows up to infinity (e.g., consider its lim inf to define it properly), *or* (b) this running sum of weights takes value zero infinitely often. As this defines a qualitative objective, the corresponding preference relation induces only two equivalence classes: *winning* and *losing* plays. The inverse relation, used by $\mathcal{P}_2$, is trivial to obtain. It is fairly easy to prove that $\mathcal{P}_1$ always has finite-memory optimal strategies in his one-player games (i.e., games where $\mathcal{P}_2$ has no choice), and so does $\mathcal{P}_2$ in his one-player games.

Now, consider the very simple two-player game depicted in Figure 1. Player $\mathcal{P}_1$ (circle) has an *infinite-memory* strategy to win: keeping track of the running sum of weights (which is unbounded, hence the need for infinite memory) and looping in s_1 up to the point where this sum hits zero, then going to s_2. This strategy ensures victory because either $\mathcal{P}_2$ always goes back to s_1, in which case (b) is satisfied; or $\mathcal{P}_2$ eventually loops forever on s_2, in which case (a) is satisfied. It remains to argue that $\mathcal{P}_1$ has no *finite-memory* winning strategy in this game. This can be done using a standard argument: whatever the amount of memory used by $\mathcal{P}_1$, $\mathcal{P}_2$ may loop in s_2 long enough as to exceed the bound up to which $\mathcal{P}_1$ can track the sum accurately; thus dooming $\mathcal{P}_1$ to fail to reset the sum to zero in s_1 infinitely often.

This modest example proves that *Gimbert and Zielonka's approach cannot work in full generality in the finite-memory case*, and for good reasons. Informally, in this case, the corollary breaks down because of (the absence of some sort of) monotony. In the case of *memoryless* strategies, as in [26], $\mathcal{P}_1$ is already doomed in one-player games in the absence of monotony: two prefixes to distinguish – in order to play optimally – can be hardcoded as different paths leading to the same state in a game arena, as if they were chosen by $\mathcal{P}_2$ in a two-player game. In the case of *finite-memory* strategies, however, the situation is different.

In one-player games, the number of such paths that can be hardcoded in an arena is always bounded, hence finite memory might suffice to react, i.e., to keep track of which prefix is the current one and how to behave accordingly. However, in two-player games, $\mathcal{P}_2$ might create an *infinite* number of prefixes to distinguish (using a cycle), thus requiring $\mathcal{P}_1$ to use infinite memory to be able to do so. This is exactly what happens in the example above: in any one-player game, the largest sum that $\mathcal{P}_1$ has to track is bounded, whereas $\mathcal{P}_2$ can make this sum as large as he wants in two-player games.

Our approach. We generalize Gimbert and Zielonka's results – characterization *and* lifting corollary – to the case of *arena-independent* finite memory. That is, we encompass *all* situations where the memory needed by the two players is *solely dependent on the preference relation* (e.g., colors, dimensions of weight vectors), and *not* on the game arena (i.e., number of edges/states). Let us take some examples.

- All memoryless-determined relations – studied in [26] – use arena-independent memory: the memory required, *none*, is the same for all arenas.
- Combinations of parity objectives use arena-independent memory [16]: the memory only depends on the number of objectives and the number of priorities – both parameters of the preference relation, not on the size of the arena.
- Lower- and upper-bounded energy objectives also use arena-independent memory (see, e.g., [3, 5, 4]): the memory only depends on the bounds and the weights – parameters of the preference relation, not on the size of the arena.
- On the contrary, combinations of lower-bounded energy objectives (with no upper bound) require *arena-dependent* memory [17, 30]: it depends on the size of the arena in addition to the weights used in it. Such a setting falls outside the scope of our results.

This informal concept of arena-independent memory is transparent in our work: in all our results, we use *memory skeletons* – essentially Mealy machines without a next-action function (Section 2) – that suffice for *all* arenas, and that are at the basis of the strategies we build. A quick look at our main concepts (Section 3) and results (Section 4) suffices to grasp the formalism behind this intuition.

This restriction to arena-independent memory is natural given the counterexample to a general approach presented above. It is also important to note that it is not as restrictive as it may seem, as hinted by the examples above: we are *not restricted to constant memory* but to memory only depending on the *parameters of the preference relation* (or equivalently, objective), and not of the arena. This framework thus already encompasses many objectives from the literature – e.g., [19, 20, 41, 12, 5, 21, 16, 10, 14, 3, 5, 4], as well as possible extensions. More details in Appendix A.

The *arena-independent* case, which we solve here, is an exact equivalent to Gimbert and Zielonka's results in the finite-memory case: the memoryless case is de facto arena-independent. Therefore, this paper strictly generalizes [26] by allowing to study any arena-independent memory skeleton instead of the unique trivial one corresponding to memoryless strategies.

Outline of our contributions. Informally, our characterization can be stated as follows: given a preference relation and a memory skeleton $\mathcal{M}$, both players have optimal finite-memory strategies based on skeleton $\mathcal{M}$ in all games if and only if the relation and its inverse are $\mathcal{M}$-monotone and $\mathcal{M}$-selective.

These last two concepts are keys to our approach. Intuitively, they correspond to Gimbert and Zielonka's monotony and selectivity, *modulo a memory skeleton*. Recall that monotony and selectivity are related to stability of the preference relation with regard to prefix addition

and cycle mixing, respectively. Our more general concepts of $\mathcal{M}$-*monotony* and $\mathcal{M}$-*selectivity* serve the same purpose, but they only compare sequences of colors that are deemed equivalent by the memory skeleton. For the sake of illustration, take selectivity: it implies that one has no interest in mixing different cycles of the game arena. For its generalization, the memory skeleton is taken into account: $\mathcal{M}$-selectivity implies that one has no interest in mixing cycles of the game arena *that are read as cycles on the same memory state in the skeleton* $\mathcal{M}$.

Let us give a quick breakdown of our paper. Due to space constraints, we only provide an intuitive overview of our results and technical approach in this conference version: formal details and proofs, along with additional results, can be found in the full article [6].

In Section 2, we introduce some basic notions, including the memory skeletons, and we establish several technical results. We also discuss *optimal strategies* and *Nash equilibria*. In Section 3, we introduce $\mathcal{M}$-monotony and $\mathcal{M}$-selectivity, cornerstones of our work. We also present two essential tools: *prefix-covers* and *cyclic-covers* of arenas. Section 4 states formally our characterization (Theorem 9), as well as the corresponding lifting corollary (Corollary 10), from one-player to two-player games. We show an example of application in Section 5. Finally, we give an overview of the technical highlights of our approach in Section 6 – its details are broken down in several intermediate results in our full paper [6].

In a nutshell, the proof of the characterization (Theorem 9) is split in two. We first establish that (the sufficiency of) finite memory based on $\mathcal{M}$ implies $\mathcal{M}$-monotony and $\mathcal{M}$-selectivity of the preference relation. The crux is to build game arenas based on automata recognizing the languages involved in the two concepts, and to use the existence of finite-memory optimal strategies in these arenas to prove that $\mathcal{M}$-monotony and $\mathcal{M}$-selectivity hold. To prove the converse implication, we proceed in two steps, first establishing the existence of *memoryless* optimal strategies in "covered" arenas, and then building on it to obtain the existence of *finite-memory* optimal strategies in general arenas. The main technical tools we use are Nash equilibria and the aforementioned notions of prefix-covers and cyclic-covers.

Alongside the technical details, we analyze our characterization in Appendix A: we highlight some limitations and interesting features, compare our techniques with Gimbert and Zielonka's, discuss our place in the research landscape, and sketch directions for future work. Let us just stress already that our result – relating a memory skeleton $\mathcal{M}$ and preference relations for which this skeleton suffices – cannot be obtained by simply considering product arenas and invoking Gimbert and Zielonka's result on memoryless determinacy [26]. While, of course, memoryless strategies on product arenas correspond to memoryfull strategies on original arenas (as we will formally establish in Lemma 1), invoking [26] requires to be able to quantify on *all* arenas, not only *product* arenas. Filling this gap is exactly the goal of this paper, and it is made possible through the new concepts we sketched above.

2 Preliminaries

We only give here the notions and results necessary to understand this overview. Necessary notions and results – some of them interesting in their own right – to understand the technical details of the approach are found in the full paper [6].

Automata and languages of colors. Let C be an arbitrary set of *colors*. We assume knowledge of *non-deterministic finite-state automata (NFA)*, which recognize *regular languages*. For any finite subset $B \subseteq C$, we denote by $\mathsf{Reg}(B)$ the set of all regular languages over B. Let $\mathcal{R}(C) = \bigcup_{B \subseteq C, |B|<\infty} \mathsf{Reg}(B)$, that is, all the regular languages built over C.

Let $K \subseteq C^*$ be a language of *finite* words. We write $\mathsf{Prefs}(K)$ for the set of all *prefixes* of words in K. We define the set of *infinite* words $[K] = \{w = c_1 \dots \in C^\omega \mid \forall\, n \geq 1,\, c_1 \dots c_n \in \mathsf{Prefs}(K)\}$, which contains all infinite words for which every finite prefix is a prefix of a word in K. Intuitively, if K is regular, $[K]$ is the language of infinite words that correspond to infinite paths that can always branch and reach a final state, on an automaton for K. Given a finite word $w \in C^*$ and a language $K \subseteq C^*$, we write wK for their concatenation, i.e., the language $wK = \{w' = ww' \mid w' \in K\} \subseteq C^*$.

Arenas. We consider two players: player 1 ($\mathcal{P}_1$) and player 2 ($\mathcal{P}_2$). An *arena* is a tuple $\mathcal{A} = (S_1, S_2, E)$ such that $S = S_1 \uplus S_2$ (disjoint union) is a finite set of *states* partitioned into states of $\mathcal{P}_1$ (S_1) and $\mathcal{P}_2$ (S_2), and $E \subseteq S \times C \times S$ is a finite set of *edges*. Let $\mathsf{col}\colon E \to C$ be the projection of edges to *colors* and $\widehat{\mathsf{col}}$ its natural extension to *sequences* of edges. For an edge $e \in E$, we use $\mathsf{in}(e)$ and $\mathsf{out}(e)$ to denote its starting state and arrival state respectively, i.e., $e = (\mathsf{in}(e), \mathsf{col}(e), \mathsf{out}(e))$. We assume all arenas to be non-blocking, i.e., for all $s \in S$, there exists $e \in E$ such that $\mathsf{in}(e) = s$. For $i \in \{1, 2\}$, we call an arena $\mathcal{A} = (S_1, S_2, E)$ a $\mathcal{P}_i$*'s one-player arena* if for all $s \in S_{3-i}$, $|\{e \in E \mid \mathsf{in}(e) = s\}| = 1$ – that is, $\mathcal{P}_{3-i}$ has no choice.

Let $\mathsf{Hists}(\mathcal{A}, s)$ denote the *histories* in $\mathcal{A}$ from $s \in S$, i.e., finite sequences of edges $\rho = e_1 \dots e_n \in E^+$ such that $\mathsf{in}(e_1) = s$ and for all i, $1 \leq i < n$, $\mathsf{out}(e_i) = \mathsf{in}(e_{i+1})$. Let $\mathsf{Plays}(\mathcal{A}, s)$ denote the *plays* in $\mathcal{A}$ from $s \in S$, i.e., infinite sequences $\pi = e_1 e_2 \dots \in E^\omega$ such that $\mathsf{in}(e_1) = s$ and for all $i \geq 1$, $\mathsf{out}(e_i) = \mathsf{in}(e_{i+1})$. We write $\mathsf{Hists}(\mathcal{A}, S')$ and $\mathsf{Plays}(\mathcal{A}, S')$ for unions over $S' \subseteq S$, and write $\mathsf{Hists}(\mathcal{A})$ and $\mathsf{Plays}(\mathcal{A})$ for the unions over all states of $\mathcal{A}$.

Let $\rho = e_1 \dots e_n \in \mathsf{Hists}(\mathcal{A})$ (resp. $\pi = e_1 e_2 \dots \in \mathsf{Plays}(\mathcal{A})$): we extend the operator in to histories (resp. plays) by identifying $\mathsf{in}(\rho)$ (resp. $\mathsf{in}(\pi)$) to $\mathsf{in}(e_1)$. We proceed similarly for out and histories: $\mathsf{out}(\rho) = \mathsf{out}(e_n)$. For the sake of convenience, we consider that any set $\mathsf{Hists}(\mathcal{A}, s)$ contains the *empty history* λ_s such that $\mathsf{in}(\lambda_s) = \mathsf{out}(\lambda_s) = s$. We write $\mathsf{Hists}_i(\mathcal{A}, s)$ and $\mathsf{Hists}_i(\mathcal{A})$ for the subsets of histories ρ such that $\mathsf{out}(\rho) \in S_i$, $i \in \{1, 2\}$, i.e., histories whose last state belongs to $\mathcal{P}_i$. For any set $H \subseteq \mathsf{Hists}(\mathcal{A})$, we write $\widehat{\mathsf{col}}(H)$ for its projection to colors, i.e., $\widehat{\mathsf{col}}(H) = \{\widehat{\mathsf{col}}(\rho) \mid \rho \in H\}$. We do the same for sets of plays.

Memory skeletons. A *memory skeleton* is a tuple $\mathcal{M} = (M, m_{\mathsf{init}}, \alpha_{\mathsf{upd}})$ where M is a *finite* set of *states*, $m_{\mathsf{init}} \in M$ is a fixed *initial* state and $\alpha_{\mathsf{upd}}\colon M \times C \to M$ is an *update function*. We write $\widehat{\alpha_{\mathsf{upd}}}$ for the natural extension of α_{upd} to sequences in C^*. Memory skeletons are deterministic and might have an *infinite* number of transitions, in contrast to NFA. We define the *trivial memory skeleton* as $\mathcal{M}_{\mathsf{triv}} = (M = \{m_{\mathsf{init}}\}, m_{\mathsf{init}}, \alpha_{\mathsf{upd}}\colon \{m_{\mathsf{init}}\} \times C \to \{m_{\mathsf{init}}\})$: it permits to formalize memoryless strategies [26] in our framework.

Let $\mathcal{M} = (M, m_{\mathsf{init}}, \alpha_{\mathsf{upd}})$ be a skeleton. For $m, m' \in M$, we define the language $L_{m,m'} = \{w \in C^* \mid \widehat{\alpha_{\mathsf{upd}}}(m, w) = m'\}$ that contains all words that can be read from m to m' in $\mathcal{M}$.

Let $\mathcal{M}^1 = (M^1, m^1_{\mathsf{init}}, \alpha^1_{\mathsf{upd}})$ and $\mathcal{M}^2 = (M^2, m^2_{\mathsf{init}}, \alpha^2_{\mathsf{upd}})$. Their *product* $\mathcal{M}^1 \otimes \mathcal{M}^2$ is the memory skeleton $\mathcal{M} = (M, m_{\mathsf{init}}, \alpha_{\mathsf{upd}})$ where $M = M^1 \times M^2$, $m_{\mathsf{init}} = (m^1_{\mathsf{init}}, m^2_{\mathsf{init}})$, and, for all $m^1 \in M^1$, $m^2 \in M^2$, $c \in C$, $\alpha_{\mathsf{upd}}((m^1, m^2), c) = (\alpha^1_{\mathsf{upd}}(m^1, c), \alpha^2_{\mathsf{upd}}(m^2, c))$. That is, the memories are updated in parallel when a color is read.

Product arenas. Let $\mathcal{A} = (S_1, S_2, E)$ be an arena and $\mathcal{M} = (M, m_{\mathsf{init}}, \alpha_{\mathsf{upd}})$ be a skeleton. Their *product* $\mathcal{A} \ltimes \mathcal{M}$ is the arena (S'_1, S'_2, E') where $S'_1 = S_1 \times M$, $S'_2 = S_2 \times M$, and $E' \subseteq S' \times C \times S'$, with $S' = S'_1 \uplus S'_2$, is such that $((s_1, m_1), c, (s_2, m_2)) \in E'$ if and only if $(s_1, c, s_2) \in E$ and $\alpha_{\mathsf{upd}}(m_1, c) = m_2$. That is, the memory is updated according to the colors of the edges in E. Though $\mathcal{M}$ might contain an infinite number of transitions, $\mathcal{A} \ltimes \mathcal{M}$ is always finite, as E is finite. Since we assume $\mathcal{A}$ is non-blocking, it is also the case of $\mathcal{A} \ltimes \mathcal{M}$.

Strategies. A *strategy* σ_i for $\mathcal{P}_i$, $i \in \{1,2\}$, on arena $\mathcal{A} = (S_1, S_2, E)$, is a function $\sigma_i \colon \mathsf{Hists}_i(\mathcal{A}) \to E$ such that for all $\rho \in \mathsf{Hists}_i(\mathcal{A})$, $\mathsf{in}(\sigma_i(\rho)) = \mathsf{out}(\rho)$. Let $\Sigma_i(\mathcal{A})$ be the set of all strategies of $\mathcal{P}_i$ on $\mathcal{A}$. A *finite-memory strategy* σ_i can be encoded as a *Mealy machine*, i.e., a skeleton $\mathcal{M} = (M, m_{\mathsf{init}}, \alpha_{\mathsf{upd}})$ with transitions over a *finite subset of colors* $B \subseteq C$, enriched with a *next-action function* $\alpha_{\mathsf{nxt}} \colon M \times S_i \to E$ such that for all $m \in M$, $s \in S_i$, $\mathsf{in}(\alpha_{\mathsf{nxt}}(m,s)) = s$. Given a Mealy machine $\Gamma_{\sigma_i} = (\mathcal{M}, \alpha_{\mathsf{nxt}})$, strategy σ_i is defined as follows:

- $\forall\, s \in S_i,\ \sigma_i(\lambda_s) = \alpha_{\mathsf{nxt}}(m_{\mathsf{init}}, s)$,
- $\forall\, \rho \cdot e \in \mathsf{Hists}_i(\mathcal{A}),\ e \in E,\ \sigma_i(\rho \cdot e) = \alpha_{\mathsf{nxt}}\left(\widehat{\alpha_{\mathsf{upd}}}\left(m_{\mathsf{init}}, \widehat{\mathsf{col}}\,(\rho \cdot e)\right), \mathsf{out}(e)\right)$.

Let $\Sigma_i^{\mathsf{FM}}(\mathcal{A})$ be the finite-memory strategies of $\mathcal{P}_i$ on $\mathcal{A}$. We say that a strategy $\sigma_i \in \Sigma_i^{\mathsf{FM}}(\mathcal{A})$ is *based on memory skeleton* $\mathcal{M}$ if it can be encoded as a Mealy machine $\Gamma_{\sigma_i} = (\mathcal{M}, \alpha_{\mathsf{nxt}})$, as above. We implicitly assume that strategies of $\Sigma_i^{\mathsf{FM}}(\mathcal{A})$ are built by restricting the transitions of their skeleton $\mathcal{M}$ to the actual subset of colors appearing in $\mathcal{A}$. A strategy σ_i is *memoryless* if it is a function $\sigma_i \colon S_i \to E$, or equivalently, if it is based on $\mathcal{M}_{\mathsf{triv}}$.

We write $\mathsf{Plays}(\mathcal{A}, s, \sigma_i)$ for the plays *consistent* with a strategy σ_i of $\mathcal{P}_i$ from a state s, i.e., plays $\pi = e_1 e_2 \ldots \in \mathsf{Plays}(\mathcal{A}, s)$ such that for all $\rho = e_1 \ldots e_n$, $\mathsf{out}(\rho) \in S_i \implies \sigma_i(\rho) = e_{n+1}$. We write $\mathsf{Plays}(\mathcal{A}, s, \sigma_1, \sigma_2)$ for the singleton set containing the unique play consistent with a couple of strategies for the two players. We use similar notations for histories.

Preference relations. Let $\sqsubseteq$ be a total preorder on C^ω, called *preference relation*. We consider *antagonistic* games, where the objective of $\mathcal{P}_1$ is to create the best possible play with regard to $\sqsubseteq$ whereas the objective of $\mathcal{P}_2$ is the opposite. That is, $\mathcal{P}_2$ uses the inverse relation $\sqsubseteq^{-1}$. This corresponds to *zero-sum* games when using a quantitative framework.

Given $w, w' \in C^\omega$, we write $w \sqsubset w'$ if we have $\neg(w' \sqsubseteq w)$ since the preorder is total. We extend $\sqsubseteq$ to subsets: for $W, W' \subseteq C^\omega$, $W \sqsubseteq W' \iff \forall\, w \in W, \exists\, w' \in W', w \sqsubseteq w'$. We also write $W \sqsubset W' \iff \exists\, w' \in W', \forall\, w \in W, w \sqsubset w'$. Note that $W \sqsubset W' \iff \neg(W' \sqsubseteq W)$.

To compare a word $w \in C^\omega$ with a language $K \subseteq C^\omega$, we simply identify it to $\{w\}$.

Games. A (deterministic turn-based two-player) *game* is a tuple $\mathcal{G} = (\mathcal{A}, \sqsubseteq)$ where $\mathcal{A}$ is an arena and $\sqsubseteq$ is a preference relation. All classical objectives from the literature (both qualitative and quantitative) can be expressed in the general framework of preference relations (see Example 3 in [6]). For $i \in \{1,2\}$, a *$\mathcal{P}_i$'s one-player game* is a game $\mathcal{G} = (\mathcal{A}, \sqsubseteq)$ such that $\mathcal{A}$ is a $\mathcal{P}_i$'s one-player arena.

Optimal strategies. Let $\mathcal{G} = (\mathcal{A}, \sqsubseteq)$ be a game on arena $\mathcal{A} = (S_1, S_2, E)$. Given a $\mathcal{P}_i$-strategy $\sigma_i \in \Sigma_i(\mathcal{A})$ and a state $s \in S$, we define

$$\mathsf{UCol}_{\sqsubseteq}(\mathcal{A}, s, \sigma_i) = \{w \in C^\omega \mid \exists\, \sigma_{3-i} \in \Sigma_{3-i}(\mathcal{A}),\ \widehat{\mathsf{col}}(\mathsf{Plays}(\mathcal{A}, s, \sigma_1, \sigma_2)) \sqsubseteq w\},$$
$$\mathsf{DCol}_{\sqsubseteq}(\mathcal{A}, s, \sigma_i) = \{w \in C^\omega \mid \exists\, \sigma_{3-i} \in \Sigma_{3-i}(\mathcal{A}),\ w \sqsubseteq \widehat{\mathsf{col}}(\mathsf{Plays}(\mathcal{A}, s, \sigma_1, \sigma_2))\}.$$

Note that $\mathsf{DCol}_{\sqsubseteq}(\mathcal{A}, s, \sigma_i) = \mathsf{UCol}_{\sqsubseteq^{-1}}(\mathcal{A}, s, \sigma_i)$. Intuitively, $\mathsf{UCol}_{\sqsubseteq}$ and $\mathsf{DCol}_{\sqsubseteq}$ represent the *upward* and *downward closures* of sequences of colors (consistent with a strategy) with respect to the preference relation.

Taking the standpoint of $\mathcal{P}_1$, we say that $\sigma_1 \in \Sigma_1(\mathcal{A})$ is *at least as good as* $\sigma_1' \in \Sigma_1(\mathcal{A})$ from $s \in S$ if $\mathsf{UCol}_{\sqsubseteq}(\mathcal{A}, s, \sigma_1) \subseteq \mathsf{UCol}_{\sqsubseteq}(\mathcal{A}, s, \sigma_1')$. Intuitively, σ_1 is at least as good as σ_1' if the "worst-case" plays consistent with σ_1 are at least as good as the ones consistent with σ_1'. The UCol operator is useful to define this notion properly even in the case where there is no "worst-case" play (i.e., if the infimum used in the classical quantitative setting is not reached). Similar notions have been used before, e.g., in [36]. Symmetrically, for $\mathcal{P}_2$, we say that $\sigma_2 \in \Sigma_2(\mathcal{A})$ is at least as good as $\sigma_2' \in \Sigma_2(\mathcal{A})$ from $s \in S$ if $\mathsf{DCol}_{\sqsubseteq}(\mathcal{A}, s, \sigma_2) \subseteq \mathsf{DCol}_{\sqsubseteq}(\mathcal{A}, s, \sigma_2')$.

Now, we say that $\sigma_i \in \Sigma_i(\mathcal{A})$ of $\mathcal{P}_i$ is *optimal* from $s \in S$, aka s-optimal, if it is at least as good as every other $\sigma'_i \in \Sigma_i(\mathcal{A})$ from s. We extend this notation to subsets of states in the natural way, and we say that a strategy σ_i is *uniformly-optimal* if it is S-optimal.

Our goal is to characterize the preference relations that admit *uniformly-optimal finite-memory (UFM) strategies based on a given skeleton* $\mathcal{M}$ in all arenas. We also discuss the simpler case of *uniformly-optimal memoryless (UML) strategies*, which corresponds to the subcase studied by Gimbert and Zielonka [26], using the trivial skeleton $\mathcal{M}_{\mathsf{triv}}$.

In that respect, the following link is important to observe.

▶ **Lemma 1.** *Let $\mathcal{G} = (\mathcal{A}, \sqsubseteq)$ be a game on arena $\mathcal{A} = (S_1, S_2, E)$. Let $\mathcal{M} = (M, m_{\mathsf{init}}, \alpha_{\mathsf{upd}})$ be a memory skeleton and let $\sigma_i \in \Sigma_i^{\mathsf{FM}}(\mathcal{A})$ be a finite-memory strategy encoded by the Mealy machine $\Gamma_{\sigma_i} = (\mathcal{M}, \alpha_{\mathsf{nxt}})$. Then, σ_i is a UFM strategy in $\mathcal{G}$ if and only if α_{nxt} corresponds to an $(S \times \{m_{\mathsf{init}}\})$-optimal memoryless strategy in $\mathcal{G}' = (\mathcal{A} \ltimes \mathcal{M}, \sqsubseteq)$.*

Nash equilibria. We use Nash equilibria [34] as tools to establish the existence of optimal strategies in some of our proofs. Let $\mathcal{G} = (\mathcal{A}, \sqsubseteq)$ be a game on arena $\mathcal{A} = (S_1, S_2, E)$. Formally, a *Nash equilibrium* (NE) from a state $s \in S$ is a couple of strategies $(\sigma_1, \sigma_2) \in \Sigma_1(\mathcal{A}) \times \Sigma_2(\mathcal{A})$ such that, for all $\sigma'_1 \in \Sigma_1(\mathcal{A})$, $\sigma'_2 \in \Sigma_2(\mathcal{A})$,

$$\widehat{\mathsf{col}}(\mathsf{Plays}(\mathcal{A}, s, \sigma'_1, \sigma_2)) \sqsubseteq \widehat{\mathsf{col}}(\mathsf{Plays}(\mathcal{A}, s, \sigma_1, \sigma_2)) \sqsubseteq \widehat{\mathsf{col}}(\mathsf{Plays}(\mathcal{A}, s, \sigma_1, \sigma'_2)). \tag{1}$$

Similarly to optimal strategies, we call an NE *uniform* if it is an NE from all states $s \in S$.

The connection between *optimal strategies* and *Nash equilibria* in our specific context of *antagonistic* games is interesting to discuss, especially with respect to Gimbert and Zielonka's original work [26]. We defer this discussion to [6] due to space constraints, and only provide here a brief account of the results one has to know to understand this overview. First, NE are de facto pairs of optimal strategies. Second, it is possible to mix different NE.

▶ **Lemma 2.** *Let $\mathcal{G} = (\mathcal{A}, \sqsubseteq)$ be a game on arena $\mathcal{A} = (S_1, S_2, E)$, and let $s \in S$ be a state. Let (σ_1^a, σ_2^a) and $(\sigma_1^b, \sigma_2^b) \in \Sigma_1(\mathcal{A}) \times \Sigma_2(\mathcal{A})$ be two Nash equilibria from s. Then, (σ_1^a, σ_2^b) is also a Nash equilibrium from s.*

▶ Remark 3. Lemma 2 crucially relies on the assumption (transparent in our definition of Nash equilibrium) that we consider *antagonistic* games, that is, $\mathcal{P}_2$ uses the inverse preference relation $\sqsubseteq^{-1}$.

3 Concepts

Generalizing monotony and selectivity. As seen in Section 1, Gimbert and Zielonka's characterization [26] relies on *monotony* and *selectivity* of the preference relation. The main difference between their technical approach and ours is the following. In the memoryless setting, all the reasoning can be *abstracted away* from the underlying arena and done on sequences of colors. In the finite-memory one, however, one has to pay attention to how sequences of colors are composed and compared, to maintain consistency with regard to the memory and the game arena. This need to intertwine abstract reasoning on arbitrary sequences of colors with concrete tracking of memory updates is the key obstacle to overcome.

Much of our effort was thus spent on trying to define concepts that would preserve the elegance of monotony and selectivity while allowing us to lift the theory to the finite-memory case. As often the case, the good concepts turned out to be the most natural ones, capturing the intuitive idea that one needs monotony and selectivity *modulo a memory skeleton*.

▶ **Definition 4** ($\mathcal{M}$-monotony). *Let $\mathcal{M} = (M, m_{\mathsf{init}}, \alpha_{\mathsf{upd}})$ be a memory skeleton. A preference relation $\sqsubseteq$ is* $\mathcal{M}$-monotone *if for all $m \in M$, for all $K_1, K_2 \in \mathcal{R}(C)$,*

$$(\exists\, w \in L_{m_{\mathsf{init}},m},\ [wK_1] \sqsubset [wK_2]) \implies (\forall\, w' \in L_{m_{\mathsf{init}},m},\ [w'K_1] \sqsubseteq [w'K_2])\,. \tag{2}$$

Recall that a skeleton $\mathcal{M}$ has a fixed initial state m_{init}. Intuitively, *$\mathcal{M}$-monotony* extends Gimbert and Zielonka's monotony by comparing prefixes *belonging to the same language* $L_{m_{\mathsf{init}},m}$, that is, prefixes that are deemed equivalent by skeleton $\mathcal{M}$. This property roughly captures that $\sqsubseteq$ is *stable with regard to prefix addition*, for memory-equivalent prefixes.

The original monotony notion is equivalent to our $\mathcal{M}$-monotony with $\mathcal{M}$ being the trivial skeleton $\mathcal{M}_{\mathsf{triv}}$: that is, the memoryless case is naturally a subcase of our framework.

▶ **Definition 5** ($\mathcal{M}$-selectivity). *Let $\mathcal{M} = (M, m_{\mathsf{init}}, \alpha_{\mathsf{upd}})$ be a memory skeleton. A preference relation $\sqsubseteq$ is* $\mathcal{M}$-selective *if for all $w \in C^*$, $m = \widehat{\alpha_{\mathsf{upd}}}(m_{\mathsf{init}}, w)$, for all $K_1, K_2 \in \mathcal{R}(C)$ such that $K_1, K_2 \subseteq L_{m,m}$, for all $K_3 \in \mathcal{R}(C)$,*

$$[w(K_1 \cup K_2)^* K_3] \sqsubseteq [wK_1^*] \cup [wK_2^*] \cup [wK_3]. \tag{3}$$

Similarly, *$\mathcal{M}$-selectivity* extends Gimbert and Zielonka's selectivity by asking one to compare sequences of colors *belonging to the same language* $L_{m,m}$, that is, sequences read as cycles on the memory skeleton. Note also that the memory state m should be consistent with the prefix w read from the initial memory state m_{init}. This property roughly captures that $\sqsubseteq$ is *stable with regard to cycle mixing*, for memory-equivalent cycles.

Again, the original selectivity notion is exactly equivalent to $\mathcal{M}_{\mathsf{triv}}$-selectivity.

In a nutshell, $\mathcal{M}$-monotony deals with prefixes up to the first cycle (on memory) and $\mathcal{M}$-selectivity deals with the cycles thereafter; we will see that memory skeletons can be built in a compositional way based on these two orthogonal yet complementary tasks.

Our notions respect the natural intuition that access to additional memory should always be helpful: if a skeleton $\mathcal{M}$ is sufficient to classify sequences of colors in a way that guarantees $\mathcal{M}$-monotony and $\mathcal{M}$-selectivity, then it should also be the case for "more powerful" skeletons.

▶ **Lemma 6.** *Let $\mathcal{M}$ and $\mathcal{M}'$ be two memory skeletons. If $\sqsubseteq$ is $\mathcal{M}$-monotone (resp. $\mathcal{M}$-selective) then, it is also $(\mathcal{M} \otimes \mathcal{M}')$-monotone (resp. $(\mathcal{M} \otimes \mathcal{M}')$-selective).*

Prefix-covers and cyclic-covers. While the concepts of $\mathcal{M}$-monotony and $\mathcal{M}$-selectivity are the primordial ones for stating the characterization, we still need two additional notions to prove it. Let us sketch the issue. To prove that monotone and selective preference relations yield UML strategies, Gimbert and Zielonka deploy an *inductive argument* on the number of choices in an arena. Intuitively, we want to use a similar approach for UFM strategies, but because of the unavoidable coupling between the memory skeleton and the arena (e.g., Lemma 1), the induction argument breaks, as adding one choice in the arena results in adding many in the *product arena* (as many as there are memory states), where the reasoning needs to take place. New insight and techniques are thus needed to patch this scheme.

To solve this issue, *we decouple the two aspects*. We first establish that, on arenas that inherently share the same good properties as product arenas (i.e., they already "classify" prefixes and cycles as the memory would), we can deploy the induction argument and obtain UML strategies. Then, we obtain UFM strategies on *general* arenas as a corollary. The crux is identifying such "good" arenas: this is done through the following notions.

▶ **Definition 7** (Prefix-covers and cyclic-covers). *Let $\mathcal{M} = (M, m_{\mathsf{init}}, \alpha_{\mathsf{upd}})$ be a memory skeleton and $\mathcal{A} = (S_1, S_2, E)$ be an arena. Let $S_{\mathsf{cov}} \subseteq S$.*

We say that $\mathcal{M}$ is a prefix-cover of S_{cov} in $\mathcal{A}$ if for all $s \in S$, there exists $m_s \in M$ such that, for all $\rho \in \mathsf{Hists}(\mathcal{A})$ such that $\mathsf{in}(\rho) \in S_{\mathsf{cov}}$, $\mathsf{out}(\rho) = s$ and such that for all ρ' proper prefix of ρ, $\mathsf{out}(\rho') \neq s$, we have $\widehat{\alpha_{\mathsf{upd}}}(m_{\mathsf{init}}, \widehat{\mathsf{col}}(\rho)) = m_s$.

We say that $\mathcal{M}$ is a cyclic-cover of S_{cov} in $\mathcal{A}$ if for all $\rho \in \mathsf{Hists}(\mathcal{A})$ such that $\mathsf{in}(\rho) \in S_{\mathsf{cov}}$, if $s = \mathsf{out}(\rho)$ and $m = \widehat{\alpha_{\mathsf{upd}}}(m_{\mathsf{init}}, \widehat{\mathsf{col}}(\rho))$, for all $\rho' \in \mathsf{Hists}(\mathcal{A})$ such that $\mathsf{in}(\rho') = \mathsf{out}(\rho') = s$, $\widehat{\alpha_{\mathsf{upd}}}(m, \widehat{\mathsf{col}}(\rho')) = m$.

Intuitively, $\mathcal{M}$ is a prefix-cover for a set of states S_{cov} if the histories starting in S_{cov} and visiting a given state $s \in S$ for the first time are read up to the same memory state in the memory skeleton. Similarly, $\mathcal{M}$ is a cyclic-cover of $\mathcal{A}$ if the cycles of $\mathcal{A}$ are read as cycles in the memory skeleton, once the memory has been initialized properly.

As hinted above, the canonical example of a prefix- and cyclic-covered arena is a product arena (but many more may be in this case; it is beneficial to be general with these concepts).

▶ **Lemma 8.** *Let $\mathcal{M} = (M, m_{\mathsf{init}}, \alpha_{\mathsf{upd}})$ be a memory skeleton and $\mathcal{A} = (S_1, S_2, E)$ be an arena. Then $\mathcal{M}$ is a prefix- and cyclic-cover for $S_{\mathsf{cov}} = S \times \{m_{\mathsf{init}}\}$ in $\mathcal{A} \ltimes \mathcal{M}$.*

4 Characterization

Equivalence. We now have the necessary ingredients to state our equivalence result.

▶ **Theorem 9** (Equivalence). *Let $\sqsubseteq$ be a preference relation and let $\mathcal{M}$ be a memory skeleton. Then, both players have UFM strategies based on memory skeleton $\mathcal{M}$ in all games $\mathcal{G} = (\mathcal{A}, \sqsubseteq)$ if and only if $\sqsubseteq$ and $\sqsubseteq^{-1}$ are $\mathcal{M}$-monotone and $\mathcal{M}$-selective.*

We state this theorem broadly and with a *focus on UFM strategies*. The actual results we have for each direction of the equivalence – see [6, Section 4 and Section 5] – are a bit stronger, of wider applicability and/or more interesting, but this statement carries the take-home message of our work. It is also meant to mirror the seminal result of Gimbert and Zielonka [26, Theorem 2]: their result can be retrieved from Theorem 9 by taking the trivial memory skeleton $\mathcal{M}_{\mathsf{triv}}$. As such, our work brings a *strict generalization of Gimbert and Zielonka's results* [26] to the finite-memory case.

Lifting corollary. As discussed in Section 1, the work of Gimbert and Zielonka contains not one, but *two* great results. Alongside the aforementioned equivalence result, Gimbert and Zielonka provide a corollary of high practical interest [26, Corollary 7]: they essentially obtain as a by-product of their approach that if memoryless strategies suffice in all one-player games of $\mathcal{P}_1$ and all one-player games of $\mathcal{P}_2$, they also suffice in all two-player games.

This provides an elegant way to prove that a preference relation (equivalently, an objective) admits memoryless optimal strategies *without proving monotony and selectivity at all*: proving it in the two one-player subcases, which is generally much easier as it boils down to graph reasoning, and then lifting the result to the general two-player case through the corollary. See examples of one-player vs. two-player complexity in [5, 4, 11].

Again, we are able to lift this corollary to the arena-independent finite-memory case.

▶ **Corollary 10.** *Let $\sqsubseteq$ be a preference relation and $\mathcal{M}_1, \mathcal{M}_2$ be two memory skeletons. Assume that*

1. *for all one-player arenas $\mathcal{A} = (S_1, S_2 = \emptyset, E)$, $\mathcal{P}_1$ has a UFM strategy $\sigma_1 \in \Sigma_1^{\mathsf{FM}}(\mathcal{A})$ based on memory skeleton $\mathcal{M}_1$ in $\mathcal{G} = (\mathcal{A}, \sqsubseteq)$;*
2. *for all one-player arenas $\mathcal{A} = (S_1 = \emptyset, S_2, E)$, $\mathcal{P}_2$ has a UFM strategy $\sigma_2 \in \Sigma_2^{\mathsf{FM}}(\mathcal{A})$ based on memory skeleton $\mathcal{M}_2$ in $\mathcal{G} = (\mathcal{A}, \sqsubseteq)$.*

Then, for all two-player arenas $\mathcal{A} = (S_1, S_2, E)$, both $\mathcal{P}_1$ and $\mathcal{P}_2$ have UFM strategies $\sigma_i \in \Sigma_i^{\mathsf{FM}}(\mathcal{A})$ based on memory skeleton $\mathcal{M} = \mathcal{M}_1 \otimes \mathcal{M}_2$ in $\mathcal{G} = (\mathcal{A}, \sqsubseteq)$.

We highlight the two (possibly different) skeletons of the two players to maintain a compositional approach, but if the same skeleton $\mathcal{M}$ works in both one-player versions, it also suffices in the two-player version.

5 Example of application

We present an illustrative application, thereby proving the existence of UFM strategies for a specific preference relation: the conjunction of two reachability objectives, a subcase of *generalized reachability games*, studied extensively in [21]. Let C be an arbitrary set of colors, and $T_1, T_2 \subseteq C$ be two target sets of colors that have to be visited. Formally, let $W \subseteq C^\omega$ be the set of words $w = c_1 c_2 \dots$ such that $\exists\, i, j \in \mathbb{N},\ c_i \in T_1 \wedge c_j \in T_2$. This winning condition induces a two-level (i.e., win/lose) preference relation $\sqsubseteq$.

In this example, we will use Theorem 9 directly in order to provide one thorough illustration of the definitions of $\mathcal{M}$-monotony and $\mathcal{M}$-selectivity. However, in practice, using Corollary 10 is preferable, as it yields a much shorter proof: by exhibiting the right skeletons for $\mathcal{P}_1$ and $\mathcal{P}_2$, we simply have to show that these skeletons are sufficient to play optimally on both players' *one-player arenas*, which amounts to graph reasoning.

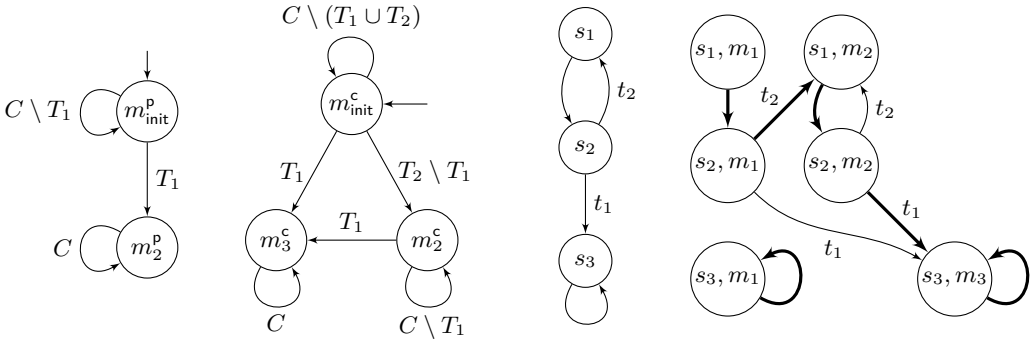


Figure 2 First and second: memory skeletons $\mathcal{M}^{\mathsf{p}}$ and $\mathcal{M}^{\mathsf{c}}$ for two-target reachability games; third: arena $\mathcal{A}$; fourth: product arena $\mathcal{A} \ltimes \mathcal{M}$ (only states reachable from $S \times \{m_{\mathsf{init}}\}$ are depicted). We assume that $T_1 = \{t_1\}$, $T_2 = \{t_2\}$. The $(S \times \{m_{\mathsf{init}}\})$-optimal memoryless strategy is in bold.

We start by showing that this preference relation is not $\mathcal{M}_{\mathsf{triv}}$-monotone (that is, is not monotone for [26]). Assume $c_1 \in T_1 \setminus T_2$, $c_2 \in T_2 \setminus T_1$, and $c_3 \notin T_1 \cup T_2$. Take $K_1 = c_1^*$, $K_2 = c_2^*$. For $w = c_1$, $w' = c_2$, we have $[wK_1] \sqsubset [wK_2]$, but $[w'K_2] \sqsubset [w'K_1]$. This means that the preference relation is not stable with regard to prefix addition (at least, without distinguishing different classes of prefixes). Similarly, it is not $\mathcal{M}_{\mathsf{triv}}$-selective (take w as the empty word, $K_1 = c_1^*$, $K_2 = c_2^*$, $K_3 = c_3^*$: to win, K_1 and K_2 need to be mixed).

In Figure 2, we exhibit skeletons $\mathcal{M}^{\mathsf{p}} = (M^{\mathsf{p}}, m^{\mathsf{p}}_{\mathsf{init}}, \alpha^{\mathsf{p}}_{\mathsf{upd}})$ and $\mathcal{M}^{\mathsf{c}} = (M^{\mathsf{c}}, m^{\mathsf{c}}_{\mathsf{init}}, \alpha^{\mathsf{c}}_{\mathsf{upd}})$ such that $\sqsubseteq$ is $\mathcal{M}^{\mathsf{p}}$-monotone and $\mathcal{M}^{\mathsf{c}}$-selective. Note that such skeletons are obviously not unique.

Let us prove that $\sqsubseteq$ is $\mathcal{M}^{\mathsf{p}}$-monotone. Let $m \in M^{\mathsf{p}}$, $K_1, K_2 \in \mathcal{R}(C)$; we want to show that Equation (2) is satisfied. We assume that there exists $w \in L_{m^{\mathsf{p}}_{\mathsf{init}}, m}$ such that $[wK_1] \sqsubset [wK_2]$: this means that all words of $[wK_1]$ are losing, and that there exists a winning word in $[wK_2]$. Let $w' \in L_{m^{\mathsf{p}}_{\mathsf{init}}, m}$; we show that we necessarily have that $[w'K_1] \sqsubseteq [w'K_2]$. Note that if $[K_1]$ is empty, this always holds; we now assume that $[K_1]$ is non-empty. We study the two possible values of m separately.

- If $m = m^{\mathsf{p}}_{\mathsf{init}}$, then w and w' do not reach T_1. If w does not reach T_2 either, as there is a winning word in $[wK_2]$, then there must be a winning word in $[K_2]$. This word is still winning after prepending w' to it, so there is a winning word in $[w'K_2]$, and $[w'K_1] \sqsubseteq [w'K_2]$. If w reaches T_2, then $[K_1]$ cannot have a word reaching T_1. As w' does not reach T_1 either, all words of $[w'K_1]$ are losing, so $[w'K_1] \sqsubseteq [w'K_2]$.
- If $m = m^{\mathsf{p}}_2$, then w and w' reach T_1. Clearly, w cannot reach T_2 (as $[wK_1]$ would be winning). This implies that $[K_2]$ must contain a word reaching T_2; as w' reaches T_1, the concatenation of w' with the word of $[K_2]$ reaching T_2 means that there is a winning word in $[w'K_2]$, so $[w'K_1] \sqsubseteq [w'K_2]$.

Let us now prove that $\sqsubseteq$ is $\mathcal{M}^{\mathsf{c}}$-selective. Let $w \in C^*$, $m = \widehat{\alpha^{\mathsf{c}}_{\mathsf{upd}}}(m^{\mathsf{c}}_{\mathsf{init}}, w)$, $K_1, K_2 \in \mathcal{R}(C)$ such that $K_1, K_2 \subseteq L_{m,m}$, and $K_3 \in \mathcal{R}(C)$. We show that Equation (3) is satisfied, i.e., that $[w(K_1 \cup K_2)^* K_3] \sqsubseteq [wK_1^*] \cup [wK_2^*] \cup [wK_3]$. If all words of $[w(K_1 \cup K_2)^* K_3]$ are losing, this equation trivially holds; we thus assume that this set contains a winning word. We therefore have to show that there is a winning word in $[wK_1^*]$, $[wK_2^*]$, or $[wK_3]$. We study the three possible values of m separately.

- If $m = m^{\mathsf{c}}_{\mathsf{init}}$, then w does not reach T_1 nor T_2, and the same holds for all words of K_1 and K_2, as $K_1, K_2 \subseteq L_{m^{\mathsf{c}}_{\mathsf{init}}, m^{\mathsf{c}}_{\mathsf{init}}}$. Therefore, if a word of $[w(K_1 \cup K_2)^* K_3]$ is winning, this must be because a word of $[wK_3]$ is winning.
- If $m = m^{\mathsf{c}}_2$, then w reaches T_2 but not T_1, and K_1, K_2 do not reach T_1. Thus, a word of $[K_3]$ must reach T_1; in particular, a word of $[wK_3]$ must reach both T_1 and T_2.
- If $m = m^{\mathsf{c}}_3$, we distinguish two cases. If w reaches T_2 and T_1, then $[wK_1^*] \cup [wK_2^*] \cup [wK_3]$ trivially contains only winning words. If w reaches T_1 but not T_2, then there must be a word reaching T_2 in $[(K_1 \cup K_2)^* K_3]$. Hence, at least one set among $[K_1^*]$, $[K_2^*]$, and $[K_3]$ must contain a word reaching T_2, so $[wK_1^*]$, $[wK_2^*]$, or $[wK_3]$ contains a winning word.

Similar arguments can be laid out to show that the preference relation $\sqsubseteq^{-1}$ of $\mathcal{P}_2$ is $\mathcal{M}^{\mathsf{p}}$-monotone and $\mathcal{M}_{\mathsf{triv}}$-selective (where $\mathcal{M}_{\mathsf{triv}}$ is the trivial memory skeleton defined earlier). Let $\mathcal{M} = \mathcal{M}^{\mathsf{p}} \otimes \mathcal{M}^{\mathsf{c}} \otimes \mathcal{M}_{\mathsf{triv}}$ be the product of all the considered skeletons. Although $\mathcal{M}$ formally has six states, its only reachable part is isometric to skeleton $\mathcal{M}^{\mathsf{c}}$, with $m_1 \leftrightarrow m^{\mathsf{c}}_{\mathsf{init}}$ as initial state, $m_2 \leftrightarrow m^{\mathsf{c}}_2$, and $m_3 \leftrightarrow m^{\mathsf{c}}_3$: we thus do not depict it to save space.

By Lemma 6, we have that both $\sqsubseteq$ and $\sqsubseteq^{-1}$ are $\mathcal{M}$-monotone and $\mathcal{M}$-selective. Using Theorem 9, we obtain that *both players have UFM strategies based on skeleton $\mathcal{M}$ in all games $\mathcal{G} = (\mathcal{A}, \sqsubseteq)$*. Note that memory skeleton $\mathcal{M}$ is minimal (no memory skeleton with two states or less suffices for $\mathcal{P}_1$ to play optimally in all arenas [21]).

We provide an example of a one-player arena $\mathcal{A} = (S_1, S_2 = \emptyset, E)$ in Figure 2, and show that there is a UFM strategy for the preference relation $\sqsubseteq$ based on skeleton $\mathcal{M}$. To do so, we invoke Lemma 1: we show equivalently that the product $\mathcal{A} \ltimes \mathcal{M}$ admits an $(S \times \{m_{\mathsf{init}}\})$-optimal memoryless strategy for $\sqsubseteq$. Notice that no memoryless strategy suffices to play optimally in $\mathcal{G} = (\mathcal{A}, \sqsubseteq)$, as when starting in s_2, $\mathcal{P}_1$ should first visit s_1 before going to s_3. Also, the $(S \times \{m_{\mathsf{init}}\})$-optimal memoryless strategy for the product arena is only optimal if the initial state is in $S \times \{m_{\mathsf{init}}\}$; it is for instance not optimal from state (s_2, m_2).

6 Technical sketch

Due to space constraints, we only sketch our proof schemes here: full proofs are in [6].

From finite memory based on $\mathcal{M}$ to $\mathcal{M}$-monotony and $\mathcal{M}$-selectivity. For the left-to-right implication of Theorem 9, it suffices to consider the weaker assumption involving only one-player: we establish that if UFM strategies based on $\mathcal{M}$ exist in all *one-player games* of

$\mathcal{P}_1$ (resp. $\mathcal{P}_2$), then his preference relation $\sqsubseteq$ (resp. $\sqsubseteq^{-1}$) is $\mathcal{M}$-monotone and $\mathcal{M}$-selective. To maintain a compositional approach, we consider $\mathcal{M}$-monotony and $\mathcal{M}$-selectivity separately. Details are in [6, Section 4].

Let us sketch the proof for $\mathcal{M}$-monotony and $\mathcal{P}_1$. We need to establish Equation (2). We instantiate the four languages involved in it: $\{w\}$, $\{w'\}$, K_1 and K_2. We take NFA recognizing them and build an NFA $\mathcal{N}$ that joins them in such a way that, when $\mathcal{N}$ is considered as a one-player game arena, its plays correspond exactly to the languages of infinite words considered in Equation (2). This arena is composed of two chains emulating the two prefixes w and w' and leading to a state t where $\mathcal{P}_1$ has to pick a side corresponding to the two languages $[K_1]$ and $[K_2]$. Now, proving the $\mathcal{M}$-monotony of $\sqsubseteq$ boils down to invoking an optimal strategy σ in the corresponding game, the crux being that σ always picks the same edge in t (i.e., the same side between subarenas corresponding to $[K_1]$ and $[K_2]$) as both prefixes w and w' are deemed equivalent by the memory skeleton $\mathcal{M}$.

The proof for $\mathcal{M}$-selectivity is similar. The main difference is that the "joining" state t can be visited repeatedly in this case – possibly infinitely often. This is because Equation (3) is about cycles and their languages. Our proof takes that into account.

From $\mathcal{M}$-monotony and $\mathcal{M}$-selectivity to finite memory based on $\mathcal{M}$. The right-to-left implication of Theorem 9 is more complex to establish. In this case, we want the result for *two-player games*, provided both preference relations are $\mathcal{M}$-monotone and $\mathcal{M}$-selective. The general scheme we use is an induction on the number of choices in arenas. The main issue is dealing with the memory: one additional choice in an arena results in many ones in the corresponding product arena. To circumvent this obstacle, we proceed in two steps. Details are in [6, Section 5].

We first establish the existence of *UML strategies in (prefix- and cyclic-)covered arenas*. Let us focus on the induction step we use to prove this result, as an example of the techniques involved. For an arena $\mathcal{A} = (S_1, S_2, E)$, we write $n_{\mathcal{A}} = |E| - |S|$ for its number of choices. To simplify, let us say we have a skeleton $\mathcal{M}$ such that $\sqsubseteq$ is $\mathcal{M}$-monotone and $\mathcal{M}$-selective, and that we assume that memoryless – *for the two players* – NE exist from all covered states in arenas with less than n choices. Then we establish that we can also build an NE in arenas with n choices, in which $\mathcal{P}_1$ uses a memoryless strategy – but maybe not $\mathcal{P}_2$! To prove this, we proceed as follows.

Let $\mathcal{A}$ be an arena with $n_{\mathcal{A}} = n$ choices. We identify a state t in which $\mathcal{P}_1$ has at least two outgoing edges. By splitting the edges in t in two sets, we obtain two corresponding subarenas $\mathcal{A}_a$ and $\mathcal{A}_b$ such that $n_{\mathcal{A}_a}, n_{\mathcal{A}_b} < n$, along with the corresponding subgames. The induction hypothesis gives us two memoryless NE (from covered states) in these subgames: (σ_1^a, σ_2^a) and (σ_1^b, σ_2^b). The arguments can then be unfolded as follows. First, using $\mathcal{M}$-monotony and $\mathcal{M}$ being a prefix-cover, we identify one subarena (say $\mathcal{A}_a$) which is clearly at least as good as the other for $\mathcal{P}_1$. Second, we build a strategy profile $(\sigma_1^\#, \sigma_2^\#)$, that we claim to be an NE in $\mathcal{G} = (\mathcal{A}, \sqsubseteq)$, in the following way: $\mathcal{P}_1$ uses strategy σ_1^a (the one from the best subarena) and $\mathcal{P}_2$ reacts to $\mathcal{P}_1$'s actions by playing the corresponding best-response strategy. I.e., if $\mathcal{P}_1$ plays in $\mathcal{A}_a$, $\mathcal{P}_2$ plays according to σ_2^a, and otherwise he plays according to σ_2^b. Third, it remains to prove the two inequalities of Equation (1). The rightmost one is easy, as well as the leftmost one in the subcase where the unique play $\pi \in \mathsf{Plays}(\mathcal{A}, s, \sigma_1^\#, \sigma_2^\#)$ does not visit state t: they can both be proved thanks to the induction hypothesis and easy construction arguments. The crux of the proof is thus in the last step: proving that the leftmost inequality holds when the play visits t. This can be achieved thanks to $\mathcal{M}$-selectivity and $\mathcal{M}$ being a cyclic-cover, properties of the union operator in languages of prefixes, inherent properties of the preference relation, $\mathcal{A}_a$ being the best subarena thanks $\mathcal{M}$-monotony, and the induction hypothesis, in that order.

The actual induction step and its proof are more subtle – for example, we use different skeletons for monotony and selectivity and obtain the result in a compositional way; but the main intuition is carried here. The same result can be established symmetrically for $\mathcal{P}_2$, but again the resulting NE is only memoryless for $\mathcal{P}_2$. Yet, assuming both $\sqsubseteq$ and $\sqsubseteq^{-1}$ are $\mathcal{M}$-monotone and $\mathcal{M}$-selective, we have two half-memoryless NE that we can mix to obtain a truly memoryless NE via Lemma 2; thus proving the existence of UML strategies in covered arenas. Observe this interesting by-product of our approach: we can actually detect *arenas where memory is not needed at all* thanks to our concepts of prefix- and cyclic-covers.

The final result – the existence of UFM strategies based on $\mathcal{M}$ in all arenas – can then be obtained as a corollary, based on the link between memoryless strategies in product arenas and memoryfull ones in original arenas (Lemma 1). Another nice by-product of our approach, witnessed in Corollary 10, is that *the product of individual memories from one-player games is sufficient to play optimally in two-player games*, for both players. This is in stark contrast to the counter-example discussed in Section 1 and it illustrates that our characterization matches well-behaved preference relations.

Equivalence and lifting corollary. The equivalence (Theorem 9) is easily obtained by putting together its two directions. Note that we also establish a similar equivalence in the one-player case as a by-product.

▶ **Theorem 11** (One-player equivalence). *Let $\sqsubseteq$ be a preference relation and let $\mathcal{M}$ be a memory skeleton. Then, $\mathcal{P}_1$ has UFM strategies based on memory skeleton $\mathcal{M}$ in all his one-player games $\mathcal{G} = (\mathcal{A}, \sqsubseteq)$ if and only if $\sqsubseteq$ is $\mathcal{M}$-monotone and $\mathcal{M}$-selective.*

Although this looks like a weak version of Theorem 9 at first sight, this is actually a distinct result as both sides of the equivalence are weaker: on the left side, it only handles the memory requirements for $\mathcal{P}_1$'s one-player games; on the right side, it does not assume anything about the inverse preference relation $\sqsubseteq^{-1}$.

The lifting corollary, Corollary 10, is also a consequence of our approach. As we have seen, the existence of UFM strategies based on a skeleton $\mathcal{M}$ in *one-player* games suffices to yield $\mathcal{M}$-monotony and $\mathcal{M}$-selectivity of the corresponding preference relation. Hence if both players have UFM strategies in their one-player games, both relations satisfy these properties and we can take the other direction of Theorem 9 to ensure that UFM strategies also exist in *two-player* games. As explained above, this approach can actually be used compositionally.

All proofs, as well as the one-player equivalence, are presented in details in [6].

References

1 Benjamin Aminof and Sasha Rubin. First-cycle games. *Inf. Comput.*, 254:195–216, 2017. doi:10.1016/j.ic.2016.10.008.

2 Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. Graph games and reactive synthesis. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 921–962. Springer, 2018. doi:10.1007/978-3-319-10575-8_27.

3 Patricia Bouyer, Ulrich Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey, and Jirí Srba. Infinite runs in weighted timed automata with energy constraints. In Franck Cassez and Claude Jard, editors, *Formal Modeling and Analysis of Timed Systems, 6th International Conference, FORMATS 2008, Saint Malo, France, September 15-17, 2008. Proceedings*, volume 5215 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2008. doi:10.1007/978-3-540-85778-5_4.

4 Patricia Bouyer, Piotr Hofman, Nicolas Markey, Mickael Randour, and Martin Zimmermann. Bounding average-energy games. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 179–195, 2017. doi:10.1007/978-3-662-54458-7_11.

5 Patricia Bouyer, Nicolas Markey, Mickael Randour, Kim G. Larsen, and Simon Laursen. Average-energy games. *Acta Inf.*, 55(2):91–127, 2018. doi:10.1007/s00236-016-0274-1.

6 Patricia Bouyer, Stéphane Le Roux, Youssouf Oualhadj, Mickael Randour, and Pierre Vandenhove. Games where you can play optimally with arena-independent finite memory. *CoRR*, abs/2001.03894, 2020. arXiv:2001.03894.

7 Romain Brenguier, Lorenzo Clemente, Paul Hunter, Guillermo A. Pérez, Mickael Randour, Jean-François Raskin, Ocan Sankur, and Mathieu Sassolas. Non-zero sum games for reactive synthesis. In Adrian-Horia Dediu, Jan Janousek, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications - 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings*, volume 9618 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2016. doi:10.1007/978-3-319-30000-9_1.

8 Thomas Brihaye, Florent Delgrange, Youssouf Oualhadj, and Mickael Randour. Life is random, time is not: Markov decision processes with window objectives. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.8.

9 Véronique Bruyère, Emmanuel Filiot, Mickael Randour, and Jean-François Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. *Inf. Comput.*, 254:259–295, 2017. doi:10.1016/j.ic.2016.10.011.

10 Véronique Bruyère, Quentin Hautem, and Mickael Randour. Window parity games: an alternative approach toward parity games with time bounds. In Domenico Cantone and Giorgio Delzanno, editors, *Proceedings of the Seventh International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2016, Catania, Italy, 14-16 September 2016*, volume 226 of *EPTCS*, pages 135–148, 2016. doi:10.4204/EPTCS.226.10.

11 Véronique Bruyère, Quentin Hautem, Mickael Randour, and Jean-François Raskin. Energy mean-payoff games. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 21:1–21:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.21.

12 Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource interfaces. In Rajeev Alur and Insup Lee, editors, *EMSOFT*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2003. doi:10.1007/978-3-540-45212-6_9.

13 Krishnendu Chatterjee and Laurent Doyen. Energy parity games. *Theor. Comput. Sci.*, 458:49–60, 2012. doi:10.1016/j.tcs.2012.07.038.

14 Krishnendu Chatterjee, Laurent Doyen, Mickael Randour, and Jean-François Raskin. Looking at mean-payoff and total-payoff through windows. *Inf. Comput.*, 242:25–52, 2015. doi:10.1016/j.ic.2015.03.010.

15 Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdzinski. Mean-payoff parity games. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 178–187. IEEE Computer Society, 2005. doi:10.1109/LICS.2005.26.

16 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Generalized parity games. In Helmut Seidl, editor, *Foundations of Software Science and Computational Structures, 10th International Conference, FOSSACS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007, Braga, Portugal, March 24-April 1, 2007, Proceedings*, volume 4423 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2007. doi:10.1007/978-3-540-71389-0_12.

17 Krishnendu Chatterjee, Mickael Randour, and Jean-François Raskin. Strategy synthesis for multi-dimensional quantitative objectives. *Acta Inf.*, 51(3-4):129–163, 2014. doi:10.1007/s00236-013-0182-6.

18 Florent Delgrange, Joost-Pieter Katoen, Tim Quatmann, and Mickael Randour. Simple strategies in multi-objective MDPs. In Armin Biere and David Parker, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part I*, volume 12078 of *Lecture Notes in Computer Science*, pages 346–364. Springer, 2020. doi:10.1007/978-3-030-45190-5_19.

19 Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *Int. Journal of Game Theory*, 8(2):109–113, 1979.

20 E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. In *FOCS*, pages 328–337. IEEE Computer Society, 1988. doi:10.1109/SFCS.1988.21949.

21 Nathanaël Fijalkow and Florian Horn. The surprizing complexity of reachability games. *CoRR*, abs/1010.2420, 2010. arXiv:1010.2420.

22 Nathanaël Fijalkow, Florian Horn, Denis Kuperberg, and Michal Skrzypczak. Trading bounds for memory in games with counters. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 197–208. Springer, 2015. doi:10.1007/978-3-662-47666-6_16.

23 Hugo Gimbert. Pure stationary optimal strategies in Markov decision processes. In Wolfgang Thomas and Pascal Weil, editors, *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*, volume 4393 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 2007. doi:10.1007/978-3-540-70918-3_18.

24 Hugo Gimbert and Edon Kelmendi. Two-player perfect-information shift-invariant submixing stochastic games are half-positional. Unpublished, 2014.

25 Hugo Gimbert and Wieslaw Zielonka. When can you play positionally? In Jirí Fiala, Václav Koubek, and Jan Kratochvíl, editors, *Mathematical Foundations of Computer Science 2004, 29th International Symposium, MFCS 2004, Prague, Czech Republic, August 22-27, 2004, Proceedings*, volume 3153 of *Lecture Notes in Computer Science*, pages 686–697. Springer, 2004. doi:10.1007/978-3-540-28629-5_53.

26 Hugo Gimbert and Wieslaw Zielonka. Games where you can play optimally without any memory. In Martín Abadi and Luca de Alfaro, editors, *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, volume 3653 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2005. doi:10.1007/11539452_33.

27 Hugo Gimbert and Wieslaw Zielonka. Pure and Stationary Optimal Strategies in Perfect-Information Stochastic Games with Global Preferences. Unpublished, December 2009. URL: https://hal.archives-ouvertes.fr/hal-00438359.

28 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.

29 Marcin Jurdzinski. Deciding the winner in parity games is in UP ∩ co-UP. *Inf. Process. Lett.*, 68(3):119–124, 1998. doi:10.1016/S0020-0190(98)00150-1.

30 Marcin Jurdzinski, Ranko Lazic, and Sylvain Schmitz. Fixed-dimensional energy games are in pseudo-polynomial time. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 2015. doi:10.1007/978-3-662-47666-6_21.

31 Eryk Kopczyński. Half-positional determinacy of infinite games. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 336–347. Springer, 2006. doi:10.1007/11787006_29.

32 Eryk Kopczyński. *Half-positional Determinacy of Infinite Games*. PhD thesis, Warsaw University, 2008.

33 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

34 Martin J. Osborne and Ariel Rubinstein. *A course in game theory*. The MIT Press, Cambridge, USA, 1994.

35 Mickael Randour. Automated synthesis of reliable and efficient systems through game theory: A case study. In *Proc. of ECCS 2012*, Springer Proceedings in Complexity XVII, pages 731–738. Springer, 2013. doi:10.1007/978-3-319-00395-5_90.

36 Stéphane Le Roux. Infinite sequential Nash equilibrium. *Logical Methods in Computer Science*, 9(2), 2013. doi:10.2168/LMCS-9(2:3)2013.

37 Stéphane Le Roux. Concurrent games and semi-random determinacy. In Igor Potapov, Paul G. Spirakis, and James Worrell, editors, *43rd International Symposium on Mathematical Foundations of Computer Science, MFCS 2018, August 27-31, 2018, Liverpool, UK*, volume 117 of *LIPIcs*, pages 40:1–40:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.40.

38 Stéphane Le Roux and Arno Pauly. Extending finite-memory determinacy to multi-player games. *Inf. Comput.*, 261(Part):676–694, 2018. doi:10.1016/j.ic.2018.02.024.

39 Stéphane Le Roux, Arno Pauly, and Mickael Randour. Extending finite-memory determinacy by Boolean combination of winning conditions. In Sumit Ganguly and Paritosh K. Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPIcs*, pages 38:1–38:20. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.38.

40 Yaron Velner, Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, Alexander Moshe Rabinovich, and Jean-François Raskin. The complexity of multi-mean-payoff and multi-energy games. *Inf. Comput.*, 241:177–196, 2015. doi:10.1016/j.ic.2015.03.001.

41 Wieslaw Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theor. Comput. Sci.*, 200(1-2):135–183, 1998. doi:10.1016/S0304-3975(98)00009-7.

A Discussion

We close our paper with a discussion of the assets and limits of our approach, its applicability with regard to the current research landscape, and the directions we aim to follow in future work.

Technical overview. Naturally, our technical approach is inspired by the one of Gimbert and Zielonka for the memoryless case [26], which can actually be rediscovered through our results using a trivial memory skeleton. Two of the most important challenges we had to overcome were:

1. establishing natural concepts of *monotony and selectivity modulo memory* that are exactly as powerful as required to maintain a complete characterization (i.e., sufficient *and* necessary conditions) in the finite-memory case;
2. circumventing the seemingly unavoidable *coupling between the memory skeleton and the arena* in the inductive argument needed to prove the implication from $\mathcal{M}$-monotony and $\mathcal{M}$-selectivity to finite-memory optimal strategies – which we were able to do using our notions of prefix-covers and cyclic-covers.

All along our paper, we highlighted the similarities and discrepancies between our work and Gimbert and Zielonka's [26]. As observed through [6, Remark 16], our results are established using fine-grained assumptions and conclusions, in an effort to push the approach to its limits. They also preserve *compositionality*, splitting the reasoning for $\mathcal{M}$-monotony and $\mathcal{M}$-selectivity, and for the two players.

Alongside $\mathcal{M}$-monotony and $\mathcal{M}$-selectivity, we define two other key concepts to solve the technical issues related to the induction on product arenas: *prefix-covers* and *cyclic-covers*. These notions are crucial tools to prove the right-to-left implication of Theorem 9.

Some advantages. The aforementioned concepts of prefix-covers and cyclic-covers also have benefits from a practical point of view: given a preference relation $\sqsubseteq$ and the corresponding memory skeleton $\mathcal{M}$, they let us *identify game arenas where memoryless strategies suffice* whereas finite memory (based on $\mathcal{M}$) might be necessary in general. Such arenas are the ones covered by $\mathcal{M}$. Hence in practice, this approach permits to obtain UML strategies for many arenas where a coarser approach would only provide UFM ones.

Our approach yields *two methods* to establish that a preference relation (or equivalently a payoff function or a winning condition) admits UFM strategies. The first one, exhibiting appropriate memory skeletons and proving $\mathcal{M}$-monotony and $\mathcal{M}$-selectivity, is based on Theorem 9 and can be used *compositionally* through [6, Corollary 25]. The second one follows the *lifting corollary*, Corollary 10: one only has to study the one-player subcases then invoke this result to lift the existence of UFM strategies to the two-player case, without checking for $\mathcal{M}$-monotony and $\mathcal{M}$-selectivity at all. Hence this second method is often painless in practice.

Two interesting facts can be seen through Corollary 10. First, there is *no blow-up in the memory* required when going from one-player games to two-player games: the overall memory simply combines the memory skeletons of the two players. Second, assuming that one has an algorithm to solve[1] one-player games – say for $\mathcal{P}_1$ – for a winning condition satisfying our hypotheses, this lifting corollary also induces a *naive algorithm for the two-player case for free*: thanks to the bounds on memory, one may enumerate the strategies of the adversary, $\mathcal{P}_2$ – or guess one if one aims for a non-deterministic algorithm – and solve the corresponding $\mathcal{P}_1$'s game(s) where the strategy of $\mathcal{P}_2$ is fixed. Note that while such a simple algorithm might not be optimal, it does correspond to the approach giving the best complexity class known for the renowned family of games in $\mathsf{NP} \cap \mathsf{coNP}$, such as, e.g., parity or mean-payoff games (e.g., [29]). These last two cases could already be dealt with thanks to Gimbert and Zielonka's result since they involve memoryless strategies, but now a similar road can be taken for any objective that admits arena-independent finite-memory optimal strategies, such as, e.g., generalized parity games.

Applicability. Let us give a quick tour of some classical (combinations of) objectives – expressed through winning conditions, payoffs or preference relations – and assess whether our approach permits to establish the existence of UFM strategies in the corresponding games.

Note that when considering multiple (quantitative) objectives, optimal strategies usually do not exist, and one has to settle for *Pareto-optimal* ones (e.g., [18]). However, in many cases, the (decision) problem under study is as follows: given a threshold (vector), define the winning condition as all the plays achieving at least this threshold, and check for a

[1] I.e., decide who has a winning strategy from a given state.

winning strategy. Hence multi-objective quantitative games are often de facto reduced to qualitative win-lose games for this so-called *threshold problem*. Observe that, given a multi-objective setting, if UFM strategies exist for all threshold problems, then finite-memory strategies suffice to realize the Pareto front (as each point of this front can be considered as a threshold). Therefore, *our approach also enables reasoning about the existence of finite-memory Pareto-optimal strategies in multi-objective games.*

We start our overview with some game settings that fall under the scope of our approach. Obviously, *all memoryless-determined objectives* are among them, since we generalize Gimbert and Zielonka's work [26]: this includes, e.g., mean-payoff [19], parity [20, 41], energy [12] or average-energy games [5]. As established in Section 1, our results encompass all cases where *arena-independent* memory suffices. Hence they permit to rediscover the existence of UFM strategies for games such as, e.g., generalized reachability [21], generalized parity [16], window parity games [10], some variants of window mean-payoff games [14], or lower- and upper-bounded (multi-dimension) energy games [3, 5, 4]. Our approach can also be useful to extend these known results to more general combinations, either via appropriate memory skeletons or through the lifting corollary (see an application in Section 5).

There are many games that do not fit our approach for *good reasons*, as they do not admit UFM strategies in general: e.g., multi-dimension mean-payoff [40], mean-payoff parity [15], or energy mean-payoff games [11]. More interesting are games for which finite-memory strategies exist, but the memory is *arena-dependent*. These notably include games with multi-dimension lower-bounded energy objectives and no upper bound [17, 30], or other variants of window mean-payoff games [14]. In such games, the players usually have to keep track of information such as, e.g., the sum of weights along an acyclic path, which is bounded for any given arena, but by a value that grows when the arena grows. Hence the need for memory that grows with the arena parameters. Our results cannot be applied directly to such cases in order to obtain the existence of finite-memory strategies for all games. An adaptation of our approach could potentially be used for subclasses of arenas where the parameters are bounded (in order to regain a skeleton working on all arenas of the class).

Critical analysis. Let us take a step back and assess the place of our work in its larger line of research. The natural endgame is characterizing all preference relations admitting finite-memory optimal strategies, including those using *arena-dependent* memory, and pinpointing the frontiers of application of the lifting corollary – that is, under which conditions is finite-memory determinacy preserved when going from one-player to two-player games?

The road is long from Gimbert and Zielonka's characterization in the memoryless case [26] to such a general result, and this work is but a first step. We have already established that Gimbert and Zielonka's approach cannot be fully transposed for finite memory. Our focus on *arena-independent* memory is a way to study the frontiers of this approach while providing an extension of practical interest. While it may seem limited at first, note that our framework already encompasses arguably rich classes of games such as, e.g., generalized parity games and fully-bounded energy games. As argued in Section 1, recall that our result is in no way a simple application of [26] to product arenas.

From a practical point of view, our equivalence result has limitations as it inherently uses the memory skeleton $\mathcal{M}$. At this point, our approach neither helps in finding an appropriate skeleton, nor in determining the minimal one; two highly interesting questions from a practical standpoint. Nonetheless, to advance toward answering these questions and to be able to find good skeletons automatically, one first has to understand their theoretical characteristics, which we do here as a necessary stepping stone. Focusing on applications, let us note that the equivalence result is often not the most suited tool: this is instead where the lifting corollary shines. As noted before, reasoning on one-player games (i.e., graphs) is

generally much easier than in two-player games (e.g., [5, 4, 11]). Hence, a reasonably easy way to tackle practical cases is to find skeletons sufficient for $\mathcal{P}_1$ and $\mathcal{P}_2$ in their respective one-player games and to use our constructive result to build a skeleton that suffices for both in two-player games.

Comparison with related work. We already discussed the most important related papers [25, 26, 31, 1, 33, 39] in Section 1. Let us highlight here some works where similar approaches have been considered to establish "meta-theorems" applying to general classes of games, or works that inspire interesting directions of research. First and foremost is the determinacy theorem by Martin that guarantees determinacy (without considering the complexity of strategies) for Borel winning conditions [33].

Aminof and Rubin provide a simpler (but incomplete) approach to memoryless determinacy through the prism of first-cycle games in [1]: a similar take on finite-memory determinacy could be appealing – it could provide sufficient conditions easier to test than $\mathcal{M}$-monotony and $\mathcal{M}$-selectivity.

Let us discuss the work of Kopczyński. First, in [31], he establishes sufficient (and relaxed) conditions to ensure the existence of UML strategies *for one player, in two-player games*: it would be interesting to study the corresponding problem in the finite-memory case. Indeed, in many games where infinite memory is needed, it is only the case for one of the players (e.g., [40, 15, 11]) and conditions à la Kopczyński could thus prove useful. Note that this is different from Theorem 11, which gives a sufficient *and necessary* condition but for one-player games only. Second, we recently discovered unpublished content in Kopczyński's PhD thesis [32]. Kopczyński distinguishes *chromatic* memory (which corresponds to our definition of memory skeleton), and the more powerful *chaotic* memory, where transitions can depend on the actual edges of the arenas, rather than simply on their colors. Chaotic memory is thus intrinsically *arena-dependent*. Our notion of an arena being both prefix- and cyclic-covered by a memory skeleton $\mathcal{M}$ is equivalent to a notion in [32, Definition 8.12], which defines that an arena *adheres to chromatic memory* $\mathcal{M}$ if it is possible to assign a state of $\mathcal{M}$ to every state of the arena such that moving along the edges updates these memory states in a consistent way. Our definitions of *prefix-* and *cyclic-cover* can be seen as two distinct sides of this idea of *adherence*, which when added up, are actually equivalent to it.

Following the same motivation as our work – the need to characterize (combinations of) objectives admitting finite-memory optimal strategies, Le Roux et al. [39] take another road: whereas our work permits to lift results from one-player games to two-player games, they provide a lifting from the single-objective case to the multi-objective one. Their techniques, as well as the scope of their results, are somewhat orthogonal to ours. Whether both approaches can be intertwined to obtain results on more general settings remains an open question.

Our work focuses on *deterministic turn-based* two-player games. Sufficient conditions have been published for stochastic models but to the best of our knowledge, no complete characterization, even for the simplest case of Markov decision processes (e.g., [23]). Two unpublished articles contain interesting results on stochastic games [27, 24], including an extension of Gimbert and Zielonka's original work, by the same authors [27]. Whether part of our approach can be useful to tackle the finite-memory case in this context, or in richer contexts mixing games and stochastic models (e.g., [9]) is a question for future research. Some sufficient criteria, orthogonal to our approach, were studied for *concurrent* games in [37].

Limits and future work. To close this paper, we recall three limits of our approach, and the corresponding open problems.

First, as explained throughout the paper, our results cover all cases where *arena-independent* memory suffices, and are *limited to these cases*. We have argued that the approach cannot be fully lifted to the general case, for good reasons, as the lifting corollary breaks in some situations (Section 1). Still, we have hope to generalize our approach to some extent to the *arena-dependent* case, through some *function* associating memory skeletons to arenas, as discussed in Section 1. Obtaining a lifting corollary – under well-chosen conditions – in the arena-dependent case would be of tremendous help in practice: see for example [5, 4, 11]. Hence this is clearly the next step in our quest.

Second, our result is a characterization *instantiated by a memory skeleton* $\mathcal{M}$. While the lifting corollary is helpful in applications, it would be fantastic to be able to find an appropriate skeleton automatically, and to be able to determine if a given skeleton is minimal (with regard to a preference relation). This paper is a first step toward these long-term objectives.

Lastly, as explained in Remark 3 and [6, Remark 24], most of our arguments carry over to the case of *general Nash equilibria*. That is, when considering not necessarily antagonistic games where the two players use different, not necessarily inverse, preference relations. Whether our approach can be adapted in this case, at the price of an unavoidable blow-up of memory, is an open question worth considering. In particular, we want to study the links between our results (including the lifting from one-player to two-player games) and recent results lifting finite-memory determinacy in two-player games to the existence of finite-memory Nash equilibria in multi-player games [38].

Abstraction, Up-To Techniques and Games for Systems of Fixpoint Equations

Paolo Baldan
Università Padova, Italy
baldan@math.unipd.it

Barbara König
Universität Duisburg-Essen, Germany
barbara_koenig@uni-due.de

Tommaso Padoan
Università di Padova, Italy
padoan@math.unipd.it

Abstract

Systems of fixpoint equations over complete lattices, consisting of (mixed) least and greatest fixpoint equations, allow one to express many verification tasks such as model-checking of various kinds of specification logics or the check of coinductive behavioural equivalences. In this paper we develop a theory of approximation for systems of fixpoint equations in the style of abstract interpretation: a system over some concrete domain is abstracted to a system in a suitable abstract domain, with conditions ensuring that the abstract solution represents a sound/complete overapproximation of the concrete solution. Interestingly, up-to techniques, a classical approach used in coinductive settings to obtain easier or feasible proofs, can be interpreted as abstractions in a way that they naturally fit into our framework and extend to systems of equations. Additionally, relying on the approximation theory, we can characterise the solution of systems of fixpoint equations over complete lattices in terms of a suitable parity game, generalising some recent work that was restricted to continuous lattices. The game view opens the way for the development of local algorithms for characterising the solution of such equation systems and we explore some special cases.

2012 ACM Subject Classification Theory of computation → Verification by model checking; Software and its engineering → Model checking

Keywords and phrases fixpoint equation systems, complete lattices, parity games, abstract interpretation, up-to techniques, μ-calculus, bisimilarity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.25

Related Version A full version of the paper is available as [3], `https://arxiv.org/abs/2003.08877`.

Funding Supported by the PRIN Project Analysis of Program Analyses (ASPRA), the University of Padova project ASTA and the DFG projects BEMEGA and SpeQt.

1 Introduction

Systems of fixpoint equations over complete lattices, consisting of (mixed) least and greatest fixpoint equations, allow one to uniformly express many verification tasks. Notable examples come from the area of model-checking. Invariant/safety properties can be characterised as greatest fixpoints, while liveness/reachability properties as least fixpoints. Using both least and greatest fixpoints leads to expressive specification logics. The μ-calculus [27] is a prototypical example, encompassing various other logics such as LTL and CTL. Another area of special interest for the present paper is that of behavioural equivalences, which typically arise as solutions of greatest fixpoint equations (see, e.g., [38]).

In the first part of the paper we develop a theory of approximation for systems of equations in the style of abstract interpretation. The general idea of abstract interpretation [13, 14] consists of extracting properties of programs by defining an approximated program semantics over a so-called abstract domain, usually a complete lattice. Concrete and abstract semantics are typically expressed in terms of (systems of) least fixpoint equations, with conditions ensuring that the approximation obtained is sound, i.e., that properties derived from the abstract semantics are also valid at the concrete level. In an ideal situation also the converse holds and the abstract interpretation is called complete (see e.g. [18]). Abstract interpretation has been applied also for the model checking of various kinds of mu-calculi and temporal logics (see, e.g., [19, 30, 15, 40, 17, 28]).

We generalise this idea to systems of fixpoint equations, where least and greatest fixpoints can coexist (§4). A system over some concrete domain C is abstracted by a system over some abstract domain A. Suitable conditions are identified that ensure the soundness and completeness of the approximation. This enables the use of the approximation theory on a number of verification tasks. We show how to recover some results on property preserving abstractions for the μ-calculus [30]. We also discuss a fixpoint extension of Łukasiewicz logic, considered in [34] as a precursor to model-checking PCTL or probabilistic μ-calculi.

When dealing with greatest fixpoints, a key proof technique relies on the coinduction principle, which uses the fact that a monotone function f over a complete lattice has a greatest fixpoint νf, which is the join of all post-fixpoints, i.e., the elements l such that $l \sqsubseteq f(l)$. As a consequence proving $l \sqsubseteq f(l)$ suffices to conclude that $l \sqsubseteq \nu f$.

In this setting, up-to techniques have been proposed for "simplifying" proofs [32, 39, 37, 35]. and for reducing the search space in verification (e.g., in [8], up-to techniques applied to language equivalence of NFAs are shown to provide in many cases an exponential speed-up). A sound up-to function is a function u on the lattice such that $\nu(f \circ u) \sqsubseteq \nu f$ and hence $l \sqsubseteq f(u(l))$ implies $l \sqsubseteq \nu(f \circ u) \sqsubseteq \nu f$. The characteristics of u (typically, extensiveness) make it easier to show that an element is a post-fixpoint of $f \circ u$ rather than a post-fixpoint of f.

We show that up-to techniques admit a natural interpretation as abstractions in our framework (§5). This allows us to generalise the theory of up-to techniques to systems of fixpoint equations and contributes to the understanding of the relation between abstract interpretation and up-to techniques, a theme that received some recent attention [6].

We have recently shown in [2] that the solution of systems of fixpoint equations can be characterised in terms of a parity game when working in a suitable subclass of complete lattices, the so-called continuous lattices [41]. Here, relying on our approximation theory, we get rid of continuity and design a game that works for general complete lattices (§6.1).

The above results open the way to the development of game-theoretical algorithms, possibly integrating abstraction and up-to techniques, for solving systems of equations over complete lattices. While global algorithms deciding the game at all positions, based on progress measures [25], have already been studied in [20, 2], here we focus on local algorithms, confining the attention to specific positions. Taking inspiration from backtracking methods for bisimilarity [21] and for the μ-calculus [45, 44], we design a local (also called on-the-fly) algorithm for the case of a single equation (§6.2) (general systems are dealt with in [3]). This also establishes a link with some recent work relating abstract interpretation and up-to techniques [6] and exploiting up-to techniques for language equivalence on NFAs [8].

2 Preliminaries and Notation

A preordered or partially ordered set $\langle P, \sqsubseteq \rangle$ is often denoted simply as P, omitting the (pre)order relation. Given $X \subseteq P$, we denote by $\downarrow X = \{p \in P \mid \exists x \in X.\ p \sqsubseteq x\}$ the *downward-closure* and by $\uparrow X = \{p \in P \mid \exists x \in X.\ x \sqsubseteq p\}$ the *upward-closure* of X. The *join* and the *meet* of a subset $X \subseteq P$ (if they exist) are denoted $\bigsqcup X$ and $\bigsqcap X$, respectively.

▶ **Definition 1** (complete lattice, basis). *A* complete lattice *is a partially ordered set $(L, \sqsubseteq)$ such that each subset $X \subseteq L$ admits a join $\bigsqcup X$ and a meet $\bigsqcap X$. A complete lattice $(L, \sqsubseteq)$ always has a least element $\bot = \bigsqcup \emptyset$ and a greatest element $\top = \bigsqcap \emptyset$. A* basis *for a complete lattice is a subset $B_L \subseteq L$ such that for each $l \in L$ it holds that $l = \bigsqcup(\downarrow l \cap B_L)$.*

For instance, the powerset of any set X, ordered by subset inclusion $(\mathbf{2}^X, \subseteq)$ is a complete lattice. Join is union, meet is intersection, top ($\top$) is X and bottom ($\bot$) is $\emptyset$. A basis is the set of singletons $B_{\mathbf{2}^X} = \{\{x\} \mid x \in X\}$. Another complete lattice used in the paper is the real interval $[0, 1]$ with the usual order $\leq$. Join and meet are the sup and inf over the reals, 0 is bottom and 1 is top. Any dense subset, e.g., the set of rationals $\mathbb{Q} \cap (0, 1]$, is a basis.

A function $f \colon L \to L$ is *monotone* if for all $l, l' \in L$, if $l \sqsubseteq l'$ then $f(l) \sqsubseteq f(l')$. By Knaster-Tarski's theorem [46, Theorem 1], any monotone function f on a complete lattice has a least fixpoint arising as the meet of all pre-fixpoints $\mu f = \bigsqcap\{l \mid f(l) \sqsubseteq l\}$ and a greatest fixpoint arising as the join of all post-fixpoints $\nu f = \bigsqcup\{l \mid l \sqsubseteq f(l)\}$.

Given a complete lattice L, a subset $X \subseteq L$ is *directed* if $X \neq \emptyset$ and every pair of elements in X has an upper bound in X. If L, L' are complete lattices, a function $f : L \to L'$ is *(directed-)continuous* if for any directed set $X \subseteq L$ it holds that $f(\bigsqcup X) = \bigsqcup f(X)$. The function f is called *strict* if $f(\bot) = \bot$. *Co-continuity* and *co-strictness* are defined dually.

▶ **Definition 2** (Galois connection). *Let $(C, \sqsubseteq)$, $(A, \leq)$ be complete lattices. A* Galois connection *(or* adjunction*) is a pair of monotone functions $\langle \alpha, \gamma \rangle$ such that $\alpha : C \to A$, $\gamma : A \to C$ and for all $a \in A$ and $c \in C$ it holds that $\alpha(c) \leq a$ iff $c \sqsubseteq \gamma(a)$.*

Equivalently, for all $a \in A$ and $c \in C$, (i) $c \sqsubseteq \gamma(\alpha(c))$ and (ii) $\alpha(\gamma(a)) \leq a$. In this case we will write $\langle \alpha, \gamma \rangle : C \to A$. The Galois connection is called an insertion *when $\alpha \circ \gamma = id_A$.*

For a Galois connection $\langle \alpha, \gamma \rangle : C \to A$, the function α is called the left (or lower) adjoint and γ the right (or upper) adjoint. The left adjoint α preserves all joins and the right adjoint γ preserves all meets. Hence, in particular, the left adjoint is strict and continuous, while the right adjoint is co-strict and co-continuous.

A function $f : L \to L$ is *idempotent* if $f \circ f = f$ and *extensive* if $l \sqsubseteq f(l)$ for all $l \in L$. When f is monotone, extensive and idempotent it is called an *(upper) closure*. In this case, $\langle f, i \rangle : L \to f(L)$, where i is the inclusion, is an insertion and $f(L) = \{f(l) \mid l \in L\}$ is a complete lattice.

We will often consider tuples of elements. Given a set A, an n-tuple in A^n is denoted by a boldface letter $\boldsymbol{a}$ and its components are denoted as $\boldsymbol{a} = (a_1, \ldots, a_n)$. For an index $n \in \mathbb{N}$ we write $\underline{n}$ for the integer interval $\{1, \ldots, n\}$. Given $\boldsymbol{a} \in A^n$ and $i, j \in \underline{n}$ we write $\boldsymbol{a}_{i,j}$ for the subtuple $(a_i, a_{i+1}, \ldots, a_j)$. The empty tuple is denoted by $()$. Given two tuples $\boldsymbol{a} \in A^m$ and $\boldsymbol{a'} \in A^n$ we denote by $(\boldsymbol{a}, \boldsymbol{a'})$ or simply by $\boldsymbol{a}\boldsymbol{a'}$ their concatenation in A^{m+n}.

Given a complete lattice $(L, \sqsubseteq)$ we will denote by $(L^n, \sqsubseteq)$ the set of n-tuples endowed with the *pointwise order* defined, for $\boldsymbol{l}, \boldsymbol{l'} \in L^n$, by $\boldsymbol{l} \sqsubseteq \boldsymbol{l'}$ if $l_i \sqsubseteq l'_i$ for all $i \in \underline{n}$. The structure $(L^n, \sqsubseteq)$ is a complete lattice. More generally, for any set X, the set of functions $L^X = \{f \mid f \colon X \to L\}$, endowed with pointwise order, is a complete lattice.

A tuple of functions $\boldsymbol{f} = (f_1, \ldots, f_m)$ with $f_i : X \to Y$, will be seen itself as a function $\boldsymbol{f} : X \to Y^m$, defined by $\boldsymbol{f}(x) = (f_1(x), \ldots, f_m(x))$. We will also need to consider the *product function* $\boldsymbol{f}^\times : X^m \to Y^m$, defined by $\boldsymbol{f}^\times(x_1, \ldots, x_m) = (f_1(x_1), \ldots, f_m(x_m))$.

3 Systems of Fixpoint Equations over Complete Lattices

We deal with systems of (fixpoint) equations over some complete lattice, where, for each equation one can be interested either in the least or in the greatest solution. We define systems, their solutions and we provide some examples that will be used as running examples.

▶ **Definition 3** (system of equations). *Let L be a complete lattice. A system of equations E over L is an ordered list of m equations of the form $x_i =_{\eta_i} f_i(x_1, \ldots, x_m)$, where $f_i \colon L^m \to L$ are monotone functions (with respect to the pointwise order on L^m) and $\eta_i \in \{\mu, \nu\}$. The system will often be denoted as $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$, where $\boldsymbol{x}$, $\boldsymbol{\eta}$ and $\boldsymbol{f}$ are the obvious tuples. We denote by $\emptyset$ the system with no equations.*

Systems of this kind have been often considered in connection to verification problems (see e.g., [11, 42, 20, 2]). In particular, [20, 2] work on general classes of complete lattices.

Note that $\boldsymbol{f}$ can be seen as a function $\boldsymbol{f} \colon L^m \to L^m$. The solution of the system is a selected fixpoint of such function. We first need some auxiliary notation.

▶ **Definition 4** (substitution). *Given a system E of m equations over a complete lattice L of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$, an index $i \in \underline{m}$ and $l \in L$ we write $E[x_i := l]$ for the system of $m-1$ equations obtained from E by removing the i-th equation and replacing x_i by l in the other equations, i.e., if $\boldsymbol{x} = \boldsymbol{x}' x_i \boldsymbol{x}''$, $\boldsymbol{\eta} = \boldsymbol{\eta}' \eta_i \boldsymbol{\eta}''$ and $\boldsymbol{f} = \boldsymbol{f}' f_i \boldsymbol{f}''$ then $E[x_i := l]$ is $\boldsymbol{x}'\boldsymbol{x}'' =_{\boldsymbol{\eta}'\boldsymbol{\eta}''} \boldsymbol{f}'\boldsymbol{f}''(\boldsymbol{x}', l, \boldsymbol{x}'')$.*

For solving a system of m equations $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$, the last variable x_m is considered as a fixed parameter x and the system of $m-1$ equations $E[x_m := x]$ that arises from dropping the last equation is recursively solved. This produces an $(m-1)$-tuple parametric on x, i.e., we get $\boldsymbol{s}_{1,m-1}(x) = sol(E[x_m := x])$. Inserting this parametric solution into the last equation, we get an equation in a single variable $x =_{\eta_m} f_m(\boldsymbol{s}_{1,m-1}(x), x)$ that can be solved by taking for the function $\lambda x.\, f_m(\boldsymbol{s}_{1,m-1}(x), x)$, the least or greatest fixpoint, depending on whether the last equation is a μ- or ν-equation. This provides the m-th component of the solution $s_m = \eta_m(\lambda x.\, f_m(\boldsymbol{s}_{1,m-1}(x), x))$. The remaining components are obtained inserting s_m in the parametric solution $\boldsymbol{s}_{1,m-1}(x)$ previously computed, i.e., $\boldsymbol{s}_{1,m-1} = \boldsymbol{s}_{1,m-1}(s_m)$.

▶ **Definition 5** (solution). *Let L be a complete lattice and let E be a system of m equations over L of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$. The* solution *of E, denoted $sol(E) \in L^m$, is defined inductively:*

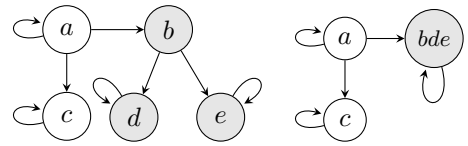
$$sol(\emptyset) = () \qquad sol(E) = (sol(E[x_m := s_m]),\, s_m)$$

where $s_m = \eta_m(\lambda x.\, f_m(sol(E[x_m := x]), x))$.

The order of equations matters: changing the order typically leads to a different solution.

▶ **Example 6** (solving a simple system of equations). Consider the powerset lattice $\mathbf{2}^S$ of any non-empty set S and the system of equations E consisting of the following two equations

$$x =_\mu x \cup y$$
$$y =_\nu x \cap y$$



(a) (b)

(c) $\begin{array}{lll} x_1 & =_\nu & p \wedge \Box x_1 \\ x_2 & =_\mu & x_1 \vee \Diamond x_2 \end{array}$

(d) $\begin{array}{lll} x_1 & =_\nu & \{b,d,e\} \cap \blacksquare_T x_1 \\ x_2 & =_\mu & x_1 \cup \blacklozenge_T x_2 \end{array}$

Figure 1 Transition systems and equational form for a μ-calculus formula.

In order to solve the system E, initially we need to compute the solution of the first equation $x =_\mu x \cup y$ parametric in y, that is, $s_x(y) = \mu(\lambda x.(x \cup y)) = y$. Now we can solve the second equation $y =_\nu x \cap y$ replacing x with the parametric solution, obtaining an equation in a single variable whose solution is $\nu(\lambda y.(s_x(y) \cap y)) = \nu(\lambda y.y) = S$. Finally, the solution of the first equation is obtained by inserting $y = S$ in the parametric solution $x = s_x(S) = S$.

Observe that even in this simple example the order of the equations matters. Indeed, if we consider the system where the two equations above are swapped the solution is $x = y = \emptyset$.

▶ **Example 7** (μ-calculus formulae as fixpoint equations)**.** We adopt a standard μ-calculus syntax. For fixed disjoint sets $PVar$ of propositional variables, ranged over by $x, y, z, \ldots$ and $Prop$ of propositional symbols, ranged over by $p, q, r, \ldots$, formulae are defined by

$$\varphi \ ::= \ \mathbf{t} \mid \mathbf{f} \mid p \mid x \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \Box\varphi \mid \Diamond\varphi \mid \eta x.\varphi$$

where $p \in Prop$, $x \in PVar$ and $\eta \in \{\mu, \nu\}$.

The semantics of a formula is given with respect to an unlabelled transition system (or Kripke structure) $T = (\mathbb{S}_T, \to_T)$ where $\mathbb{S}_T$ is the set of states and $\to_T \subseteq \mathbb{S}_T \times \mathbb{S}_T$ is the transition relation. Given a formula φ and an environment ρ: $Prop \cup PVar \to \mathbf{2}^{\mathbb{S}_T}$ mapping each proposition or propositional variable to the set of states where it holds, we denote by $\|\varphi\|^T_\rho$ the semantics of φ defined as usual (see, e.g., [9]).

As observed by several authors (see, e.g., [11, 42]), a μ-calculus formula can be seen as a system of equations, with an equation for each fixpoint subformula. For instance, consider $\varphi = \mu x_2.((\nu x_1.(p \wedge \Box x_1)) \vee \Diamond x_2)$ that requires that a state is eventually reached from which p always holds. The equational form is reported in Fig. 1c. Consider a transition system $T = (\mathbb{S}_T, \to_T)$ where $\mathbb{S}_T = \{a, b, c, d, e\}$ and $\to_T$ is as depicted in Fig. 1a, with p that holds in the grey states b, d and e. Define the semantic counterpart of the modal operators as follows: given a relation $R \subseteq X \times X$ let $\blacklozenge_R, \blacksquare_R : \mathbf{2}^X \to \mathbf{2}^X$ be the functions defined, for $Y \subseteq X$, by $\blacklozenge_R(Y) = \{x \in X \mid \exists y \in Y.\ (x, y) \in R\}$, $\blacksquare_R(Y) = \{x \in X \mid \forall y \in X.(x, y) \in R \Rightarrow y \in Y\}$. Then the formula φ interpreted over the transition system T leads to the system of equations over the lattice $\mathbf{2}^{\mathbb{S}_T}$ in Fig. 1d, where we write $\blacklozenge_T$ and $\blacksquare_T$ for $\blacklozenge_{\to_T}$ and $\blacksquare_{\to_T}$.

The solution is $x_1 = \{b, d, e\}$ (states where p always holds) and $x_2 = \{a, b, d, e\}$ (states where the formula φ holds).

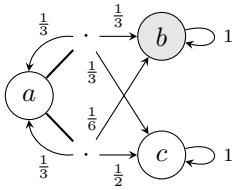
▶ **Example 8** (Łukasiewicz μ-terms)**.** Systems of equations over the real interval $[0, 1]$ have been considered in [34] as a precursor to model-checking PCTL or probabilistic μ-calculi. More precisely, the authors study a fixpoint extension of Łukasiewicz logic, referred to as Łukasiewicz μ-terms, whose syntax is as follows:

$$t \ ::= \ \mathbf{1} \mid \mathbf{0} \mid x \mid r \cdot t \mid t \sqcup t \mid t \sqcap t \mid t \oplus t \mid t \odot t \mid \eta x.t$$

where $x \in PVar$ is a variable (ranging over $[0, 1]$), $r \in [0, 1]$ and $\eta \in \{\mu, \nu\}$. The various syntactic operators have a semantic counterpart, given in Fig. 2a.

$\mathbf{0}(x) = 0,\ \mathbf{1}(x) = 1$	(constant)
$r \cdot x = rx$	(scalar mult.)
$x \sqcup y = \max(x, y)$	(weak disj.)
$x \sqcap y = \min(x, y)$	(weak conj.)
$x \oplus y = \min(x + y, 1)$	(strong disj.)
$x \odot y = \max(x + y - 1, 0)$	(strong conj.)

(a) Semantics of μ-terms ($x, y \in [0, 1]$).



(b) A PNDT.

$$\varphi \quad \begin{cases} x_1 =_\nu p \odot \Box x_1 \\ x_2 =_\mu x_1 \oplus \Diamond x_2 \end{cases}$$

$$\varphi' \quad \begin{cases} x_1 =_\nu p \odot \Box x_1 \\ x_2 =_\mu x_1 \oplus \Box x_2 \end{cases}$$

(c) Formulas as systems.

Figure 2

Then, each Łukasiewicz μ-term, in an environment $\rho : PVar \to [0, 1]$, can be assigned a semantics which is a real number in $[0, 1]$, denoted as $\|t\|_\rho$. Exactly as for the μ-calculus, a Łukasiewicz μ-term can be naturally seen as a system of fixpoint equations over the lattice $[0, 1]$. For instance, the term $\nu x_2.\ (\mu x_1.\ (\frac{5}{8} \oplus \frac{3}{8}x_2) \odot (\frac{1}{2} \sqcup (\frac{3}{8} \oplus \frac{1}{2}x_1)))$ from an example in [34], can be written as the system:

$$x_1 =_\mu (\frac{5}{8} \oplus \frac{3}{8}x_2) \odot (\frac{1}{2} \sqcup (\frac{3}{8} \oplus \frac{1}{2}x_1))$$
$$x_2 =_\nu x_1$$

▶ **Example 9** (Łukasiewicz μ-calculus)**.** The Łukasiewicz μ-calculus, as defined in [34], extends the Łukasiewicz μ-terms with propositions and modal operators. The syntax is as follows:

$$\varphi \ ::= \ p \mid \bar{p} \mid r \cdot \varphi \mid \varphi \sqcup \varphi \mid \varphi \sqcap \varphi \mid \varphi \oplus \varphi \mid \varphi \odot \varphi \mid \Diamond\varphi \mid \Box\varphi \mid \eta x.t$$

where x ranges in a set $PVar$ of propositional variables, p ranges in a set $Prop$ of propositional symbols, each paired with an associated complement $\bar{p}$, and $\eta \in \{\mu, \nu\}$.

The Łukasiewicz μ-calculus can be seen as a logic for probabilistic transition systems. It extends the quantitative modal μ-calculus of [31, 24] and it allows to encode PCTL [5]. For a finite set $\mathbb{S}$, the set of (discrete) probability distributions over $\mathbb{S}$ is defined as $\mathcal{D}(\mathbb{S}) = \{d : \mathbb{S} \to [0, 1] \mid \sum_{s \in \mathbb{S}} d(s) = 1\}$. A formula is interpreted over a *probabilistic non-deterministic transition system (PNDT)* $N = (\mathbb{S}, \to)$ where $\to \subseteq \mathbb{S} \times \mathcal{D}(\mathbb{S})$ is the transition relation. An example of PNDT can be found in Fig. 2b. Imagine that the aim is to reach state b. State a has two transitions. A "lucky" one where the probability to get to b is $\frac{1}{3}$ and an "unlucky" one where b is reached with probability $\frac{1}{6}$. For both transitions, with probability $\frac{1}{3}$ one gets back to a and then, with the residual probability, one moves to c. Once in states b or c, the system remains in the same state with probability 1.

Given a formula φ and an environment $\rho : Prop \cup PVar \to (\mathbb{S} \to [0, 1])$ mapping each proposition or propositional variable to a real-valued function over the states, the semantics of φ is a function $\|\varphi\|^N_\rho : \mathbb{S} \to [0, 1]$ defined as expected using the semantic operators. In addition to those already discussed, we have the semantic operators for the complement and the modalities: for $v : \mathbb{S} \to [0, 1]$

$$\bar{v}(x) = 1 - v(x) \qquad \blacklozenge_N(v)(x) = \max_{x \to d} \sum_{y \in \mathbb{S}} d(y) \cdot v(y) \qquad \blacksquare_N(v)(x) = \min_{x \to d} \sum_{y \in \mathbb{S}} d(y) \cdot v(y)$$

As it happens for the propositional μ-calculus, also formulas of the Łukasiewicz μ-calculus can be seen as systems of equations, but on a different complete lattice, i.e., $[0, 1]^{\mathbb{S}}$. For instance, consider the formulas $\varphi = \mu x_2.(\nu x_1.(p \odot \Box x_1) \oplus \Diamond x_2)$ and $\varphi' = \mu x_2.(\nu x_1.(p \odot \Box x_1) \oplus \Box x_2)$, rendered as (syntactic) equations in Fig. 2c. Roughly speaking, they capture the probability of eventually satisfying forever p, with an angelic scheduler and a daemonic one, choosing at each step the best or worst transition, respectively. Assuming that p holds with probability 1 on b and 0 on a and c, we have $\|\varphi\|_\rho(a) = \frac{1}{2}$ and $\|\varphi'\|_\rho(a) = \frac{1}{4}$.

▶ **Example 10** ((bi)similarity over transition systems). For defining (bi)similarity uniformly with the example on μ-calculus, we work on unlabelled transition systems with atoms $T = (\mathbb{S}, \to, A)$ where $A \subseteq \mathbf{2}^{\mathbb{S}}$ is a fixed set of atomic properties over the states. Everything can be easily adapted to labelled transition systems.

Given $T = (\mathbb{S}, \to, A)$, consider the lattice of relations on $\mathbb{S}$, namely $\mathsf{Rel}(\mathbb{S}) = (\mathbf{2}^{\mathbb{S}\times\mathbb{S}}, \subseteq)$. We take as basis the set of singletons $B_{\mathsf{Rel}(\mathbb{S})} = \{\{(x, y)\} \mid x, y \in \mathbb{S}\}$. The *similarity relation* on T, denoted $\precsim_T$, is the greatest fixpoint of the function $sim_T : \mathsf{Rel}(\mathbb{S}) \to \mathsf{Rel}(\mathbb{S})$, defined by

$$sim_T(R) = \left\{ (x, y) \in R \;\middle|\; \forall a \in A.\, (x \in a \Rightarrow y \in a) \wedge \forall x \to x'.\, \exists y \to y'.\, (x', y') \in R \right\}$$

In other words it can be seen as the solution of a system consisting of a single greatest fixpoint equation $x =_\nu sim_T(x)$.

For instance, consider the transition system T in Fig. 1a and take $p = \{b, d, e\}$ as the only atom. Then similarity $\precsim_T$ is the transitive reflexive closure of $\{(c, a), (a, b), (b, d), (d, e), (e, b)\}$.

Bisimilarity $\sim_T$ can be obtained analogously as the greatest fixpoint of $bis_T(R) = sim_T(R) \cap sim_T(R^{-1})$. In the transition system T above, bisimilarity $\sim_T$ is the equivalence such that $b \sim_T d \sim_T e$.

4 Approximation for Systems of Fixpoint Equations

In this section we design a theory of approximation for systems of fixpoint equations over complete lattices. The general setup is borrowed from abstract interpretation [13, 14], where a concrete domain C and an abstract domain A are fixed. Semantic operators on the concrete domain C have a counterpart in the abstract domain A, and suitable conditions can be imposed on such operators to ensure that the least fixpoints of the abstract operators are sound and/or complete approximations of the fixpoints of their concrete counterparts.

Similarly, here we will have a system of equations $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^C(\boldsymbol{x})$ over a concrete domain C and its abstract counterpart $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^A(\boldsymbol{x})$ over an abstract domain A, and we want that the solution of the latter provides an approximation of the solution of the former.

Let us first focus on the case of a single equation. Let $(C, \sqsubseteq)$ and $(A, \leq)$ be complete lattices and let $f^C : C \to C$ and $f^A : A \to A$ be monotone functions. The fact that f^A is a sound (over)approximation of f^C can be formulated in terms of a concretisation function $\gamma : A \to C$, that maps each abstract element $a \in A$ to a concrete element $\gamma(a) \in C$, for which, intuitively, a is an overapproximation. In the setting of abstract interpretation, where the interest is for program semantics, typically expressed in terms of least fixpoints, the desired *soundness* property is $\mu f^C \sqsubseteq \gamma(\mu f^A)$. A standard sufficient condition for soundness (see [13, 14, 33]) is $f^C \circ \gamma \sqsubseteq \gamma \circ f^A$. The same condition ensures soundness also for greatest fixpoints, i.e., $\nu f^C \sqsubseteq \gamma(\nu f^A)$, provided that γ is co-continuous and co-strict (see, e.g., [15, Proposition 15], which states the dual result).

Then we can suitably combine the conditions for least and greatest fixpoints. We will allow a different concretisation function for each equation.

▶ **Theorem 11** (sound concretisation for systems). *Let $(C, \sqsubseteq)$ and $(A, \leq)$ be complete lattices, let E_C of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^C(\boldsymbol{x})$ and E_A of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^A(\boldsymbol{x})$ be systems of m equations over C and A, with solutions $\boldsymbol{s}^C \in C^m$ and $\boldsymbol{s}^A \in A^m$, respectively. Let $\boldsymbol{\gamma}$ be an m-tuple of monotone functions, with $\gamma_i : A \to C$ for $i \in \underline{m}$. If $\boldsymbol{\gamma}$ satisfies $\boldsymbol{f}^C \circ \boldsymbol{\gamma}^\times \sqsubseteq \boldsymbol{\gamma}^\times \circ \boldsymbol{f}^A$ with γ_i co-continuous and co-strict for each $i \in \underline{m}$ such that $\eta_i = \nu$, then $\boldsymbol{s}^C \sqsubseteq \boldsymbol{\gamma}^\times(\boldsymbol{s}^A)$.*

The standard abstract interpretation framework of [16] relies on Galois connections: concretisation functions γ are right adjoints, whose left adjoint, the abstraction function α, intuitively maps each concrete element in C to its "best" overapproximation in A. When

$\langle \alpha, \gamma \rangle$ is a Galois connection, α is automatically continuous and strict, while γ is co-continuous and co-strict. This leads to the following result, where, besides the soundness conditions, we also make explicit the completeness conditions.

▶ **Theorem 12** (abstraction via Galois connections). *Let $(C, \sqsubseteq)$ and $(A, \leq)$ be complete lattices, let E_C of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^C(\boldsymbol{x})$ and E_A of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}^A(\boldsymbol{x})$ be systems of m equations over C and A, with solutions $\boldsymbol{s}^C \in C^m$ and $\boldsymbol{s}^A \in A^m$, respectively. Let $\boldsymbol{\alpha}$ and $\boldsymbol{\gamma}$ be m-tuples of monotone functions, with $\langle \alpha_i, \gamma_i \rangle : C \to A$ a Galois connection for each $i \in \underline{m}$.*

1. Soundness*: If $\boldsymbol{\gamma}$ satisfies $\boldsymbol{f}^C \circ \boldsymbol{\gamma}^\times \sqsubseteq \boldsymbol{\gamma}^\times \circ \boldsymbol{f}^A$ or equivalently $\boldsymbol{\alpha}$ satisfies $\boldsymbol{\alpha}^\times \circ \boldsymbol{f}^C \leq \boldsymbol{f}^A \circ \boldsymbol{\alpha}^\times$, then $\boldsymbol{\alpha}^\times(\boldsymbol{s}^C) \leq \boldsymbol{s}^A$ (equivalent to $\boldsymbol{s}^C \sqsubseteq \boldsymbol{\gamma}^\times(\boldsymbol{s}^A)$).*
2. Completeness (for abstraction)*: If $\boldsymbol{\alpha}$ satisfies $\boldsymbol{f}^A \circ \boldsymbol{\alpha}^\times \leq \boldsymbol{\alpha}^\times \circ \boldsymbol{f}^C$ with α_i co-continuous and co-strict for each $i \in \underline{m}$ such that $\eta_i = \nu$, then $\boldsymbol{s}^A \leq \boldsymbol{\alpha}^\times(\boldsymbol{s}^C)$.*
3. Completeness (for concretisation)*: If $\boldsymbol{\gamma}$ satisfies $\boldsymbol{\gamma}^\times \circ \boldsymbol{f}^A \sqsubseteq \boldsymbol{f}^C \circ \boldsymbol{\gamma}^\times$ with γ_i continuous and strict for each $i \in \underline{m}$ such that $\eta_i = \mu$, then $\boldsymbol{\gamma}^\times(\boldsymbol{s}^A) \sqsubseteq \boldsymbol{s}^C$.*

Completeness for the abstraction, i.e., $\boldsymbol{s}^A \leq \boldsymbol{\alpha}^\times(\boldsymbol{s}^C)$, together with soundness, leads to $\boldsymbol{\alpha}^\times(\boldsymbol{s}^C) = \boldsymbol{s}^A$. This is a rare but very pleasant situation in which the abstraction does not lose any information as far as the abstract properties are concerned. We remark that here the notion of "completeness" slightly deviates from the standard abstract interpretation terminology where soundness is normally indispensable, and thus complete abstractions (see, e.g., [18]) are, by default, also sound.

Moreover, completeness for the concretisation is normally of limited interest in abstract interpretation. Alone, it states that the abstract solution is an underapproximation of the concrete one, while typically the interest is for overapproximations. Together with soundness, it leads to $\boldsymbol{s^C} = \boldsymbol{\gamma}^\times(\boldsymbol{s^A})$, a very strong property which is not meaningful in program analysis. In our case, keeping the concepts of soundness and completeness separated and considering also completeness for the concretisation is helpful in some cases, especially when dealing with up-to functions, which are designed to provide underapproximations of fixpoints.

Standard arguments also show that abstract operators can be obtained compositionally out of basic ones, preserving soundness.

▶ **Example 13** (abstraction for the μ-calculus). The paper [30] observes that (bi)simulations over transition systems can be seen as Galois connections and interpreted as abstractions. Then it characterises fragments of the μ-calculus which are preserved and strongly preserved by the abstraction. We next discuss how this can be derived as an instance of our framework.

Let $T_C = (\mathbb{S}_C, \to_C)$ and $T_A = (\mathbb{S}_A, \to_A)$ be transition systems and let $\langle \alpha, \gamma \rangle : \mathbf{2}^{\mathbb{S}_C} \to \mathbf{2}^{\mathbb{S}_A}$ be a Galois connection. It is a *simulation*, according to [30], if it satisfies the following condition: $\alpha \circ \blacklozenge_{T_C} \circ \gamma \subseteq \blacklozenge_{T_A}$. In this case T_A is called a $\langle \alpha, \gamma \rangle$-*abstraction* of T_C, written $T_C \sqsubseteq_{\langle \alpha, \gamma \rangle} T_A$. This can be shown to be equivalent to the ordinary notion of simulation between transition systems [30, Propositions 9 and 10]. In particular, if $R \subseteq \mathbb{S}_C \times \mathbb{S}_A$ is a simulation in the ordinary sense then one can consider $\langle \blacklozenge_{R^{-1}}, \blacksquare_R \rangle : \mathbf{2}^{\mathbb{S}_C} \to \mathbf{2}^{\mathbb{S}_A}$, where $\blacklozenge_{R^{-1}}$ is the function $\blacklozenge_{R^{-1}}(X) = \{y \in \mathbb{S}_A \mid \exists x \in X.\ (x, y) \in R\}$. This is a Galois connection (in the abstract interpretation setting $\blacklozenge_{R^{-1}}$ and $\blacksquare_R$ are often denoted $\widetilde{pre}_R$ and $post_R$, respectively [12]) inducing a simulation in the above sense, i.e., $\blacklozenge_{R^{-1}} \circ \blacklozenge_{T_C} \circ \blacksquare_R \subseteq \blacklozenge_{T_A}$.

When $T_C \sqsubseteq_{\langle \alpha, \gamma \rangle} T_A$, by [30, Theorem 2], one has that α "preserves" the $\mu\Diamond$-calculus, i.e., the fragment of the μ-calculus without $\Box$ operators. More precisely, for any formula φ of the $\mu\Diamond$-calculus, we have $\alpha(\|\varphi\|^{T_C}_{\rho}) \subseteq \|\varphi\|^{T_A}_{\alpha \circ \rho}$. This means that for each $s_C \in \mathbb{S}_C$, if s_C satisfies φ in the concrete system, then all the states in $\alpha(\{s_C\})$ satisfy φ in the abstract system, provided that each proposition p is interpreted in A with $\alpha(\rho(p))$, the abstraction of its interpretation in C.

This can be obtained as an easy consequence of Theorem 12, where we use the same function α as an abstraction for all equations. The condition $\alpha \circ \blacklozenge_{T_C} \circ \gamma \subseteq \blacklozenge_{T_A}$ above can be rewritten as $\alpha \circ \blacklozenge_{T_C} \subseteq \blacklozenge_{T_A} \circ \alpha$ which is the soundness condition $(\boldsymbol{\alpha}^{\times} \circ \boldsymbol{f}^{C} \leq \boldsymbol{f}^{A} \circ \boldsymbol{\alpha}^{\times})$ in Theorem 12 for the semantics of the diamond operator. For the other operators the soundness condition is trivially shown to hold. In fact,

- for $\mathbf{t}$ and $\mathbf{f}$ we have $\alpha(\emptyset) = \emptyset$ and $\alpha(\mathbb{S}_C) \subseteq \mathbb{S}_A$;
- for $\wedge$ and $\vee$ we have $\alpha(X \cup Y) = \alpha(X) \cup \alpha(Y)$ and $\alpha(X \cap Y) \subseteq \alpha(X) \cap \alpha(Y)$;
- a proposition p represents the constant function $\rho(p)$ in T_C and $\alpha(\rho(p))$ in T_A.

In order to extend the logic by including negation on propositions, in [30], an additional condition is required, called *consistency* of the abstraction with respect to the interpretation: $\alpha(\rho(p)) \cap \alpha(\overline{\rho(p)}) = \emptyset$, for all p. This is easily seen to be equivalent to $\alpha(\overline{\rho(p)}) \subseteq \overline{\alpha(\rho(p))}$ which is the soundness condition $(\boldsymbol{\alpha}^{\times} \circ \boldsymbol{f}^{C} \leq \boldsymbol{f}^{A} \circ \boldsymbol{\alpha}^{\times})$ in Theorem 12 for negated propositions.

Our theory naturally suggests generalisations of [30]. E.g., by (the dual of) Theorem 11, continuity and strictness of the abstraction α are sufficient to retain the results, hence one could deal with an abstraction not being an adjoint, thus going beyond ordinary simulations.

▶ **Example 14** (abstraction for Łukasiewicz μ-terms)**.** For Łukasiewicz μ-terms, as introduced in Example 8, leading to systems of fixpoint equations over the reals, we can consider as an abstraction a form of discretisation: for some fixed n define the abstract domain $[0,1]_{/n} = \{0\} \cup \{k/n \mid k \in \underline{n}\}$ and the insertion $\langle \alpha_n, \gamma_n \rangle : [0,1] \to [0,1]_{/n}$ with α_n defined by $\alpha_n(x) = \lceil n \cdot x \rceil / n$ and γ_n the inclusion. We can consider for all operators op, their best abstraction $op^{\#} = \alpha_n \circ op \circ \boldsymbol{\gamma_n}^{\times}$, thus getting a sound abstraction.

Note that for all semantic operators, $op^{\#}$ is the restriction of op to the abstract domain, with the exception of $r \cdot^{\#} x = \alpha_n(r \cdot x)$ for $x \in [0,1]_{/n}$. Moreover, for $x, y \in [0,1]$ we have

- $\alpha_n(\mathbf{0}(x)) = \mathbf{0}^{\#}(\alpha_n(x))$, $\alpha_n(\mathbf{1}(x)) = \mathbf{1}^{\#}(\alpha_n(x))$;
- $\alpha_n(r \cdot x) \leq r \cdot^{\#} \alpha_n(x)$;
- $\alpha_n(x \sqcup y) = \alpha_n(x) \sqcup^{\#} \alpha_n(y)$, $\alpha_n(x \sqcap y) = \alpha_n(x) \sqcap^{\#} \alpha_n(y)$;
- $\alpha_n(x \oplus y) \leq \alpha_n(x) \oplus^{\#} \alpha_n(y)$, $\alpha_n(x \odot y) \leq \alpha_n(x) \odot^{\#} \alpha_n(y)$ since $\alpha_n(x+y) \leq \alpha_n(x) + \alpha_n(y)$

i.e., the abstraction is complete for $\mathbf{0}$, $\mathbf{1}$, $\sqcup$, $\sqcap$, while it is just sound for the remaining operators.

For instance, the system in Example 8 can be shown to have solution $x_1 = x_2 = 0.2$. With abstraction α_{10} we get $x_1 = x_2 = 0.8$, with a more precise abstraction α_{100} we get $x_1 = x_2 = 0.22$ and with α_{1000} we get $x_1 = x_2 = 0.201$.

▶ **Example 15** (abstraction for Łukasiewicz μ-calculus)**.** Although space limitations prevent a detailed discussion, observe that when dealing with Łukasiewicz μ-calculus over some probabilistic transition system $N = (\mathbb{S}, \to)$, we can lift the Galois insertion above to $[0,1]^{\mathbb{S}}$. Define $\alpha_n^{\to} : [0,1]^{\mathbb{S}} \to [0,1]^{\mathbb{S}}_{/n}$ by letting, $\alpha_n^{\to}(v) = \alpha_n \circ v$ for $v \in [0,1]^{\mathbb{S}}$. Then $\langle \alpha_n^{\to}, \gamma_n^{\to} \rangle : [0,1]^{\mathbb{S}} \to [0,1]^{\mathbb{S}}_{/n}$, where $\gamma_n^{\to}$ is the inclusion, is a Galois insertion and, as in the previous case, we can consider the best abstraction for the operators of the Łukasiewicz μ-calculus.

For instance, consider the system for φ' in Example 9. Recall that the exact solution is $x_2(a) = 0.25$. With abstraction α_{10} we get $x_2(a) = 0.3$, with α_{15} we get $x_2(a) = 0.2\bar{6}$.

5 Up-To Techniques

Up-to techniques have been shown effective in easing the proof of properties of greatest fixpoints. Originally proposed for coinductive behavioural equivalences [32, 39], they have been later studied in the setting of complete lattices [35, 36]. Some recent work [6] started the exploration of the relation between up-to techniques and abstract interpretation. Roughly,

they work in a setting where the semantic functions of interest $f^* : L \to L$ admits a left adjoint $f_* : L \to L$, the intuition being that f^* and f_* are predicate transformers mapping a condition into, respectively, its strongest postcondition and weakest precondition. Then complete abstractions for f^* and sound up-to functions for f_* are shown to coincide. This has a natural interpretation in our game theoretic framework, as discussed in §6.2.

Here we take another view. We work with general semantic functions and, in §5.1, we first argue that up-to techniques can be naturally interpreted as abstractions where the concretisation is complete (and sound, if the up-to function is a closure). Then, in §5.2 we can smoothly extend up-to techniques from a single fixpoint to systems of fixpoint equations.

5.1 Up-To Techniques as Abstractions

The general idea of up-to techniques is as follows. Given a monotone function $f : L \to L$ one is interested in the greatest fixpoint νf. In general, the aim is to establish whether some given element of the lattice $l \in L$ is under the fixpoint, i.e., if $l \sqsubseteq \nu f$. In turn, since by Tarski's Theorem, $\nu f = \bigsqcup\{x \mid x \sqsubseteq f(x)\}$, this amounts to proving that l is under some post-fixpoint l', i.e., $l \sqsubseteq l' \sqsubseteq f(l')$. For instance, consider the function $bis_T : \mathsf{Rel}(\mathbb{S}) \to \mathsf{Rel}(\mathbb{S})$ for bisimilarity on a transition system T in Example 10. Given two states $s_1, s_2 \in \mathbb{S}$, proving $\{(s_1, s_2)\} \subseteq \nu bis_T$, i.e., showing the two states bisimilar, amounts to finding a post-fixpoint, i.e., a relation R such that $R \subseteq bis_T(R)$ (namely, a bisimulation) such that $\{(s_1, s_2)\} \subseteq R$.

▶ **Definition 16** (up-to function). *Let L be a complete lattice and let $f : L \to L$ be a monotone function. A* sound up-to function *for f is any monotone function $u : L \to L$ such that $\nu(f \circ u) \sqsubseteq \nu f$. It is called* complete *if also the converse inequality $\nu f \sqsubseteq \nu(f \circ u)$ holds.*

When u is sound, if l is a post-fixpoint of $f \circ u$, i.e., $l \sqsubseteq f(u(l))$ we have $l \sqsubseteq \nu(f \circ u) \sqsubseteq \nu f$. The idea is that the characteristics of u should make it easier to prove that l is a postfix-point of $f \circ u$ than proving that it is for f. This is clearly the case when u is extensive. In fact by extensiveness of u and monotonicity of f we get $f(l) \sqsubseteq f(u(l))$ and thus obtaining $l \sqsubseteq f(u(l))$ is "easier" than obtaining $l \sqsubseteq f(l)$. Note that extensiveness also implies "completeness" of the up-to function: since $f \sqsubseteq f \circ u$ clearly $\nu f \sqsubseteq \nu(f \circ u)$. We remark that for up-to functions, since the interest is for underapproximating fixpoints, the terms soundness and completeness are somehow reversed with respect to their meaning in abstract interpretation.

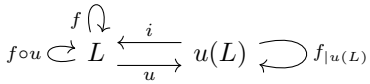
A common sufficient condition ensuring soundness of up-to functions is compatibility [35].

▶ **Definition 17** (compatibility). *Let L be a complete lattice and let $f : L \to L$ be a monotone function. A monotone function $u : L \to L$ is* f-compatible *if $u \circ f \sqsubseteq f \circ u$.*

The soundness of an f-compatible up-to function u can be proved by viewing it as an abstraction. When u is a closure (i.e., extensive and idempotent), $u(L)$ is a complete lattice that can be seen as an abstract domain in a way that $\langle u, i\rangle : L \to u(L)$, with i being the inclusion, is a Galois insertion. Moreover $f_{|u(L)}$ can be shown to provide an abstraction of both f and $f \circ u$ over L, sound and complete with respect to the inclusion i, seen as the concretisation. The formal details are given below. Since we later aim to apply up-to techniques to systems of equations, we deal with not only greatest but also least fixpoints.

▶ **Lemma 18** (compatible up-to functions as sound and complete abstractions). *Let $f : L \to L$ be a monotone function and let $u : L \to L$ be an f-compatible closure. Consider the Galois insertion $\langle u, i\rangle : L \to u(L)$ where $i : u(L) \to L$ is the inclusion. Then*

1. *f restricts to $u(L)$, i.e., $f_{|u(L)} : u(L) \to u(L)$;*
2. *$\nu f = i(\nu f_{|u(L)}) = \nu(f \circ u)$. If u is continuous and strict then $\mu f = i(\mu f_{|u(L)}) = \mu(f \circ u)$.*



When the up-to function is just f-compatible (hence sound), but possibly not a closure, we canonically turn u into an f-compatible closure by taking the least closure $\bar{u}$ above u.

▶ **Corollary 19** (soundness of compatible up-to functions). *Let $f : L \to L$ be a monotone function, let $u : L \to L$ be an f-compatible up-to function and let $\bar{u}$ be the least closure above u. Then $\nu(f \circ u) \sqsubseteq \nu(f \circ \bar{u}) = \nu f$. If u is continuous and strict, then $\mu(f \circ u) \sqsubseteq \mu(f \circ \bar{u}) = \mu f$.*

In [35] the proof of soundness of a compatible up-to technique u relies on the definition of a function u^ω defined as $u^\omega(x) = \bigsqcup\{u^n(x) \mid n \in \mathbb{N}\}$, where $u^n(x)$ is defined inductively as $u^0(x) = x$ and $u^{n+1}(x) = u(u^n(x))$. The function u^ω is extensive but not idempotent in general, and it can be easily seen that $u^\omega \sqsubseteq \bar{u}$. The paper [36] shows that for any monotone function one can consider the largest compatible up-to function, the so-called companion, which is extensive and idempotent. The companion could be used in place of $\bar{u}$ for part of the theory. However, we find it convenient to work with $\bar{u}$ since, despite not discussed in the present paper, it plays a key role for the integration of up-to techniques into the verification algorithms. Furthermore the companion is usually hard to determine.

5.2 Up-To Techniques for Systems of Equations

Exploiting the view of up-to functions as abstractions, moving to systems of equations is easy. As in the case of abstractions, a different up-to function is allowed for each equation.

▶ **Definition 20** (compatible up-to for systems of equations). *Let $(L, \sqsubseteq)$ be a complete lattice and let E be $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$, a system of m equations over L. A* compatible tuple of up-to functions *for E is an m-tuple of monotone functions $\boldsymbol{u}$, with $u_i : L \to L$, satisfying compatibility ($\boldsymbol{u}^\times \circ \boldsymbol{f} \sqsubseteq \boldsymbol{f} \circ \boldsymbol{u}^\times$) with u_i continuous and strict for each $i \in \underline{m}$ such that $\eta_i = \mu$.*

We can then generalise Corollary 19 to systems of equations.

▶ **Theorem 21** (up-to for systems). *Let $(L, \sqsubseteq)$ be a complete lattice and let E be $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$, a system of m equations over L, with solution $\boldsymbol{s} \in L^m$. Let $\boldsymbol{u}$ be a compatible tuple of up-to functions for E and let $\bar{\boldsymbol{u}} = (\bar{u}_1, \ldots, \bar{u}_m)$ be the corresponding tuple of least closures. Let $\boldsymbol{s}'$ and $\bar{\boldsymbol{s}}$ be the solutions of the systems $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{u}^\times(\boldsymbol{x}))$ and $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\bar{\boldsymbol{u}}^\times(\boldsymbol{x}))$, respectively. Then $\boldsymbol{s}' \sqsubseteq \bar{\boldsymbol{s}} = \boldsymbol{s}$. Moreover, if $\boldsymbol{u}$ is extensive then $\boldsymbol{s}' = \boldsymbol{s}$.*

▶ **Example 22** (μ-calculus up-to (bi)similarity). Consider the problem of model-checking the μ-calculus over some transition system with atoms $T = (\mathbb{S}, \to, A)$.

Assuming that we have an a priori knowledge about the similarity relation $\precsim$ over some of the states in T, then, restricting to a suitable fragment of the μ-calculus we can avoid checking the same formula on similar states. This intuition can be captured in the form of an up-to technique, that we refer to as up-to similarity. It is based on an up-to function $\boldsymbol{u}_{\precsim} : \mathbf{2}^{\mathbb{S}} \to \mathbf{2}^{\mathbb{S}}$ defined, for $X \in \mathbf{2}^{\mathbb{S}}$, by $u_{\precsim}(X) = \{s \in \mathbb{S} \mid \exists s' \in X.\ s' \precsim s\}$.

Function $u_{\precsim}$ is monotone, extensive, and idempotent. It is also continuous and strict.

Moreover, $u_{\precsim}$ is a compatible (and thus sound) up-to function for the $\mu\Diamond$-calculus where propositional variables are interpreted as atoms. In fact, $\precsim$ is a simulation (the largest one) and the function $u_{\precsim}$ is the associated abstraction as defined in Example 13, namely $u_{\precsim} = \blacklozenge_{\precsim}$. Therefore, compatibility $u_{\precsim} \circ f \sqsubseteq f \circ u_{\precsim}$ corresponds to condition $\alpha \circ \blacklozenge_{T_C} \circ \gamma \subseteq \blacklozenge_{T_A}$ in Example 13 which has been already observed to coincide with soundness in the sense of

Table 1 The game on the powerset of the basis.

Position	Player	Moves
(b, i)	$\exists$	$\boldsymbol{X}$ s.t. $b \sqsubseteq f_i(\bigsqcup \boldsymbol{X})$
$\boldsymbol{X}$	$\forall$	(b', j) s.t. $b' \in X_j$

Theorem 12 for the operators of the $\mu\Diamond$-calculus. Concerning propositional variables, in Example 13, they were interpreted, in the target transition system, by the abstraction of their interpretation in the source transition system. Since here we have a single transition system and a single interpretation $\rho : \mathit{Prop} \to \mathbf{2}^{\mathbb{S}}$, we must have $\rho(p) = u_{\precsim}(\rho(p))$, i.e., $\rho(p)$ upward-closed with respect to $\precsim$. This automatically holds by the fact that $\precsim$ is a simulation.

Similarly, we can define up-to bisimilarity via the up-to function $u_{\sim}(X) = \{s \in \mathbb{S} \mid \exists s' \in X.\ s \sim s'\}$. As above, one can see that compatibility $u_{\sim} \circ f \sqsubseteq f \circ u_{\sim}$ holds for the full μ-calculus with propositional variables interpreted as atoms. For instance, consider the formula φ in Example 7 and the transition system in Fig. 1a. Using the up-to function $u_{\sim}$ corresponds to working in the bisimilarity quotient in Fig. 1b. Note, however, that when using a local algorithm (see §6.2) the quotient does not need to be actually computed. Rather, only the bisimilarity over the states explored by the searching procedure is possibly exploited.

▶ **Example 23** (bisimilarity up-to transitivity)**.** Consider the problem of checking bisimilarity on a transition system $T = \langle \mathbb{S}, \to \rangle$. A number of well-known sound up-to techniques have been introduced in the literature [37]. As an example, we consider the up-to function $u_{tr} : \mathsf{Rel}(\mathbb{S}) \to \mathsf{Rel}(\mathbb{S})$ performing a single step of transitive closure. It is defined as:

$$u_{tr}(R) = R \circ R = \{(x, y) \mid \exists\, z \in \mathbb{S}.\ (x, z) \in R \ \wedge\ (z, y) \in R\}.$$

It is easy to see that u_{tr} is monotone and compatible with respect to the function $\mathit{bis}_T : \mathsf{Rel}(\mathbb{S}) \to \mathsf{Rel}(\mathbb{S})$ of which bisimilarity is the greatest fixpoint (see Example 10). Since A is deterministic, bisimilarity coincides with language equivalence.

Note that u_{tr} is neither idempotent nor extensive. The corresponding closure $\bar{u}_{tr}$ maps a relation to its (full) transitive closure (this is known to be itself a sound up-to technique, a fact that we can also derive from the compatibility of u_{tr} and Corollary 19).

6 Solving Systems of Equations via Games

In this section, we first provide a characterisation of the solution of a system of fixpoint equations over a complete lattice in terms of a parity game. This generalises a result in [2]. While the original result was limited to continuous lattices, here, exploiting the results on abstraction in §4, we devise a game working for any complete lattice.

The game characterisation opens the way to the development of algorithms for solving the game and thus the associated verification problem. A proper treatment of these aspects is beyond the scope of the present paper, but covered in [3]. Here, in §6.2, we hint at the algorithmic potentials of our theory focusing on the case of a single equation.

6.1 Game Characterization

We show that the solution of a system of equations over a complete lattice can be characterised using a parity game.

▶ **Definition 24** (powerset game). *Let L be a complete lattice with a basis B_L. Given a system E of m equations over L of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$, the corresponding* powerset game *is a parity game, with an existential player $\exists$ and a universal player $\forall$, defined as follows:*

- *The positions of $\exists$ are pairs (b, i) where $b \in B_L$, $i \in \underline{m}$. Those of $\forall$ are tuples of subsets of the basis $\boldsymbol{X} = (X_1, \ldots, X_m) \in (\mathbf{2}^{B_L})^m$.*
- *From position (b, i) the moves of $\exists$ are $\mathbf{E}(b, i) = \{\boldsymbol{X} \mid \boldsymbol{X} \in (\mathbf{2}^{B_L})^m \ \wedge \ b \sqsubseteq f_i(\bigsqcup \boldsymbol{X})\}$.*
- *From position $\boldsymbol{X} \in (\mathbf{2}^{B_L})^m$ the moves of $\forall$ are $\mathbf{A}(\boldsymbol{X}) = \{(b, i) \mid i \in \underline{m} \ \wedge \ b \in X_i\}$.*

The game is schematised in Table 1. For a finite play, the winner is the player who moved last. For an infinite play, let h be the highest index that occurs infinitely often in a pair (b, i). If $\eta_h = \nu$ then $\exists$ wins, else $\forall$ wins.

Interestingly, the correctness and completeness of the game can be proved by exploiting the results in §4. The crucial observation is that there is a Galois insertion between L and the powerset lattice of its basis (which is algebraic hence continuous) $\langle \alpha, \gamma \rangle : \mathbf{2}^{B_L} \to L$ where abstraction α is the join $\alpha(X) = \bigsqcup X$ and concretisation γ takes the lower cone $\gamma(l) = {\downarrow} l \cap B_L$. Then a system of equations over a complete lattice L can be "transferred" to a system of equations over the powerset of the basis $\mathbf{2}^{B_L}$ along such insertion, in a way that the system in L can be seen as a sound and complete abstraction of the one in $\mathbf{2}^{B_L}$.

▶ **Theorem 25** (correctness and completeness). *Let E be a system of m equations over a complete lattice L of the kind $\boldsymbol{x} =_{\boldsymbol{\eta}} \boldsymbol{f}(\boldsymbol{x})$ with solution $\boldsymbol{s}$. For all $b \in B_L$ and $i \in \underline{m}$, $b \sqsubseteq s_i$ iff $\exists$ has a winning strategy from position (b, i).*

6.2 An Algorithmic View

The game theoretical characterisation can be the basis for the development of algorithms, possibly integrating abstraction and up-to techniques, for solving systems of equations. Here we consider local algorithms for the case of a single equation. Our main focus is to provide a general procedure which transcends the verification problem at hand, and also takes advantage of heuristics based on abstractions and up-to techniques. This allows us also to establish a link with some recent work relating abstract interpretation and up-to techniques [6] and exploiting up-to techniques for computing language equivalence on NFAs [8]. While not improving the complexity bounds, our algorithm is still in line with other local algorithms designed for specific settings, such as [8, 21, 22], as they arise as proper instantiations.

An algorithm for general systems is considerably more difficult and cannot be described here due to lack of space (it can be found in the full version of this paper [3]). Here we focus on the special case of a single (greatest) fixpoint equation $x =_\nu f(x)$.

6.2.1 Selections

For a practical use of the game it can be useful to observe that the set of moves of the existential player can be suitably restricted without affecting the completeness of the game, by introducing a notion of selection, similarly to what is done in [2].

Given a lattice L, define a preorder $\sqsubseteq_H$ on $\mathbf{2}^{B_L}$ by letting, for $X, Y \in \mathbf{2}^{B_L}$, $X \sqsubseteq_H Y$ if $\bigsqcup X \sqsubseteq \bigsqcup Y$. (The subscript H comes from the fact that for completely distributive lattices, if B_L is the set of irreducible elements, then $\sqsubseteq_H$ is the "Hoare preorder" [1], requiring that $\forall x \in X. \exists y \in Y. x \sqsubseteq y$.) Observe that $\sqsubseteq_H$ is not antisymmetric. We write $\equiv_H$ for the corresponding equivalence, i.e., $X \equiv_H Y$ when $X \sqsubseteq_H Y \sqsubseteq_H X$.

The moves of player $\exists$ can be ordered by the pointwise extension of $\sqsubseteq_H$, thus leading to the following definition. Since we deal with a single equation, we will omit the indices from the positions of player $\exists$ and write b instead of $(b, 1)$.

▶ **Definition 26** (selection). *Let $x =_\nu f(x)$ be an equation over a complete lattice L, with basis B_L. A* selection *is a function $\sigma : B_L \to \mathbf{2}^{\mathbf{2}^{B_L}}$ such that for all $b \in B_L$ it holds $\uparrow_H \sigma(b) = \mathbf{E}(b)$, i.e. the set of moves of $\exists$ from position b, where $\uparrow_H$ is the upward-closure with respect to $\sqsubseteq_H$.*

This is equivalent to requiring that $\sigma(b) \subseteq \mathbf{E}(b)$ and for each $X \in \mathbf{E}(b)$ there exists $Y \in \sigma(b)$ such that $\bigsqcup Y \sqsubseteq \bigsqcup X$.

For the case of a single fixpoint equation it is easy to see that Theorem 25 continues to hold if we restrict the moves of player $\exists$ to those prescribed by a selection.

▶ **Theorem 27** (game with selections). *Let $x =_\nu f(x)$ be an equation over a complete lattice L with solution s. For all $b \in B_L$, it holds that $b \sqsubseteq s$ iff $\exists$ has a winning strategy from position b in the game restricted to selections.*

6.2.2 Local Algorithm for a Special Case

In this section we assume that $f : L \to L$ is some fixed function that preserves non-empty meets, i.e., for $X \neq \emptyset$, $f(\bigsqcap X) = \bigsqcap f(X)$. This is equivalent to asking $f(x) = f^*(x) \sqcap c$ for some $c \in L$ (just take $c = f(\top)$), with f^* being a right adjoint of a map f_*, a setting that has been studied also in [6]. We will call a function satisfying this assumption a *deterministic function*. Note that the adjunction $\langle f_*, f^* \rangle$ is completely orthogonal to the adjunctions (Galois connections) studied so far.

▶ **Example 28.** For a simple example adopted from [8], consider a deterministic finite automaton $A = (Q, \Sigma, \delta, F)$, where Q is a finite set of states, Σ is a finite alphabet, $\delta : Q \times \Sigma \to Q$ is the transition function and $F \subseteq Q$ is the set of final states. Since A is deterministic, language equivalence coincides with bisimilarity. Consider the lattice of relations $L = (\mathbf{2}^{Q \times Q}, \sqsubseteq)$ with basis $B_L = \{\{(q_1, q_2)\} \mid q_1, q_2 \in Q\}$. The behaviour map, having bisimilarity as largest fixpoint, is $f : \mathbf{2}^{Q \times Q} \to \mathbf{2}^{Q \times Q}$ defined as $f(R) = f^*(R) \cap C$ where $f^*(R) = \{(q_1, q_2) \mid \forall a \in \Sigma.\, (\delta(q_1, a), \delta(q_2, a)) \in R\}$ with $C = \{(q_1, q_2) \mid q_1 \in F \iff q_2 \in F\}$. The left adjoint is $f_*(R) = \{(\delta(q_1, a), \delta(q_2, a)) \mid (q_1, q_2) \in R, a \in \Sigma\}$.

Given two states $q_1, q_2 \in R$, we want to decide whether $(q_1, q_2) \in S$, where S is bisimilarity, the solution of the greatest fixpoint equation $R =_\nu f(R)$.

We first observe that for deterministic functions we can take a very simple selection.

▶ **Lemma 29** (selection). *Let L be a complete lattice with basis B_L, and let $f : L \to L$ be a deterministic function, i.e., $f(x) = f^*(x) \sqcap c$ for some $c \in L$ and $\langle f_*, f^* \rangle : L \to L$ a Galois connection. A selection $\sigma : B_L \to \mathbf{2}^{\mathbf{2}^{B_L}}$ for $x =_\nu f(x)$ can be defined, for $b \in B_L$, as:*

$$\sigma(b) = \begin{cases} \{X\} \text{ with } X \subseteq B_L \text{ s.t. } X \equiv_H {\downarrow} f_*(b) \cap B_L & \text{when } b \sqsubseteq c \\ \emptyset & \text{otherwise} \end{cases}$$

Observe that there might be several choices for $X \subseteq B_L$: one that always works is $X = {\downarrow} f_*(b) \cap B_L$, but subsets $X \subseteq {\downarrow} f_*(b) \cap B_L$ are also feasible, as long as $\bigsqcup X = f_*(b)$. In Example 28, given $\{(q_1, q_2)\} \in B_L$, we can define $\sigma(\{(q_1, q_2)\}) = \{\{\{(q_1', q_2')\} \mid (q_1', q_2') \in f_*(\{(q_1, q_2)\})\}\} = \{\{\{(\delta(q_1, a), \delta(q_2, a))\} \mid a \in \Sigma\}\}$.

By Lemma 29, in the game for $x =_\nu f(x)$, either the existential player is stuck or she has a best move. As a consequence, the game in §6.1 can be simplified. Let B_L be any basis for L such that $\bot \notin B_L$. The moves of player $\exists$ are deterministic, governed by σ, and only player $\forall$ has choices when exploring the elements included in such moves.

For checking whether $b \sqsubseteq \nu f$, for some $b \in B_L$, the game starts from position b. Then, at a generic position b', we do the following:

1. if $b' \not\sqsubseteq c$ then $\sigma(b') = \emptyset$ and $\exists$ loses;
2. otherwise, $\exists$ has to play the only element in $\sigma(b') = \{X\}$
   a. if $f_*(b') = \bot$ then take $X = \emptyset$; hence $\exists$ wins since $\forall$ has no moves;
   b. if instead $f_*(b') \neq \bot$, we can take $X \equiv_H {\downarrow} f_*(b') \cap B_L$ and thus player $\forall$ can play any $b'' \in X$ and the game continues.

Player $\exists$ wins the game iff no losing position for her ($b' \not\sqsubseteq c$) is encountered in the exploration. When a losing position for $\exists$ is encountered we immediately know that $\forall$ wins.

The game can be further simplified by observing that, if W denotes the set of positions already visited during the exploration, whenever, at a position b', we have $b' \sqsubseteq \bigsqcup W$ then $\exists$ wins from b' as long as she wins from all the positions in W. This leads to the local algorithm outlined in List. 1, whose proof of correctness formalises the arguments above. The procedure `Explore` allows to check if $b \sqsubseteq \nu f = \nu(f^* \sqcap c)$ by invoking `Explore(`b`,`$\emptyset$`)`, which returns `true` if and only if player $\exists$ wins in the simplified game.

Listing 1 Local algorithm for the simplified game.

```
Explore(b',W):
  if b' ⋢ c then return false;
  else if b' ⊑ ⨆W then return true;
  else take X ⊆ B_L s.t. X ≡_H ↓f_*(b') ∩ B_L;
       return ∧_{b''∈X} Explore(b'',W ∪ {b'});
```

▶ **Theorem 30** (correctness and completeness of the simplified game). *Let L be a complete lattice with basis $B_L \subseteq L \setminus \{\bot\}$, and let $f : L \to L$ be a deterministic function, i.e., $f(x) = f^*(x) \sqcap c$ for some $c \in L$ and $\langle f_*, f^* \rangle : L \to L$ a Galois connection. Then, for all $b \in B_L$, $b \sqsubseteq \nu f$ iff the invocation* `Explore(`b`,`$\emptyset$`)` *returns* `true`.

For instance, for Example 28, the local algorithm of List. 1 works as follows: for checking whether $\{(q_1, q_2)\}$ is dominated by the solution, i.e., states q_1 and q_2 are bisimilar, one starts from $\{(q_1, q_2)\}$. At position $\{(q_1', q_2')\}$, if one state is final and the other is not, $\exists$ loses. If the pair has been already explored, the branch is not considered. Otherwise, the pairs arising as a-successors $\{(q_1', q_2')\}$ are explored. If no losing position is found, the exploration finishes (recall that there are finitely many states) and $\bigcup W$ is a bisimulation including (q_1, q_2).

Observe that when the basis is $B_L = L \setminus \{\bot\}$, the game becomes deterministic also for player $\forall$: in List. 1, when $f_*(b') \neq \bot$ one can take $X = \{\{f_*(b')\}\}$, otherwise $X = \emptyset$. Therefore, since f_* is a left adjoint and thus continuous, if we take the set S of all the positions generated during the exploration (i.e., W with the addition of the last position, for finite games) then $\bigsqcup S = \bigsqcup_i f_*^i(b)$ is the least fixpoint of f_* above b, which in turn coincides with the least fixpoint of $f^* \sqcup b$. This establishes a direct link with [6] which shows that for $b \in L$ it holds that $\mu(f^* \sqcup b) \sqsubseteq c$ iff $b \sqsubseteq \nu(f^* \sqcap c) = \nu f$.

Furthermore, we can bring up-to techniques into the picture: given an up-to function u we can modify the procedure in List. 1 by replacing the winning condition for $\exists$, that is, $b' \sqsubseteq \bigsqcup W$, by $b' \sqsubseteq u(\bigsqcup W)$. The procedure remains clearly complete and it is also correct due to Theorem 21. This allows us to cover the algorithm in [8] which checks language equivalence for non-deterministic automata. It performs on-the-fly determinization and constructs a bisimulation up-to congruence on the determinized automaton. More concretely, it tries to construct a bisimulation relation for the determinized automaton (along the lines of Example 28) and remembers pairs (X_1, X_2) of sets of states seen so far in a relation W (as explained in the algorithm in List. 1). Once a pair (Y_1, Y_2) is encountered that is contained in the congruence closure of W (the least equivalence, closed under union, that contains W), one can stop exploring this branch. A more detailed comparison can be found in Appendix A.

7 Conclusion

Our contribution is based on the notion of approximation as formalised in abstract interpretation [13, 14]. Due to the intimate connection of Galois connections and closure functions, there is a close correspondence with up-to techniques for enhancing coinduction proofs [35, 37], originally developed for CCS [32]. However, as far as we know, recent research has only started to explore this connection: [6] explains the relation between sound up-to techniques and complete abstract domains in the setting where the semantic function has an adjoint. This adjunction or Galois connection plays a different role than the abstractions: it gives the existential player a unique best move, a concept explored in §6.2.2.

Fixpoint equation systems largely derive their interest from μ-calculus model-checking [9]. Evaluating μ-calculus formulae on a transition system can be reduced to solving a parity game and the exact complexity of this task is still open. Progress measures, introduced in [25], allow one to solve parity games with a complexity which is polynomial in the number of states and exponential in (half of) the alternation depth of the formula. Recently quasi-polynomial algorithms for parity games [10, 26, 29] have been devised. Instead of improving the complexity bounds, our aim here is to introduce heuristics, based on an on-the-fly algorithm and up-to functions that are known to achieve good efficiency in practice.

Many papers deal with abstraction in the setting of μ-calculus model checking. We noted that the results on simulation-based abstraction in [30] can be obtained as an instance of our framework. The abstraction of the μ-calculus along a Galois connection and its soundness is discussed in [4]. A general framework for abstract interpretation of temporal calculi and logics is developed in [15]. In particular, an abstract calculus for expressing nested fixpoint expressions is studied, parametric with respect to the basic operators. The calculus is interpreted over complete boolean lattices and conditions ensuring the soundness and the completeness of the abstraction along a Galois connection are singled out. Such results are closely related to those in Section 4. The main differences reside in the fact that we work with general complete lattices, rather than with boolean lattices. In addition, we treat separately soundness and completeness, and, in order to establish a connection with up-to techniques, we distinguish two forms of completeness (for the abstraction and for the concretisation).

We showed – for a special case – how on-the-fly algorithms inspired by [8, 6, 21, 22] for a single (greatest) fixpoint equation can be adapted to the case of general lattices. For the general case of arbitrary fixpoint equation systems a considerably more complex generalisation along the lines of [44] is possible, but omitted due to lack of space.

The use of assumptions as stopping conditions in the algorithm is reminiscent of parameterized coinduction [43, 23], closely related to up-to-techniques, as spelled out in [36].

The notion of progress measures that has been studied in [2] can be adapted to the game for arbitrary complete (rather than just continuous) lattices, introduced in this paper. A first natural question is whether the on-the-fly algorithm arises as an instance of the single equation algorithm instantiated with the progress measure fixpoint equation.

With respect to the applications, we believe that our case study on abstractions respectively simulations for μ-calculus model-checking can also be generalised to modal respectively mixed transition systems [40, 17, 28] or to abstraction for the full μ-calculus as studied in [19] by combining both under- and over-approximations. Furthermore, we plan to further study over-approximations for fixpoint equations over the reals, closely connected to probabilistic logics. In particular, we will investigate under which circumstances one can obtain guarantees to be close to the exact solution or to compute the exact solution directly. Another interesting area is the use of up-to techniques for behavioural metrics [7].

References

1 Samson Abramsky and Achim Jung. Domain theory. In Samson Abramsky, Dov Gabbay, and Thomas Stephen Edward Maibaum, editors, *Handbook of Logic in Computer Science*, pages 1–168. Oxford University Press, 1994.

2 Paolo Baldan, Barbara König, Christina Mika-Michalski, and Tommaso Padoan. Fixpoint games in continuous lattices. In Stephanie Weirich, editor, *Proc. of POPL'19*, volume 3, pages 26:1–26:29. ACM, 2019.

3 Paolo Baldan, Barbara König, and Tommaso Padoan. Abstraction, up-to techniques and games for systems of fixpoint equations, 2020. `arXiv:2003.08877`.

4 Gourinath Banda and John P. Gallagher. Constraint-based abstract semantics for temporal logic: A direct approach to design and implementation. In Edmund M. Clarke and Andrei Voronkov, editors, *LPAR 2010*, volume 6355 of *Lecture Notes in Computer Science*, pages 27–45. Springer, 2010.

5 Andrea Bianco and Luca de Alfaro. Model checking of probabalistic and nondeterministic systems. In *Proc. of FSTTCS '95*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer, 1995.

6 Filippo Bonchi, Pierre Ganty, Roberto Giacobazzi, and Dusko Pavlovic. Sound up-to techniques and complete abstract domains. In *Proc. of LICS '18*, pages 175–184. ACM, 2018.

7 Filippo Bonchi, Barbara König, and Daniela Petrişan. Up-to techniques for behavioural metrics via fibrations. In *Proc. of CONCUR '18*, volume 118 of *LIPIcs*, pages 17:1–17:17. Schloss Dagstuhl – Leibniz Center for Informatics, 2018.

8 Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In *Proc. of POPL '13*, pages 457–468. ACM, 2013.

9 Julian Bradfield and Igor Walukiewicz. The mu-calculus and model checking. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 871–919. Springer, 2018.

10 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proc. of STOC '17*, pages 252–263. ACM, 2017.

11 Rance Cleaveland, Marion Klein, and Bernhard Steffen. Faster model checking for the modal mu-calculus. In *Proc. of CAV 1992*, volume 663 of *Lecture Notes in Computer Science*, pages 410–422. Springer, 1992.

12 Patrick Cousot. Partial completeness of abstract fixpoint checking. In Berthe Y. Choueiry and Toby Walsh, editors, *Proceedings of the Fourth International Symposium on Abstraction, Reformulations and Approximation, SARA'2000*, volume 1864 of *Lecture Notes in Computer Science*, pages 1–25. Springer, 26–29 July 2000.

13 Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. of POPL '77*, pages 238–252. ACM, 1977.

14 Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Proc. of POPL '79*, pages 269–282. ACM, 1979.

15 Patrick Cousot and Radhia Cousot. Temporal abstract interpretation. In Mark N. Wegman and Thomas W. Reps, editors, *Proc. of POPL '00*, pages 12–25. ACM, 2000.

16 Radhia Cousot and Patrick Cousot. Constructive versions of tarski's fixed point theorems. *Pacific Journal of Mathematics*, 82(1):43–57, 1979.

17 Dennis Dams, Rob Gerth, and Orna Grumberg. Abstract interpretation of reactive systems. *ACM Trans. Program. Lang. Syst.*, 19(2):253–291, 1997.

18 Roberto Giacobazzi, Francesco Ranzato, and Francesca Scozzari. Making abstract interpretations complete. *Journal of the ACM*, 47(2):361–416, 2000.

19 Orna Grumberg, Martin Lange, Martin Leucker, and Sharon Shoham. When not losing is better than winning: Abstraction and refinement for the full mu-calculus. *Information and Computation*, 205(8):1130–1148, 2007.

20 Ichiro Hasuo, Shunsuke Shimizu, and Corina Cîrstea. Lattice-theoretic progress measures and coalgebraic model checking. In *Proc. of POPL '16*, pages 718–732. ACM, 2016.

21 Daniel Hirschkoff. Automatically proving up to bisimulation. In *Proc. of MFCS '98 Workshop on Concurrency*, number 18 in Electronic Notes in Theoretical Computer Science, pages 75–89. Elsevier, 1998.

22 Daniel Hirschkoff. *Mise en oeuvre de preuves de bisimulation.* PhD thesis, Ecole Nationale des Ponts et Chaussées, 1999.

23 Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. The power of parameterization in coinductive proof. In *Proc. of POPL '13*, pages 193–206. ACM, 2013.

24 Michael Huth and Marta Kwiatkowska. Quantitative analysis and model checking. In *Proc. of LICS '97*, pages 111–122. IEEE, 1997.

25 Marcin Jurdziński. Small progress measures for solving parity games. In *Proc. of STACS '00*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2000.

26 Marcin Jurdzinski and Ranko Lazic. Succinct progress measures for solving parity games. In *Proc. of LICS '17*, pages 1–9. ACM/IEEE, 2017.

27 Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983.

28 Kim Guldstrand Larsen and Bent Thomsen. A modal process logic. In *Proc. of LICS '88*, pages 203–210. IEEE, 1988.

29 Karoliina Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In *Proc. of LICS '18*, pages 639–648. ACM/IEEE, 2018.

30 Claire Loiseaux, Susanne Graf, Joseph Sifakis, Ahmed Bouajjani, and Saddek Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6:1–35, 1995.

31 Annabelle McIver and Carroll Morgan. Results on the quantitative μ-calculus qmμ. *ACM Trans. Comp. Log.*, 8(1:3), 2007.

32 Robin Milner. *Communication and Concurrency.* Prentice Hall, 1989.

33 Antoine Miné. Tutorial on static inference of numeric invariants by abstract interpretation. *Foundations and Trends in Programming Languages*, 4(3-4):120–372, 2017.

34 Matteo Mio and Alex Simpson. Łukasiewicz μ-calculus. *Fundamenta Informaticae*, 150(3-4):317–346, 2017.

35 Damien Pous. Complete lattices and up-to techniques. In *Proc. of APLAS '07*, pages 351–366. Springer, 2007. LNCS 4807.

36 Damien Pous. Coinduction all the way up. In *Proc. of LICS'16*, pages 307–316. ACM, 2016.

37 Damien Pous and Davide Sangiorgi. Enhancements of the bisimulation proof method. In Davide Sangiorgi and Jan Rutten, editors, *Advanced Topics in Bisimulation and Coinduction.* Cambridge University Press, 2011.

38 Davide Sangiorgi. *Introduction to Bisimulation and Coinduction.* Cambridge University Press, 2011.

39 Davide Sangiorgi and Robin Milner. The problem of "weak bisimulation up to". In W.R. Cleaveland, editor, *Proc. of CONCUR'92*, pages 32–46. Springer, 1992.

40 David A. Schmidt. Binary relations for abstraction and refinement. In *Workshop on Refinement and Abstraction.* Elsevier Electronic, 2000.

41 Dana Scott. Continuous lattices. In F. W. Lawvere, editor, *Toposes, Algebraic Geometry and Logic*, Lecture Notes in Mathematics, pages 97–136. Springer, 1972.

42 Helmut Seidl. Fast and simple nested fixpoints. *Information Processing Letters*, 59(6):303–308, 1996.

43 David Sprunger and Lawrence S. Moss. Precongruences and parametrized coinduction for logics for behavioral equivalence. In *Proc. of CALCO '17*, volume 72 of *LIPIcs*, pages 23:1–23:15. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017.

44 Perdita Stevens and Colin Stirling. Practical model-checking using games. In *Proc. of TACAS '98*, volume 1384 of *Lecture Notes in Computer Science*, pages 85–101. Springer, 1998.

45 Colin Stirling. Local model checking games. In *Proc. of CONCUR '95*, volume 962 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 1995.

46 Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

A Comparison to the Bonchi/Pous Algorithm

In a seminal paper [8] Bonchi and Pous revisited the question of checking language equivalence for non-deterministic automata and presented an algorithm based on an up-to congruence technique that behaves very well in practice.

We will here give a short description of this algorithm and then explain how it arises as a special case of the algorithm developed in §6.2.2.

We are given a non-deterministic finite automaton (Q, Σ, δ, F), where Q is the finite set of states, Σ is the finite alphabet, $\delta\colon Q \times \Sigma \to \mathbf{2}^Q$ is the transition function and $F \subseteq Q$ is the set of final states. Note that we omit initial states. Given $a \in \Sigma$, $X \subseteq Q$ we define $\delta_a(X) = \bigcup_{q\in X} \delta(q, a)$.

Given $q_1, q_2 \in Q$, the aim is to show whether q_1, q_2 accept the same language (in the standard sense).

In order to do this, the algorithm performs an on-the-fly determinization and constructs a bisimulation relation $R \subseteq \mathbf{2}^Q \times \mathbf{2}^Q$ on the determinized automaton. This relation has to satisfy the following properties:

- $\{q_1\}\, R\, \{q_2\}$
- Whenever $X_1\, R\, X_2$, then
  - $\delta_a(X_1)\, R\, \delta_a(X_2)$ for all $a \in \Sigma$ (*transfer property*)
  - and $X_1 \cap F \neq \emptyset \iff X_2 \cap F \neq \emptyset$ (*one set is accepting iff the other is accepting*)

Due to the up-to technique there is no need to fully enumerate R. Instead in the second item above, it suffices to show that $\delta_a(X_1)\, c(R)\, \delta_a(X_2)$ where $c(R)$ is the congruence closure of R, i.e., the least relation R' containing R that is an equivalence and satisfies that $X_1\, R\, X_2$ implies $X_1 \cup X\, R\, X_2 \cup X$ (for $X_1, X_2, X \subseteq Q$). A major contribution of [8] is an algorithm for efficiently checking whether two given sets are in the congruence closure of a given relation. Here we will simply assume that this procedure is given and use it as a black box.

We will now translate this into our setting: the lattice is $L = \mathbf{2}^{\mathbf{2}^Q \times \mathbf{2}^Q}$ (the lattice of all relations over the powerset of states) with inclusion as partial order. The basis B consists of all singletons $\{(X_1, X_2)\}$ where $X_1, X_2 \subseteq Q$. That is, we consider the setting of of §6.2.2.

The behaviour map f is given as follows: $f(R) = f^*(R) \cap C$ where

$$\begin{aligned} f^*(R) &= \{(X_1, X_2) \mid (\delta_a(X_1), \delta_a(X_2)) \in R \text{ for all } a \in \Sigma\} \\ C &= \{(X_1, X_2) \mid X_1 \cap F = \emptyset \iff X_2 \cap F = \emptyset\} \end{aligned}$$

We want to solve a single fixpoint equation $R =_\nu f(R)$ where we are interested in the greatest fixpoint. In particular, we want to check whether $(Q_1, Q_2) \in R$ (where $Q_1 = \{q_1\}$, $Q_2 = \{q_2\}$) or alternatively $I = \{(Q_1, Q_2)\} \subseteq R$.

Since we have determinized the automaton, f^* has a left adjoint f_*, given as

$$f_*(R) = \{(\delta_a(X_1), \delta_a(X_2)) \mid (X_1, X_2) \in R, a \in \Sigma\}.$$

Now we can start exploring the game positions. Starting with $I = \{(Q_1, Q_2)\} \subseteq F$, the only move of $\exists$ is to play $\{\{(X_1, X_2)\} \mid (X_1, X_2) \in f_*(I)\}$, then it is the turn of $\forall$ who can choose any singleton set $\{(X_1, X_2)\}$ and one has to explore all those singletons. This continues until one encounters a singleton $\{(X_1, X_2)\} \not\subseteq C$ (which implies that $\exists$ has no move and loses) or

one finds a set $\{(X_1, X_2)\}$ where one can cut off a branch due to the up-to technique – more concretely $(X_1, X_2) \in c(W)$ where W is the collection of all pairs visited so far on all paths and $c(W)$ is its congruence closure. One can conclude that $\exists$ wins if all encountered pairs are in C. This is a straightforward instance of the more general algorithm, enriched with an up-to technique, as explained in §6.2.2.

Reaching Your Goal Optimally by Playing at Random with No Memory

Benjamin Monmege
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
benjamin.monmege@univ-amu.fr

Julie Parreaux
ENS Rennes, France
julie.parreaux@ens-rennes.fr

Pierre-Alain Reynier
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
pierre-alain.reynier@univ-amu.fr

Abstract

Shortest-path games are two-player zero-sum games played on a graph equipped with integer weights. One player, that we call Min, wants to reach a target set of states while minimising the total weight, and the other one has an antagonistic objective. This combination of a qualitative reachability objective and a quantitative total-payoff objective is one of the simplest settings where Min needs memory (pseudo-polynomial in the weights) to play optimally. In this article, we aim at studying a tradeoff allowing Min to play at random, but using no memory. We show that Min can achieve the same optimal value in both cases. In particular, we compute a randomised memoryless ε-optimal strategy when it exists, where probabilities are parametrised by ε. We also show that for some games, no optimal randomised strategies exist. We then characterise, and decide in polynomial time, the class of games admitting an optimal randomised memoryless strategy.

2012 ACM Subject Classification Software and its engineering → Formal software verification; Theory of computation → Algorithmic game theory

Keywords and phrases Weighted games, Algorithmic game theory, Randomisation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.26

Funding Benjamin Monmege and Pierre-Alain Reynier are partly funded by ANR project Ticktac (ANR-18-CE40-0015).

1 Introduction

Game theory is now an established model in the computer-aided design of correct-by-construction programs. Two players, the controller and an environment, are fighting one against the other in a zero-sum game played on a graph of all possible configurations. A winning strategy for the controller results in a correct program, while the environment is a player modelling all uncontrollable events that the program must face. Many possible objectives have been studied in such two-player zero-sum games played on graphs: reachability, safety, repeated reachability, and even all possible ω-regular objectives [10].

Apart from such *qualitative* objectives, more *quantitative* ones are useful in order to select a particular strategy among all the ones that are correct with respect to a qualitative objective. Some metrics of interest, mostly studied in the quantitative game theory literature, are mean-payoff, discounted-payoff, or total-payoff. All these objectives have in common that both players have strategies using no memory or randomness to win or play optimally [9].

Combining quantitative and qualitative objectives, enabling to select a good strategy among the valid ones for the selected metrics, often leads to the need of memory to play optimally. One of the simplest combinations showing this consists in the shortest-path

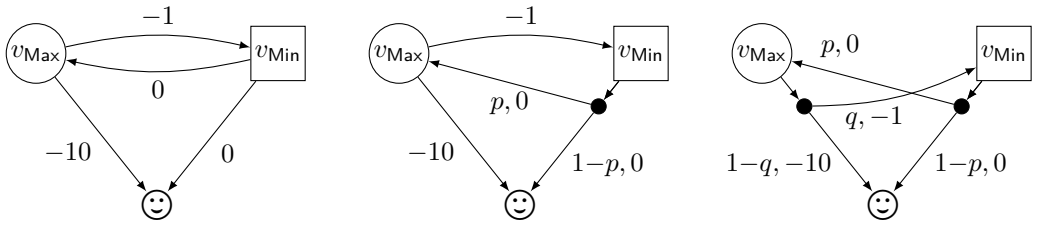


Figure 1 On the left, a shortest-path game, where Min requires memory to play optimally. In the middle, the Markov Decision Process obtained when letting Min play at random, with a parametric probability $p \in (0, 1)$. On the right, the Markov Chain obtained when Max plays along a memoryless randomised strategy, with a parametric probability $q \in [0, 1]$.

games combining a reachability objective with a total-payoff quantitative objective (studied in [11, 4] under the name of *min-cost reachability games*). Another case of interest is the combination of a parity qualitative objective (modelling every possible ω-regular condition), with a mean-payoff objective (aiming for a controller of good quality in the average long-run), where controllers need memory, and even infinite memory, to play optimally [6].

It is often crucial to enable randomisation in the strategies. For instance, Nash equilibria are only ensured to exist in matrix games (like rock-paper-scissors) when players can play at random [13]. In the context of games on graphs, a player may choose, depending on the current history, the probability distribution on the successors. In contrast, strategies that do not use randomisation are called *deterministic* (we sometimes say *pure*).

In this article, we will focus on shortest-path games, as the one depicted on the left of Figure 1. The objective of Min is to reach vertex ☺, while minimising the total weight. Let us consider the vertex v_{Min} as initial. Player Min could reach directly ☺, thus leading to a payoff of 0. But he can also choose to go to v_{Max}, in which case Max either jumps directly in ☺ (leading to a beneficial payoff -10), or comes back to v_{Min}, but having already capitalised a total payoff -1. We can continue this way *ad libitum* until Min is satisfied (at least 10 times) and jumps to ☺. This guarantees a value at most -10 for Min when starting in v_{Min}. Reciprocally, Max can guarantee a payoff at least -10 by directly jumping into ☺ when she must play for the first time. Thus, the optimal value is -10 when starting from v_{Min} or v_{Max}. However, Min cannot achieve this optimal value by playing *without memory* (we sometimes say *positionally*), since it either results in a total-payoff 0 (directly going to the target) or Max has the opportunity to keep Min in the negative cycle for ever, thus never reaching the target. Therefore, Min needs memory to play optimally. He can do so by playing a *switching strategy*, turning in the negative cycle long enough so that no matter how he reaches the target finally, the value he gets as a payoff is lower than the optimal value. This strategy uses pseudo-polynomial memory with respect to the weights of the game graph.

In this example, such a switching strategy can be *mimicked* using randomisation only (and no memory), Min deciding to go to v_{Max} with high probability $p < 1$ and to go to the target vertex with the remaining low probability $1 - p > 0$ (we enforce this probability to be positive, in order to reach the target with probability 1, no matter how the opponent is playing). The resulting *Markov Decision Process (MDP)* is depicted in the middle of Figure 1. The shortest path problem in such MDPs has been thoroughly studied in [2], where it is proved that Max does not require memory to play optimally. Denoting by q the probability that Max jumps in v_{Min} in its memoryless strategy, we obtain the *Markov chain (MC)* on the right of Figure 1. We can compute (see Example 4) the expected value in this MC, as well as the best strategy for both players: in the overall, the optimal value remains -10, even if Min no longer has an optimal strategy. He rather has an ε-optimal strategy, consisting in choosing $p = 1 - \varepsilon/10$ that ensures a value at most $-10 + \varepsilon$.

This article thus aims at studying the tradeoff between memory and randomisation in strategies for shortest-path games. The study is only interesting in the presence of both positive and negative weights, since both players have optimal memoryless deterministic strategies when the graph contains only non-negative weights [11]. The tradeoff between memory and randomisation has already been investigated in many classes of games where memory is required to win or play optimally. This is for instance the case for qualitative games like Street or Müller games thoroughly studied (with and without randomness in the arena) in [5]. The study has been extended to timed games [7] where the goal is to use as little information as possible about the precise values of real-time clocks. Memory or randomness is also crucial in multi-dimensional objectives [8]: for instance, in mean-payoff parity games, if there exists a deterministic finite-memory winning strategy, then there exists a randomised memoryless almost-sure winning strategy.

In contrast to previous work, we show that deterministic memory and memoryless randomisation provide the same power to Min. We leave the combination of memory and randomisation for future work, as explained in the discussion. After a presentation of the model of shortest-path games in Section 2, we show in Section 3 how the previous simulation of memory with randomisation can be performed for all shortest-path games. The general case is much more challenging, in particular in the presence of positive cycles in the graph, that Min cannot avoid in general. Section 4 shows reciprocally how to mimic randomised strategies with memory only. Section 5 studies the optimality of randomised strategies. Indeed, all shortest-path games admit an optimal deterministic strategy for both players, but Min may require memory to play optimally (even with randomisation allowed). We thus characterises the shortest-path games in which Min admits an optimal memoryless strategy, and decide this characterisation in polynomial time.

2 Shortest-path games: deterministic or memoryless strategies

In this section, we formally introduce the shortest-path games we consider throughout the article, as already thoroughly studied in [4] under the name of *min-cost reachability games*. We denote by $\mathbb{Z}$ the set of integers, and $\mathbb{Z}_\infty = \mathbb{Z} \cup \{-\infty, +\infty\}$. For a finite set V, we denote by $\Delta(V)$ the set of *distributions* over V, that are all mappings $\delta \colon V \to [0,1]$ such that $\sum_{v \in V} \delta(v) = 1$. The support of a distribution δ is the set $\{v \in V \mid \delta(v) > 0\}$, denoted by $\mathrm{supp}(\delta)$. A Dirac distribution is a distribution with a singleton support: the Dirac distribution of support $\{v\}$ is denoted by Dirac_v.

We consider two-player turn-based games played on weighted graphs and denote the players by Max and Min. Formally, a *shortest-path game* (SPG) is a tuple $\langle V_{\mathsf{Max}}, V_{\mathsf{Min}}, E, \omega, T \rangle$ where $V := V_{\mathsf{Max}} \uplus V_{\mathsf{Min}} \uplus T$ is a finite set of vertices partitioned into the sets V_{Max} and V_{Min} of Max and Min respectively, and a set T of target vertices, $E \subseteq V \setminus T \times V$ is a set of *directed edges*, and $\omega \colon E \to \mathbb{Z}$ is the *weight function*, associating an integer weight with each edge. In the drawings, Max vertices are depicted by circles; Min vertices by rectangles. For every vertex $v \in V$, the set of successors of v with respect to E is denoted by $E(v) = \{v' \in V \mid (v, v') \in E\}$. Without loss of generality, we assume that non-target vertices are deadlock-free, i.e. for all vertices $v \in V \setminus T$, $E(v) \neq \emptyset$. Finally, throughout this article, we let $W = \max_{(v,v') \in E} |\omega(v, v')|$ be the greatest edge weight (in absolute value) in the arena. A *finite play* is a finite sequence of vertices $\pi = v_0 v_1 \cdots v_k \in V^*$ such that for all $0 \leqslant i < k$, $(v_i, v_{i+1}) \in E$. Its *total weight* is the sum $\sum_{i=0}^{k-1} \omega(v_i, v_{i+1})$ of its weights. A *play* is either a finite play ending in a target vertex, or an infinite sequence of vertices $\pi = v_0 v_1 \cdots$ avoiding the target such that every finite prefix $v_0 \cdots v_k$, denoted by $\pi[k]$, is a finite play.

The total-payoff of a play $\pi = v_0 v_1 \dots$ is given by $\mathbf{TP}(\pi) = +\infty$ if the play is infinite (and therefore avoids T), or by the total weight $\mathbf{TP}(\pi) = \sum_{i=0}^{k-1} \omega(v_i, v_{i+1})$ if $\pi = v_0 v_1 \cdots v_k$ is a finite play ending in a vertex $v_k \in T$ (for the first time).

A *strategy* for Min over an arena $\mathcal{G} = \langle V_{\mathsf{Max}}, V_{\mathsf{Min}}, E, \omega, T\rangle$ is a mapping $\sigma\colon V^* V_{\mathsf{Min}} \to \Delta(V)$ such that for all sequences $\pi = v_0 \cdots v_k$ with $v_k \in V_{\mathsf{Min}}$, the support of the distribution $\sigma(\pi)$ is included in $E(v_k)$. A play or finite play $\pi = v_0 v_1 \cdots$ conforms to the strategy σ if for all k such that $v_k \in V_{\mathsf{Min}}$, we have that $\sigma(\pi[k])(v_{k+1}) > 0$. A similar definition allows one to define strategies $\tau\colon V^* V_{\mathsf{Max}} \to \Delta(V)$ for Max, and plays conforming to them.

A strategy σ is *deterministic* (or *pure*) if for all finite plays π, $\sigma(\pi)$ is a Dirac distribution: in this case, we let $\sigma(\pi)$ denote the unique vertex in the support of this Dirac distribution. We let $\mathsf{d}\Sigma_{\mathsf{Min}}$ and $\mathsf{d}\Sigma_{\mathsf{Max}}$ be the deterministic strategies of players Min and Max, respectively. A strategy σ is *memoryless* if for all finite plays π, π', and all vertices $v \in V$, we have that $\sigma(\pi v) = \sigma(\pi' v)$ for all $v \in V$. We let $\mathsf{m}\Sigma_{\mathsf{Min}}$ and $\mathsf{m}\Sigma_{\mathsf{Max}}$ be the memoryless strategies of players Min and Max, respectively. To distinguish them easily from deterministic strategies, we will denote a memoryless strategy of Min using letter ρ (for *r*andom).

In this article, we focus on deterministic strategies on the one hand, and memoryless strategies on the other hand. Even if the notion of values that we will now introduce could be defined in a more general setting, we prefer to give two simpler definitions in the two separate cases, for the sake of clarity.

2.1 Deterministic strategies

In case of deterministic strategies, for all vertices v, we let $\mathsf{Play}(v, \sigma, \tau)$ be the unique play conforming to strategies σ and τ of Min and Max, respectively, and starting in v. This unique play has a payoff $\mathbf{TP}(\mathsf{Play}(v, \sigma, \tau))$. Then, we define the value of strategies σ and τ by letting for all v,

$$\mathsf{dVal}^{\sigma}(v) = \sup_{\tau' \in \mathsf{d}\Sigma_{\mathsf{Max}}} \mathbf{TP}(\mathsf{Play}(v, \tau', \sigma)) \qquad \text{and} \qquad \mathsf{dVal}^{\tau}(v) = \inf_{\sigma' \in \mathsf{d}\Sigma_{\mathsf{Min}}} \mathbf{TP}(\mathsf{Play}(v, \tau, \sigma'))$$

Finally, the game itself has two possible values, an *upper value* describing the best Min can hope for, and a *lower value* describing the best Max can hope for: for all vertices v,

$$\overline{\mathsf{dVal}}(v) = \inf_{\sigma \in \mathsf{d}\Sigma_{\mathsf{Min}}} \mathsf{dVal}^{\sigma}(v) \qquad \text{and} \qquad \underline{\mathsf{dVal}}(v) = \sup_{\tau \in \mathsf{d}\Sigma_{\mathsf{Max}}} \mathsf{dVal}^{\tau}(v)$$

We may easily show that $\underline{\mathsf{dVal}}(v) \leqslant \overline{\mathsf{dVal}}(v)$ for all initial vertices v. In [3, Theorem 1], shortest-path games are shown to be determined when both players use deterministic strategies, i.e. $\underline{\mathsf{dVal}}(v) = \overline{\mathsf{dVal}}(v)$. We thus denote $\mathsf{dVal}(v)$ this common value. We say that deterministic strategies $\sigma^\star$ of Min and $\tau^\star$ of Max are optimal (respectively, ε-optimal for a positive real number ε) if, for all vertices v: $\mathsf{dVal}^{\sigma^\star}(v) = \mathsf{dVal}(v)$ and $\mathsf{dVal}^{\tau^\star}(v) = \mathsf{dVal}(v)$ (respectively, $\mathsf{dVal}^{\sigma^\star}(v) \leqslant \mathsf{dVal}(v) + \varepsilon$ and $\mathsf{dVal}^{\tau^\star}(v) \geqslant \mathsf{dVal}(v) - \varepsilon$).

▶ **Example 1.** The deterministic value of the game on the left of Figure 1 is described in the introduction: $\mathsf{dVal}(v_{\mathsf{Min}}) = \mathsf{dVal}(v_{\mathsf{Max}}) = -10$. An optimal strategy for player Min consists in going to v_{Max} the first 10 times, and switching to the target vertex afterwards. An optimal strategy for player Max consists in directly going towards the target vertex.
If we remove the edge from v_{Max} to the target (of weight -10), we obtain another game in which $\mathsf{dVal}(v_{\mathsf{Min}}) = \mathsf{dVal}(v_{\mathsf{Max}}) = -\infty$ since Min can decide to turn as long as he wants in the negative cycle, before switching to the target. There is no optimal strategy for Min but a sequence of strategies guaranteeing a value as low as we want.

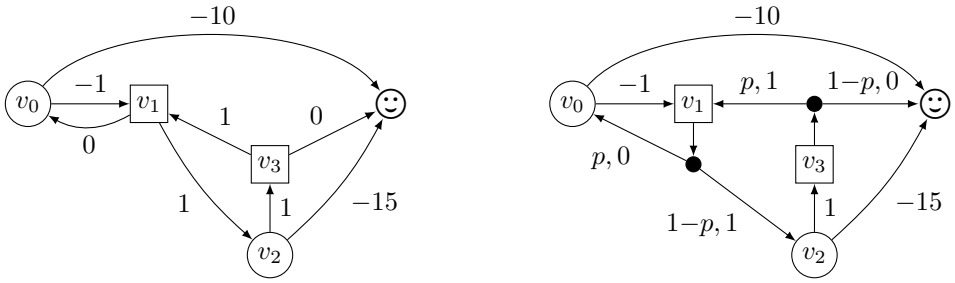


Figure 2 On the left, a more complex example of shortest-path game. On the right, the MDP associated with a randomised strategy of Min with a parametric probability $p \in (0, 1)$.

2.2 Memoryless strategies

Definitions above can be adapted for memoryless (randomised) strategies. In order to keep the explanations simple, we only define the upper value above, without relying on hypothetical determinacy results in this context. Once we fix a memoryless (randomised) strategy $\rho \in \mathsf{m}\Sigma_{\mathsf{Min}}$, we obtain a *Markov decision process* (MDP) where the other player must still choose how to react. An MDP is a tuple $\langle V, A, P \rangle$ where V is a set of vertices, A is a set of actions, and $P \colon V \times A \to \Delta(V)$ is a partial function mapping to some pair of vertices and actions a distribution of probabilities over the successor vertices. In our context, we let $\mathcal{G}^\rho$ be the MDP with the same set V of vertices as $\mathcal{G}$, actions $A = V \cup \{\bot\}$ being either successor vertices of the game or an additional action $\bot$ denoting the random choice of ρ, and a probability distribution P defined by:

- if $v \in V_{\mathsf{Max}}$, $P(v, v')$ is only defined if $(v, v') \in E$ in which case $P(v, v') = \mathsf{Dirac}_{v'}$, and $P(v, \bot)$ is also undefined;
- if $v \in V_{\mathsf{Min}}$, $P(v, \bot) = \rho(v)$, and $P(v, v')$ is undefined for all $v' \in V$.

In drawings of MDPs (and also of Markov chains, later), we show weights as trivially transferred from the game graph.

▶ **Example 2.** In Figure 1, a shortest-path game is presented on the left, with the MDP in the middle obtained by picking as a memoryless strategy for Min the one choosing to go to v_{Max} with probability $p \in (0, 1)$ and to the target vertex with probability $1 - p$. Another more complex example is given in Figure 2 where the memoryless strategy for Min consists, in vertex v_1, to choose successor v_0 with probability $p \in (0, 1)$ and successor v_2 with probability $1 - p$, and in vertex v_3, to choose successor v_1 with the same probability p and the target vertex with probability $1 - p$.

In such an MDP, when player Max has chosen her strategy, there will remain no "choices" to make, and we will thus end up in a *Markov chain*. A Markov chain (*MC*) is a tuple $\mathcal{M} = \langle V, P \rangle$ where V is a set of vertices, and $P \colon V \to \Delta(V)$ associates to each vertex a distribution of probabilities over the successor vertices. In our context, for all memoryless strategies $\chi \in \mathsf{m}\Sigma_{\mathsf{Max}}$, we let $\mathcal{G}^{\rho,\chi}$ the MC obtained from the MDP $\mathcal{G}^\rho$ by following strategy χ and action $\bot$. Formally, it consists of the same set V of vertices as $\mathcal{G}$, and mapping P associating to a vertex $v \in V_{\mathsf{Min}}$, $P(v) = \rho(v)$ and to a vertex $v \in V_{\mathsf{Max}}$, $P(v) = \chi(v)$.

▶ **Example 3.** On the right of Figure 1 is depicted the MC obtained when Max decides to go to v_{Min} with probability $q \in [0, 1]$ and to the target vertex with probability $1 - q$.

When starting in a given initial vertex v, we let $\mathbb{P}_v^{\rho,\chi}$ denote the induced probability measure over the sets of paths in the MC $\mathcal{G}^{\rho,\chi}$ (as before, $\mathcal{G}$ is made implicit in the notation). A *property* is any measurable subset of finite or infinite paths in the MC with respect to the

standard cylindrical sigma-algebra. For instance, we denote by $\mathbb{P}_v^{\rho,\chi}(\diamond T)$ the probability of the set of plays that reach the target set $T \subseteq V$ of vertices. Given a random variable X over the infinite paths in the MC, we let $\mathbb{E}_v^{\rho,\chi}(X)$ be the expectation of X with respect to the probability measure $\mathbb{P}_v^{\rho,\chi}$. Therefore, $\mathbb{E}_v^{\rho,\chi}(\mathbf{TP})$ is the expected weight of a path in the MC, weights being the ones taken from $\mathcal{G}$.

The objective of Max is to maximise the payoff in the MDP $\mathcal{G}^\rho$. We therefore define the value of strategy ρ of Min as the best case scenario for Max:

$$\mathsf{mVal}^\rho(v) = \sup_{\chi \in \mathsf{m}\Sigma_{\mathsf{Max}}} \mathbb{E}_v^{\rho,\chi}(\mathbf{TP})$$

By [1, Section 10.5.1], the value $\mathsf{mVal}^\rho(v)$ is finite if and only if $\mathbb{P}_v^{\rho,\chi}(\diamond T) = 1$ for all χ, i.e. if strategy ρ ensures the reachability of a target vertex with probability 1, no matter how the opponent plays. In this case, letting P be the probability mapping defining the MC $\mathcal{G}^{\rho,\chi}$, the vector $(\mathbb{E}_v^{\rho,\chi}(\mathbf{TP}))_{v\in V}$ is the only solution of the system of equations

$$\mathbb{E}_v^{\rho,\chi}(\mathbf{TP}) = \begin{cases} 0 & \text{if } v \in T \\ \sum_{v' \in E(v)} P(v,v') \times (\omega(v,v') + \mathbb{E}_{v'}^{\rho,\chi}(\mathbf{TP})) & \text{if } v \notin T \end{cases} \tag{1}$$

Since Min wants to minimise the shortest-path payoff, we finally define the memoryless upper value as

$$\overline{\mathsf{mVal}}(v) = \inf_{\rho \in \mathsf{m}\Sigma_{\mathsf{Min}}} \mathsf{mVal}^\rho(v)$$

Once again, we say that a memoryless strategy ρ is optimal (respectively, ε-optimal for a positive real number ε) if $\mathsf{mVal}^\rho(v) = \overline{\mathsf{mVal}}(v)$ (respectively, $\mathsf{mVal}^\rho(v) \leqslant \overline{\mathsf{mVal}}(v) + \varepsilon$). With respect to player Max, we only consider optimality and ε-optimality in the MDP $\mathcal{G}^\rho$.

▶ **Example 4.** For the game of Figure 1, we let σ and τ the memoryless strategies that result in the MC on the right. Letting $x = \mathbb{E}_{v_{\mathsf{Min}}}^{\rho,\chi}(\mathbf{TP})$ and $y = \mathbb{E}_{v_{\mathsf{Max}}}^{\rho,\chi}(\mathbf{TP})$, the system (1) rewrites as $x = (1-p) \times 0 + p \times y$ and $y = q \times (-1 + x) + (1-q) \times (-10)$. We thus have $x = p(9q-10)/(1-pq)$. Two cases happen, depending on the value of p: if $p < 9/10$, then Max maximises x by choosing $q = 1$, while she chooses $q = 0$ when $p \geqslant 9/10$. In all cases, player Max will therefore play deterministically: if $p < 9/10$, the expected payoff from v_{Min} will then be $\mathsf{mVal}^\rho(v_{\mathsf{Min}}) = -p/(1-p)$; if $p \geqslant 9/10$, it will be $\mathsf{mVal}^\rho(v_{\mathsf{Min}}) = -10p$. This value is always greater than the optimum -10 that Min were able to achieve with memory, since we must keep $1 - p > 0$ to ensure reaching the target with probability 1. We thus obtain $\overline{\mathsf{mVal}}(v_{\mathsf{Min}}) = \overline{\mathsf{mVal}}(v_{\mathsf{Max}}) = -10$ as before. There are no optimal strategies for Min, but an ε-optimal one consisting in choosing probability $p \geqslant 1 - \varepsilon/10$.

The fact that Max can play optimally with a deterministic strategy in the MDP $\mathcal{G}^\rho$ is not specific to this example. Indeed, in an MDP $\mathcal{G}^\rho$ such that $\mathbb{P}_v^{\rho,\chi}(\diamond T) = 1$ for all χ, Max cannot avoid reaching the target: she must then ensure the most expensive play possible. Considering the MDP $\tilde{\mathcal{G}}^\rho$ obtained by multiplying all the weights in the graph by -1, the objective of Max becomes a shortest-path objective. We can then deduce from [2] that she has an optimal deterministic memoryless strategy: the same applies in the original MDP $\mathcal{G}^\rho$.

▶ **Proposition 5.** *In the MDP $\mathcal{G}^\rho$ such that $\mathbb{P}_v^{\rho,\chi}(\diamond T) = 1$ for all χ,* Max *has an optimal deterministic memoryless strategy.*

2.3 Contribution

Our contribution consists in showing that optimal values are the same when restricting both players to memoryless or deterministic strategies:

▶ **Theorem 6.** *For all games $\mathcal{G}$ with a shortest-path objective, for all vertices v, we have* $\mathsf{dVal}(v) = \overline{\mathsf{mVal}}(v)$.

We show this theorem in the two next sections by a simulation of deterministic strategies with memoryless ones, and vice versa. We start here by ruling out the case of values $+\infty$. Indeed, $\mathsf{dVal}(v) = +\infty$ signifies that Min is not able to reach a target vertex from v with deterministic strategies. This also implies that Min has no memoryless randomised strategies to ensure reaching the target with probability 1, and thus $\overline{\mathsf{mVal}}(v) = +\infty$. Reciprocally, if $\overline{\mathsf{mVal}}(v) = +\infty$, then Min has no memoryless strategies to reach the target with probability 1 (since this is the only reason for having a value $+\infty$). Since reachability is a purely qualitative objective, and the game graph does not contain probabilities, Min cannot use memory in order to guarantee reaching the target: therefore, this also means that $\mathsf{dVal}(v) = +\infty$. In the end, we have shown that $\mathsf{dVal}(v) = +\infty$ if and only if $\overline{\mathsf{mVal}}(v) = +\infty$. We thus remove every such vertex from now on, which does not change the values of other vertices in the game.

▶ **Assumption.** From now on, all games $\mathcal{G}$ with a shortest-path objective are such that $\mathsf{dVal}(v)$ and $\overline{\mathsf{mVal}}(v)$ are different from $+\infty$, for all vertices v.

3 Simulating deterministic strategies with memoryless strategies

Towards proving Theorem 6, we show in this section that, for all shortest-path games $\mathcal{G} = \langle V, E, \omega, \mathbf{P}\rangle$ (where no values are $+\infty$) and vertices $v \in V$, $\overline{\mathsf{mVal}}(v) \leqslant \mathsf{dVal}(v)$. This is done by considering the *switching strategies* originated from [3], which are a particular kind of deterministic strategies: they are optimal from vertices of finite value, and they can get a value as low as wanted from vertices of value $-\infty$. A switching strategy $\sigma = \langle \sigma_1, \sigma_2, \alpha\rangle$ is described by two deterministic memoryless strategies σ_1 and σ_2, as well as a switching parameter α. The strategy σ consists in playing along σ_1, until eventually switching to σ_2 when the length of the current finite play is greater than α. Strategy σ_2 is thus any *attractor strategy* ensuring that plays reach the target set of vertices: it can be computed via a classical attractor computation. Strategy σ_1 is chosen so that every cyclic finite play $v_0v_1\dots v_kv_0$ conforming to σ_1 has a negative total weight: this is called an *NC-strategy* (for *negative-cycle-strategy*) in [3]. The *fake-value* of σ_1 from a vertex v_0 is defined by $\mathsf{fake}^{\sigma_1}(v_0) = \sup\{\mathbf{TP}(v_0v_1\cdots v_k) \mid v_k \in T, v_0v_1\cdots v_k \text{ conforming to } \sigma_1\}$, letting $\sup\emptyset = -\infty$: it consists of only considering plays conforming σ_1 that reach the target. Strategy σ_1 is said to be *fake-optimal* if $\mathsf{fake}^{\sigma_1}(v) \leqslant \mathsf{dVal}(v)$ for all vertices v: in this case, if a play from v conforms to σ_1 (or σ before the switch happens) and reaches the target set of vertices, it has a weight at most $\mathsf{dVal}(v)$.

▶ **Proposition 7** ([3]). *There exists a fake-optimal NC-strategy σ_1. Moreover, for all such fake-optimal NC-strategies σ_1, for all attractor strategies σ_2, and for all $n \in \mathbb{N}$, the switching parameter $\alpha = (2W(|V|-1)+n)|V|+1$ defines a switching strategy $\sigma = \langle\sigma_1, \sigma_2, \alpha\rangle$ with a value $\mathsf{dVal}^\sigma(v) \leqslant \max(-n, \mathsf{dVal}(v))$, from all initial vertices $v \in V$.*

In particular, if $\mathsf{dVal}(v)$ is finite, for n large enough, the switching strategy is optimal. If $\mathsf{dVal}(v) = -\infty$ however, the sequence $(\sigma^n)_{n\in\mathbb{N}}$ of strategies, each with a different parameter n, has a value that tends to $-\infty$.

▶ **Example 8.** For all $n \in \mathbb{N}$, let $\sigma = (\sigma_1, \sigma_2, \alpha)$ the switching strategy described above. In Figure 1, we have $\sigma_1(v_{\mathsf{Min}}) = v_{\mathsf{Max}}$, $\sigma_2(v_{\mathsf{Min}}) = \☺$ and $\alpha = 3(40+n)+1$. In Figure 2, σ_1 chooses v_0 from v_1 and v_1 from v_3, σ_2 chooses v_2 from v_1 and ☺ from v_3 and $\alpha = 5(60+n)+1$, for all $n \in \mathbb{N}$.

Definition of a memoryless (randomised) strategy. Let $n \in \mathbb{N}$, we consider the switching strategy $\sigma = \langle \sigma_1, \sigma_2, \alpha \rangle$ described before, of value $\mathsf{dVal}^\sigma(v) \leqslant \max(-n, \mathsf{dVal}(v))$, and simulate it with a memoryless (randomised) strategy for Min, denoted ρ_p, with a parametrised probability $p \in (0,1)$. This new strategy is a probabilistic superposition of the two memoryless deterministic strategies σ_1 and σ_2.

Formally, we define ρ_p on each strongly connected components (SCC) of the graph according to the presence of a negative cycle. In an SCC that does not contain negative cycles, for each vertex $v \in V_{\mathsf{Min}}$ of the SCC, we let $\rho_p(v) = \mathsf{Dirac}_{\sigma_1(v)}$: player Min chooses to play the first strategy σ_1 of the switching strategy, thus looking for a negative cycle in the next SCCs (in topological order) if any. In an SCC that contains a negative cycle, for each vertex $v \in V_{\mathsf{Min}}$ of the SCC, we let $\rho_p(v)$ be the distribution of support $\{\sigma_1(v), \sigma_2(v)\}$ that chooses $\sigma_1(v)$ with probability p and $\sigma_2(v)$ with probability $1-p$, except if $\sigma_1(v) = \sigma_2(v)$ in which case we choose it with probability 1. Note that MDPs in Figures 1 and 2 are obtained by applying this strategy ρ_p.

We fix some vertex $v_0 \in V$. In the rest of this section, we prove the following result:

▶ **Proposition 9.** *For ε small enough and p close enough to* 1, $\mathsf{mVal}^{\rho_p,\tau}(v_0) \leqslant \mathsf{dVal}^\sigma(v_0) + \varepsilon$.

This entails the expected result. Indeed, if $\mathsf{dVal}(v_0) \in \mathbb{Z}$, we get (with $n = |\mathsf{dVal}(v_0)|$) that $\mathsf{mVal}^{\rho_p}(v_0) \leqslant \mathsf{dVal}(v_0) + \varepsilon$, and thus $\overline{\mathsf{mVal}}(v_0) \leqslant \mathsf{dVal}(v_0)$ since this holds for all $\varepsilon > 0$. Otherwise, $\mathsf{dVal}(v_0) = -\infty$, and letting n tend towards $+\infty$, we also get $\overline{\mathsf{mVal}}(v_0) = -\infty$.

We first prove that ρ_p is one of the strategies of Min that guarantee to reach the target with probability 1 in the MDP $\mathcal{G}^{\rho_p}$ no matter how Max reacts.

▶ **Proposition 10.** *For all strategies $\chi \in \mathsf{m}\Sigma_{\mathsf{Max}}$, $\mathbb{P}_{v_0}^{\rho_p,\chi}(\Diamond\, T) = 1$.*

Proof. Recall that we designed our graph games so that target vertices are the only deadlocks. Thus, by using the characterisation of [1, Lemma 10.111], $\min_{\chi \in \mathsf{m}\Sigma_{\mathsf{Max}}} \mathbb{P}_{v_0}^{\rho_p,\chi}(\Diamond\, T) = 1$ if and only if for all $\chi \in \mathsf{m}\Sigma_{\mathsf{Max}}$, all bottom SCCs of the MC $\mathcal{G}^{\rho_p,\chi}$ (the ones from which we cannot exit) consist in a unique target vertex. Suppose in the contrary that Max has a memoryless strategy χ such that the MC $\mathcal{G}^{\rho_p,\chi}$ contains a bottom SCC $\mathcal{C}$ with no target vertices.

If all vertices of $\mathcal{C}$ belong to Max, then they all have a successor in $\mathcal{C}$ and therefore there also exists a deterministic memoryless strategy τ' for which all vertices $v \in \mathcal{C}$ are such that $\mathsf{dVal}^{\tau'}(v) = +\infty$, and thus $\mathsf{dVal}(v) = +\infty$: this contradicts our hypothesis that all vertices have a deterministic value different from $+\infty$.

Otherwise, for all vertices $v \in V_{\mathsf{Min}} \cap \mathcal{C}$, since $\mathcal{C}$ is a bottom SCC of $\mathcal{G}^{\rho_p,\chi}$, the distribution $\rho_p(v)$ has its support included in $\mathcal{C}$. If $\mathcal{C}$ is included in a SCC of $\mathcal{G}$ with no negative cycles, $\mathrm{supp}(\rho_p(v)) = \{\sigma_1(v)\}$: playing $\sigma_1(v)$ in $\mathcal{C}$ will end up in a cycle (since there are no deadlocks) that must be negative, by the hypothesis on σ_1, which is impossible. Thus, $\mathcal{C}$ must be included in an SCC of $\mathcal{G}$ with a negative cycle. Then, $\mathrm{supp}(\rho_p(v)) = \{\sigma_1(v), \sigma_2(v)\} \subseteq \mathcal{C}$, and in particular the attractor strategy is not able to reach a target vertex: playing the deterministic switching strategy σ will result in not reaching a target vertex either, so that $\mathsf{dVal}(v) = +\infty$ for $v \in V_{\mathsf{Min}} \cap \mathcal{C}$, which also contradicts our hypothesis. ◀

We can therefore apply Proposition 5. This result is very helpful since it allows us to only consider deterministic memoryless strategies τ to compute $\mathsf{mVal}^{\rho_p}(v_0) = \sup_\tau \mathsf{mVal}^{\rho_p,\tau}(v_0)$, for all initial vertices v_0. We thus consider such a strategy τ and we now show that

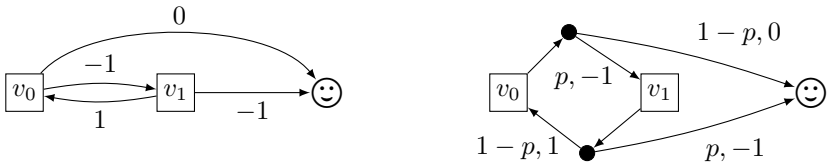


Figure 3 On the left, a game graph with no negative cycles where ρ_p is optimal. The MC obtained when playing a different randomised memoryless strategy.

$\mathsf{mVal}^{\rho_p,\tau}(v_0) \leqslant \mathsf{dVal}^{\sigma}(v) + \varepsilon$ whenever $p < 1$ is close enough to 1 (in function of $\varepsilon > 0$). By gathering the finite number of lower bounds about p, for all deterministic memoryless strategies of Max (there are a finite number of such), we obtain a lower bound for p such that $\mathsf{mVal}^{\rho_p}(v_0) \leqslant \mathsf{dVal}^{\sigma}(v_0) + \varepsilon$, as expected to prove Proposition 9.

The case where the whole game graph does not contain any negative cycles is easy. In this case, ρ_p chooses the strategy σ_1 with probability 1, by definition since no SCC contain a negative cycle (this is the only reason why we defined ρ_p as it is, for such SCCs): a play from initial vertex v_0 conforming to ρ_p is thus conforming to σ_1. Since the graph contains no negative cycles and all cycles conforming to σ_1 must be negative, all plays from v_0 conforming to σ_1 reach the target set of vertices, with a total payoff at most $\mathsf{dVal}^{\sigma}(v_0)$. This single play has probability 1 in the MC $\mathcal{G}^{\rho_p,\tau}$, thus $\mathbb{E}_{v_0}^{\rho_p,\tau}(\mathbf{TP}) \leqslant \mathsf{dVal}^{\sigma}(v_0)$, which proves that $\mathsf{mVal}^{\rho_p}(v) \leqslant \mathsf{dVal}^{\sigma}(v_0)$ as expected.

▶ **Example 11.** If the definition of ρ_p would not distinguish the SCCs with no negative cycles from the other SCCs, we would not have the optimality of ρ_p as shown before. Indeed, consider the game graph on the left of Figure 3, which has no negative cycles. We have $\mathsf{dVal}(v_0) = -2$ and $\mathsf{dVal}(v_1) = -1$. As a switching strategy, we can choose $\sigma_1(v_0) = v_1$, $\sigma_1(v_1) = ☺$, $\sigma_2(v_0) = ☺$ and $\sigma_2(v_1) = v_0$. Then, ρ_p is equal to σ_1 (and thus independent of p), and $\mathsf{mVal}^{\rho_p}(v_0) = -2$ and $\mathsf{mVal}^{\rho_p}(v_1) = -1$. However, if we would have chosen to still mix σ_1 and σ_2, we would obtain a strategy ρ'_p, and the MC on the right of Figure 3. Then, we get $\mathsf{mVal}^{\rho'_p}(v_0) = -2p^2/(1-p(1-p))$ and $\mathsf{mVal}^{\rho'_p}(v_1) = (p^2-3p+1)/(1-p(1-p))$ whose limits are -2 and -1 respectively, when p tends to 1. This strategy ρ'_p would then still be ε-optimal for p close enough to 1.

Now, suppose that the graph game contains negative cycles. We let $c > 0$ be the maximal size of an elementary cycle (that visits a vertex at most once) in $\mathcal{G}$, $w^- > 0$ be the opposite of the maximal weight of an elementary negative cycle in $\mathcal{G}$, and $w^+ \geqslant 0$ be the maximal weight of an elementary non-negative cycle in $\mathcal{G}$ (or 0 if such cycle does not exist).

▶ **Example 12.** In the graph of Figure 1, we have $c = 2$, $w^- = 1$, and $w^+ = 0$ (since there is no non-negative cycles). In the game graph of Figure 2, we have $c = 3$, $w^- = 1$, and $w^+ = 3$.

The difficulty initiates from the possible presence of non-negative cycles too. Indeed, when applying the switching strategy σ, all cycles conforming to σ_1 have a negative weight. This is no longer true with the probabilistic superposition ρ_p, as can be seen in the example of Figure 2. Finding an adequate lower-bound for p requires to estimate $\mathbb{E}_{v_0}^{\rho_p,\tau}(\mathbf{TP})$, by controlling the weight and probability of non-negative cycles, balancing them with the ones of negative cycles. The crucial argument comes from the definition of the superposition ρ_p:

▶ **Lemma 13.** *All cycles in $\mathcal{G}^{\rho_p,\tau}$ of non-negative total weight contain at least one edge of probability $1-p$.*

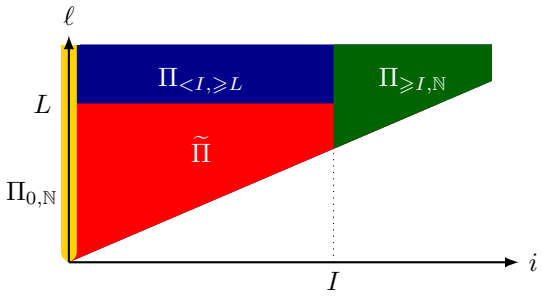


Figure 4 Partition of plays Π.

Proof. Suppose on the contrary that all edges have probability p or 1, then the cycle is conforming to strategy σ_1, and has therefore a negative weight. ◀

Proof of Proposition 9. The proof that $\mathsf{mVal}^{\rho_p,\tau}(v_0) \leqslant \mathsf{dVal}^{\sigma}(v_0)+\varepsilon$ is done by partitioning the set Π of plays starting in v_0, conforming to ρ_p and τ, and reaching the target set of vertices, into subsets $\Pi_{i,\ell}$ according to the number i of edges of probability $1-p$ they go through, and their length ℓ (we always have $i \leqslant \ell$). The partition is depicted in Figure 4:

- $\Pi_{0,\mathbb{N}}$, depicted in yellow, contains all plays with no edges of probability $1-p$;
- $\Pi_{\geqslant I,\mathbb{N}}$, depicted in green, contains all plays having at least

$$I = \left\lceil \frac{2w^+}{\gamma W} + \frac{8(w^+ + |V|W)}{\varepsilon} \right\rceil$$

  edges[1] of probability $1-p$ where $\gamma = c\left(1+\frac{w^+}{w^-}\right) \geqslant 1$;
- $\Pi_{<I,\geqslant L}$, depicted in blue, contains all plays with at most I edges of probability $1-p$, and of length at least $L = I\gamma + \frac{2|\mathsf{dVal}^{\sigma}(v_0)|+|V|W}{w^-}c + |V|$;
- $\widetilde{\Pi}$, depicted in red, is the rest of the plays.

We let $\gamma_{0,\mathbb{N}}$ (respectively, $\gamma_{<I,\geqslant L}$, $\gamma_{\geqslant I,\mathbb{N}}$, and $\widetilde{\gamma}$) be the expectation $\mathbb{E}_{v_0}^{\rho_p,\tau}(\mathbf{TP})$ restricted to plays in $\Pi_{0,\mathbb{N}}$ (respectively, $\Pi_{<I,\geqslant L}$, $\Pi_{\geqslant I,\mathbb{N}}$, $\widetilde{\Pi}$). By linearity of expectation,

$$\mathsf{mVal}^{\rho_p,\tau}(v_0) = \mathbb{E}_{v_0}^{\rho_p,\tau}(\mathbf{TP}) = \gamma_{0,\mathbb{N}} + \gamma_{<I,\geqslant L} + \gamma_{\geqslant I,\mathbb{N}} + \widetilde{\gamma} \tag{2}$$

Partitioning the plays allows us to carefully control non-negative cycles: plays with a large number of non-negative cycles contain a large number of edges of probability $1-p$, by Lemma 13; thus if p is made close enough to 1, the probability of this set of plays will be small enough. We thus control separately the four terms of (2) to obtain $\mathsf{mVal}^{\rho_p,\tau}(v_0) \leqslant \mathsf{dVal}^{\sigma}(v_0) + \varepsilon$.

Yellow and blue zones are such that $\gamma_{0,\mathbb{N}} + \gamma_{<I,\geqslant L} \leqslant \mathsf{dVal}^{\sigma}(v_0) + \varepsilon/2$

All plays of $\Pi_{0,\mathbb{N}}$ reach the target without edges of probability $1-p$, i.e. by conforming to σ_1. By fake-optimality of σ_1, their total payoff is upper-bounded by $\mathsf{dVal}^{\sigma}(v_0)$. Notice that, in case $\mathsf{dVal}(v_0) = -\infty$, no plays conforming to σ_1 starting in v_0 reach the target, since Min has the opportunity to stay as long as he wants in negative cycles: thus $\Pi_{0,\mathbb{N}} = \emptyset$ in this case, and $\gamma_{0,\mathbb{N}} = 0$.

[1] This intricate definition of I, as well as L in the next item, is justified by the computations that will follow in the proof.

All plays of $\Pi_{i,\ell}$, with $1 \leqslant i < I$ and $\ell \geqslant L$, go through i edges of probability $1-p$. By Lemma 13, they contain at most i elementary cycles of non-negative total weight (each of weight at most w^+). The total length of these cycles is at most ic. Once we have removed these cycles from the play, it remains a play of length at least $\ell - ic$. By a repeated pumping argument, it still contains at least $\left\lfloor \frac{\ell - ic - |V|}{c} \right\rfloor$ elementary cycles, that have all a negative total weight (each has a weight at most $-w^-$). The remaining part, once removed the last negative cycles it contains, has length at most $|V|$, and thus a total payoff at most $|V|W$. In summary, the total payoff of a play in $\Pi_{i,\ell}$ is at most

$$\begin{aligned} iw^+ + \left\lfloor \frac{\ell - ic - |V|}{c} \right\rfloor (-w^-) + |V|W &\leqslant Iw^+ + \frac{L - Ic - |V|}{c}(-w^-) + |V|W \\ &= -2|\mathsf{dVal}^\sigma(v_0)| \leqslant 0 \end{aligned} \tag{3}$$

Let us then consider three cases.

- If $\mathsf{dVal}^\sigma(v_0) \geqslant 0$, we note that all plays in $\Pi_{<I,\geqslant L}$ have a non-positive total payoff, therefore at most $\mathsf{dVal}^\sigma(v_0)$. Thus,

$$\begin{aligned} \gamma_{0,\mathbb{N}} + \gamma_{<I,\geqslant L} &\leqslant \mathsf{dVal}^\sigma(v_0)\mathbb{P}(\Pi_{0,\mathbb{N}}) + \mathsf{dVal}^\sigma(v_0)\mathbb{P}(\Pi_{<I,\geqslant L}) \\ &= \mathsf{dVal}^\sigma(v_0)\big(\mathbb{P}(\Pi_{0,\mathbb{N}}) + \mathbb{P}(\Pi_{<I,\geqslant L})\big) \leqslant \mathsf{dVal}^\sigma(v_0) \end{aligned}$$

- If $\mathsf{dVal}^\sigma(v_0) < 0$ and $\Pi_{<I,\geqslant L} \neq \emptyset$, we have $\gamma_{0,\mathbb{N}} \leqslant 0$ (whatever $\mathsf{dVal}(v_0) = -\infty$ or not). Moreover, a play in $\Pi_{i,\ell}$ goes through i edges of probability $1-p$ and at most ℓ edges of probability p, other edges having probability 1. So, it has probability at least $(1-p)^i p^\ell$. We can deduce that

$$\gamma_{<I,\geqslant L} \leqslant \sum_{i=1}^{I-1}\sum_{\ell=L}^{\infty}(1-p)^i p^\ell \underbrace{\left(iw^+ + \left\lfloor \frac{\ell - ic - |V|}{c} \right\rfloor (-w^-) + |V|W\right)}_{\leqslant 0 \text{ by } (3)} \leqslant \mathsf{dVal}^\sigma(v_0)$$

  the last inequality being true when p is close enough to 1, as shown in Appendix A.
- If $\mathsf{dVal}^\sigma(v_0) < 0$ and $\Pi_{<I,\geqslant L} = \emptyset$, then $\mathsf{dVal}(v_0) \neq -\infty$, since otherwise a play conforming to strategy σ_1 for L rounds, and then switching to σ_2 for at most $|V| \leqslant I$ rounds, would be in $\Pi_{<I,\geqslant L}$. Thus, $\gamma_{0,\mathbb{N}} + \gamma_{<I,\geqslant L} = \gamma_{0,\mathbb{N}} \leqslant \mathsf{dVal}^\sigma(v_0)\mathbb{P}(\Pi_{0,\mathbb{N}})$. Moreover, by the same argument, all plays in $\Pi_{0,\mathbb{N}}$ are acyclic and their length is at most $|V|$: they go through no edges of probability $1-p$, and thus at most $|V|$ edges of probability p. Therefore, $\mathbb{P}(\Pi_{0,\mathbb{N}}) \geqslant p^{|V|}$, and thus, once again because $\mathsf{dVal}^\sigma(v_0) < 0$, when $p \geqslant (1 - \varepsilon/2|\mathsf{dVal}^\sigma(v_0)|)^{1/|V|}$ which is less than 1 for ε small enough,

$$\gamma_{0,\mathbb{N}} + \gamma_{<I,\geqslant L} \leqslant \mathsf{dVal}^\sigma(v_0)p^{|V|} \leqslant \mathsf{dVal}^\sigma(v_0) + \varepsilon/2$$

In all cases, we have $\gamma_{0,\mathbb{N}} + \gamma_{<I,\geqslant L} \leqslant \mathsf{dVal}^\sigma(v_0) + \varepsilon/2$.

Red and green zones are such that $\gamma_{\geqslant I,\mathbb{N}} + \widetilde{\gamma} \leqslant \varepsilon/2$

First, a play of $\Pi_{\geqslant I,\mathbb{N}}$ has a large total payoff, but a low probability to happen, which enables us to control its expected payoff. Indeed, consider a play of $\Pi_{i,\mathbb{N}}$, with $i \geqslant I$. By Lemma 13, it contains at most i elementary cycles of non-negative total weight. The remaining of the play may contain negative cycles, as well as an acyclic part reaching the target in at most $|V|$ steps. The total payoff of the whole play is thus at most $iw^+ + |V|W$. Moreover, $\mathbb{P}(\Pi_{i,\mathbb{N}}) \leqslant (1-p)^i$ since all the plays contain i edges of probability $1-p$. Overall,

$$\gamma_{\geqslant I,\mathbb{N}} \leqslant \sum_{i=I}^{\infty}(iw^+ + |V|W)(1-p)^i = (1-p)^I\left(\frac{w^+}{p}I + \frac{w^+(1-p)}{p^2} + \frac{|V|W}{p}\right) \leqslant \frac{\varepsilon}{4}$$

where the last inequality holds for p close enough to 1, as shown in Appendix A.

Finally, all plays of $\widetilde{\Pi}$ have a length less than L (and thus a total payoff at most LW) and a number i of edges of probability $1-p$ such that $0 < i < I$. By a similar argument as before, if $p \geqslant LW/(LW+\varepsilon/4)$, we have

$$\widetilde{\gamma} \leqslant \sum_{i=1}^{I} LW(1-p)^i = LW\frac{(1-p)(1-(1-p)^I)}{p} \leqslant LW\frac{1-p}{p} \leqslant \frac{\varepsilon}{4}$$

since $p \mapsto (1-p)/p$ is decreasing on $(0,1)$. ◀

This ends the proof that for all vertices v, $\overline{\mathsf{mVal}}(v) \leqslant \mathsf{dVal}(v)$. Let us illustrate the computation of the lower-bound on probability p of the memoryless strategy ρ_p in the previously studied examples.

▶ **Example 14.** For the game in Figure 1, with initial vertex v_{Min}, we have $\gamma = 2$. For $\varepsilon = 0.1$, we then have $I = 2400$, and $L = 4903$. The lower-bound on p is then $q = 0.9999995$, which gives a value $\mathsf{mVal}^{\rho_p}(v_{\mathsf{Min}}) = -10p = -9.999995$. For the game in Figure 2, with initial vertex v_2, we have $\gamma = 12$. For $\varepsilon = 0.1$, we then have $I = 3121$, and $L = 37730$. The lower-bound on p is then $q = 0.99999998$, which gives a value $\mathsf{mVal}^{\rho_p}(v_2) \approx -7.9999996$. We see that the lower-bound are correct, even if they could certainly be made coarser.

4 Simulating memoryless strategies with deterministic strategies

To finish the proof of Theorem 6, we will show that $\mathsf{dVal}(v) \leqslant \overline{\mathsf{mVal}}(v)$, for all vertices v. For a given memoryless strategy ρ ensuring that Min reaches the target set T with probability 1, we build a deterministic strategy σ which guarantees a value $\mathsf{dVal}^\sigma(v) \leqslant \mathsf{mVal}^\rho(v)$ from vertex v. Then, as in the previous section, if $\overline{\mathsf{mVal}}(v)$ is finite, for an ε-optimal memoryless strategy ρ, we get a deterministic strategy such that $\mathsf{dVal}^\sigma(v) \leqslant \overline{\mathsf{mVal}}(v) + \varepsilon$, and thus $\mathsf{dVal}(v) \leqslant \overline{\mathsf{mVal}}(v) + \varepsilon$. We can conclude since this holds for all $\varepsilon > 0$. In case $\overline{\mathsf{mVal}}(v) = -\infty$, if ρ guarantees a value at most $-n$ with $n \in \mathbb{N}$, then so does the deterministic strategy σ, which also ensures that $\mathsf{dVal}(v) = -\infty$.

We fix a memoryless strategy ρ, and an initial vertex v_0. The first attempt to build a deterministic strategy σ such that $\mathsf{dVal}^\sigma(v) \leqslant \overline{\mathsf{mVal}}(v) + \varepsilon$ would be to use classical techniques of finite-memory strategies, for instance in Street or Müller games: for instance, to ensure the visit of two vertices v_1 and v_2 infinitely often during an infinite play (to win a Müller game with winning objective $\{v_1, v_2\}$), we would try to reach v_1 with a first memoryless strategy, and then reach v_2 with another memoryless strategy, before switching again to reach v_1 again, etc.

▶ **Example 15.** Let us try this technique on the shortest-path game of Figure 1. We consider as a starting point the memoryless strategy ρ such that $\rho(v_{\mathsf{Min}}) = \delta$ with $\delta(☺) = 2/3$ and $\delta(v_{\mathsf{Max}}) = 1/3$ (this is the case $p = 1/3$ in the MDP on the middle of Figure 1). As seen in Example 4, this strategy has value $\mathsf{mVal}^\rho(v_{\mathsf{Min}}) = -1/2$ et $\mathsf{mVal}^\rho(v_{\mathsf{Max}}) = -3/2$. Naively, we could try to mimic the distribution δ by using memory as follows: when in v_{Min}, go to ☺ two thirds of the time and to v_{Min} one third of the time. Moreover, we would naively try to follow first the choice with greatest probability. In this case, the strategy σ would first choose to go to ☹, thus stopping immediately the play. We thus get $\mathsf{dVal}^\sigma(v_{\mathsf{Min}}) = 0 > -1/2 + \varepsilon$ as soon as $\varepsilon < 1/2$.

The main reason why this naive approach fails is that the plays are essentially finite in shortest-path games. We thus cannot delay the choices and must carefully play as soon as the play starts. Instead, our solution is to define a switching strategy $\sigma = \langle \sigma_1, \sigma_2, \alpha \rangle$, with σ_2 any attractor strategy, and $\alpha = \max(0, |V|W - \mathsf{mVal}^\rho(v_0)) \times |V| + 1$.

▶ **Example 16** (Example 15 continued). In the game of Figure 1, the attractor strategy is $\sigma_2(v_{\mathsf{Min}}) = ☺$. We then choose $\sigma_1(v_{\mathsf{Min}})$ so as to minimise the immediate reward obtained by playing one turn and then getting the value ensured by ρ:

$$\sigma_1(v_{\mathsf{Min}}) = \operatorname{argmin}_{v' \in \{v_{\mathsf{Max}}, ☺\}} [w(v, v') + \mathsf{mVal}^\rho(v')] = v_{\mathsf{Max}}$$

For an appropriate choice of α, we thus recover the optimal switching strategy for this game.

In the rest of this section, we will detail how to define strategy σ_1 in general so as to obtain the following property:

▶ **Proposition 17.** *The switching strategy $\sigma = \langle \sigma_1, \sigma_2, \alpha \rangle$ built from the memoryless (randomised) strategy ρ satisfies* $\mathsf{dVal}^\sigma(v_0) \leqslant \mathsf{mVal}^\rho(v_0)$.

The construction of σ_1 is split in two parts. First, we restrict the possibilities for $\sigma_1(v)$ to a subset $\widetilde{E}(v)$ of $\mathrm{supp}(\rho(v))$ in (4): with respect to Example 15, this will forbid the use of edge $(v_{\mathsf{Min}}, ☺)$ in particular. The definition of $\sigma_1(v)$ is then given later in (7).

We restrict our attention to edges present in the MDP $\mathcal{G}^\rho$, and for each vertex $v \in V_{\mathsf{Min}}$, we let

$$\widetilde{E}(v) = \operatorname*{argmin}_{v' \in \mathrm{supp}(\rho(v))} [w(v, v') + \mathsf{mVal}^\rho(v')] \tag{4}$$

be the successors of v that minimise the expected value at horizon 1. We let $\widetilde{\mathcal{G}}$ be the game obtained from $\mathcal{G}$ by removing all edges (v, v') from a vertex $v \in V_{\mathsf{Min}}$ such that $v' \notin \widetilde{E}(v)$.

▶ **Lemma 18.** *(i) Each finite play of $\widetilde{\mathcal{G}}$ from a vertex v has a total payoff at most* $\mathsf{mVal}^\rho(v)$. *(ii) Each cycle in the game $\widetilde{\mathcal{G}}$ has a non-positive total weight.*

Proof. We prove the property (i) on finite plays π of $\widetilde{\mathcal{G}}$ by induction on the length of π, for all initial vertices v. If π has length 0, this means that $v \in T$, in which case $\mathbf{TP}(\pi) = 0 = \mathsf{mVal}^\rho(v)$. Consider then a play $\pi = v\pi'$ of length at least 1, with π' starting from v', so that $\mathbf{TP}(\pi) = \omega(v, v') + \mathbf{TP}(\pi')$. By induction hypothesis, $\mathbf{TP}(\pi') \leqslant \mathsf{mVal}^\rho(v')$, so that $\mathbf{TP}(\pi) \leqslant \omega(v, v') + \mathsf{mVal}^\rho(v')$.

Suppose first that $v \in V_{\mathsf{Max}}$. By Proposition 5, we know that Max can play optimally in the MDP $\mathcal{G}^\rho$ with a deterministic and memoryless strategy. For each possible deterministic and memoryless strategy τ of Max, we have $\mathsf{mVal}^\rho(u) \geqslant \mathbb{E}_u^{\rho,\tau}(\mathbf{TP})$ for all $u \in V_{\mathsf{Max}}$, and by the system (1) of equations, letting $u' = \tau(u)$, $\mathbb{E}_u^{\rho,\tau}(\mathbf{TP}) = \omega(u, u') + \mathbb{E}_{u'}^{\rho,\tau}(\mathbf{TP})$. We thus know that $\mathsf{mVal}^\rho(u) \geqslant \omega(u, u') + \mathbb{E}_{u'}^{\rho,\tau}(\mathbf{TP})$. By taking a maximum over all deterministic and memoryless strategies τ of Max, Proposition 5 ensures that

$$\forall u \in V_{\mathsf{Max}} \quad \forall u' \in E(u) \qquad \mathsf{mVal}^\rho(u) \geqslant \omega(u, u') + \mathsf{mVal}^\rho(u') \tag{5}$$

In particular, $\mathsf{mVal}^\rho(v) \geqslant \omega(v, v') + \mathsf{mVal}^\rho(v') \geqslant \mathbf{TP}(\pi)$.

If $v \in V_{\mathsf{Min}}$, then $v' \in \widetilde{E}(v)$ so that $\omega(v, v') + \mathsf{mVal}^\rho(v')$ is minimum over all possible successors $v' \in \mathrm{supp}(\rho(v))$. The system (1) of equations implies that, for an optimal strategy χ of Max,

$$\begin{aligned} \mathsf{mVal}^\rho(v) = \mathbb{E}_v^{\rho,\chi}(\mathbf{TP}) &= \sum_{v'' \in E(v)} P(v, v'') \times (\omega(v, v'') + \mathbb{E}_{v''}^{\rho,\chi}(\mathbf{TP})) \\ &= \sum_{v'' \in \mathrm{supp}(\rho(v))} P(v, v'') \times (\omega(v, v'') + \mathsf{mVal}^\rho(v'')) \geqslant \omega(v, v') + \mathsf{mVal}^\rho(v') \end{aligned} \tag{6}$$

so that we also get $\mathsf{mVal}^\rho(v) \geqslant \mathbf{TP}(\pi)$.

We then prove the property (ii) on cycles. Consider thus a cycle $v_1v_2\cdots v_kv_1$ of $\widetilde{\mathcal{G}}$, and let $\omega_1 = \omega(v_1,v_2), \omega_2 = \omega(v_2,v_3), \dots, \omega_k = \omega(v_k,v_1)$ be the sequence of weights of edges. We also let $v_{k+1} = v_1$. We show that $\omega_1+\omega_2+\cdots+\omega_k \leqslant 0$. Let $i \in \{1,2,\dots,k\}$. If $v_i \in V_{\mathsf{Max}}$, by (5), $\mathsf{mVal}^\rho(v_i) \geqslant \omega_i + \mathsf{mVal}^\rho(v_{i+1})$. If $v_i \in V_{\mathsf{Min}}$, by the reasoning applied above in (6), we also know that $\mathsf{mVal}^\rho(v_i) \geqslant \omega_i + \mathsf{mVal}^\rho(v_{i+1})$. By summing all the inequalities above, we get

$$\sum_{i=1}^{k} \mathsf{mVal}^\rho(v_i) \geqslant \sum_{i=1}^{k} \omega_i + \sum_{i=1}^{k} \mathsf{mVal}^\rho(v_i) \qquad \text{i.e.} \quad \omega_1+\omega_2+\cdots+\omega_k \leqslant 0$$ ◀

▶ **Example 19.** Consider again the game graph on the left of Figure 3, and the memoryless strategy ρ'_p giving rise to the MDP/MC on the right of Figure 3. Recall that $\mathsf{mVal}^{\rho'_p}(v_0) = -2p^2/(1-p(1-p))$ and $\mathsf{mVal}^{\rho'_p}(v_1) = (p^2-3p+1)/(1-p(1-p))$. Consider p close enough to 1 so that $\mathsf{mVal}^{\rho'_p}(v_0) \leqslant -3/2$ and $\mathsf{mVal}^{\rho'_p}(v_1) \leqslant -1/2$. Then, we have $\widetilde{E}(v_0) = \{v_1\}$ and $\widetilde{E}(v_1) = \{\text{☺}\}$. The corresponding game graph $\widetilde{\mathcal{G}}$ contains only edges (v_0,v_1) and $(v_1,\text{☺})$, and thus no cycles. The unique finite play from vertex v_0 has total-payoff $-2 \leqslant \mathsf{mVal}^{\rho'_p}(v_0)$. In particular, the only possible memoryless deterministic strategy σ_1 in $\widetilde{\mathcal{G}}$ is optimal in $\mathcal{G}$.

For each vertex v in the game, we let $d(v)$ be the distance (number of steps) of v to the target given by an attractor computation to the target in $\mathcal{G}^\rho$ (notice that this may be different from the distance given in the whole game graph, since some edges are taken with probability 0 in ρ, but still $d(v) < +\infty$ since ρ ensures to reach T with probability 1). We then let, for all vertices $v \in V_{\mathsf{Min}}$,

$$\sigma_1(v) = \operatorname*{argmin}_{v' \in \widetilde{E}(v)} d(v') \tag{7}$$

▶ **Example 20.** Consider once again the game graph of Figure 3, but with a new memoryless strategy ρ''_p defined by $\rho''_p(v_0) = \mathsf{Dirac}_{v_1}$ and $\rho''_p(v_1) = \delta$ such that $\delta(v_0) = 1-p$ and $\delta(\text{☺}) = p$, where $p \in (0,1)$. Then, we can check that $\mathsf{mVal}^{\rho''_p}(v_0) = -2$ and $\mathsf{mVal}^{\rho''_p}(v_1) = -1$. Thus, $\widetilde{E}(v_0) = \{v_1\}$ and $\widetilde{E}(v_1) = \{v_0, \text{☺}\}$. Not all memoryless deterministic strategies taken in $\widetilde{\mathcal{G}}$ are NC-strategies, since it contains the cycle $v_0v_1v_0$ of total weight 0. We thus apply the construction before, using the fact that $d(\text{☺}) = 0$, $d(v_1) = 1$ and $d(v_0) = 2$ (since the edge $(v_0,\text{☺})$ is not present in $\widetilde{\mathcal{G}}$). Thus, σ_1 is defined by $\sigma_1(v_0) = v_1$ and $\sigma_1(v_1) = \text{☺}$, and is indeed an NC-strategy.

▶ **Lemma 21.** *Strategy σ_1 is an NC-strategy, i.e. all cycles of $\widetilde{\mathcal{G}}$ conforming with σ_1 have a negative total weight.*

Proof. Let $v_1v_2\cdots v_kv_1$ be a cycle of $\widetilde{\mathcal{G}}$ that conforms to σ_1, with v_1 a vertex of minimal distance $d(v_1)$ among the ones of the cycle. We can choose v_1 such that it belongs to Min: otherwise, this would contradict the attractor computation in $\widetilde{\mathcal{G}}$. By Lemma 18(ii), its total weight is non-positive. Suppose that it is 0. Then, in the proof of Lemma 18(ii), all inequalities $\mathsf{mVal}^\rho(v_i) \geqslant \omega_i + \mathsf{mVal}^\rho(v_{i+1})$ are indeed equalities. In particular, $\mathsf{mVal}^\rho(v_1) = \omega_1 + \mathsf{mVal}^\rho(v_2)$. Since $v_2 \in \widetilde{E}(v_1)$, (6) ensures that all successors $v' \in \mathrm{supp}(\rho(v_1))$, $\mathsf{mVal}^\rho(v_1) = \omega(v_1,v') + \mathsf{mVal}^\rho(v')$. Since v_1 has minimal distance among all vertices of the cycle, it exists $v' \in \widetilde{E}(v_1)$ such that $d(v') = d(v_1) - 1$. But $d(v_2) \geqslant d(v_1) > d(v')$, which contradicts the choice of v_2 for $\sigma_1(v_1)$ in (7). ◀

Proof of Proposition 17. Let π be a play conforming to σ, from vertex v_0. Since σ is a switching strategy, it necessarily reaches T. If σ conforms with σ_1, by Lemma 18(i), it has a total-payoff $\mathbf{TP}(\pi) \leqslant \mathsf{mVal}^\rho(v_0)$. Otherwise, it is obtained by a switch, and is thus longer than

$\alpha = \max(0, |V|W - \mathsf{mVal}^\rho(v_0)) \times |V| + 1$. Then, it contains at least $\max(0, |V|W - \mathsf{mVal}^\rho(v_0))$ elementary cycles, before it switches to the attractor strategy σ_2. Once we remove the cycles, it remains a play of length at most $|V|$, and thus of total payoff at most $|V|W$. Since all cycles conforming to σ_1 have a total weight at most -1, by Lemma 21, $\mathbf{TP}(\pi)$ is at most $(-1) \times \max(0, |V|W - \mathsf{mVal}^\rho(v_0)) + |V|W \leqslant \mathsf{mVal}^\rho(v_0)$. ◀

This concludes the proof of Theorem 6.

5 Characterisation of optimality

All shortest-path games admit an optimal deterministic strategy for both players: however, as we have seen in Example 1, Min may require memory to play optimally. In this case, we also have seen in Example 4 that Min does not have an optimal memoryless (randomised) strategy: he only has ε-optimal ones, for all $\varepsilon > 0$. But some shortest-path games indeed admit optimal memoryless strategies for Min: the strategy ρ_p described in Section 3 is indeed optimal in graph games not containing negative cycles, for instance. In this final section, we characterise the shortest-path games in which Min admits an optimal memoryless strategy. For sure, Min does not have an optimal strategy if there is some vertex v of value $\mathsf{dVal}(v) = -\infty$.

▶ **Assumption.** In this last section, we therefore suppose that all shortest-path games are such that $\mathsf{dVal}(v) \neq -\infty$ for all vertices v.

We first recall the computations performed in [3] to compute values $\mathsf{dVal}(v)$. It consists of an iterated computation, called *value iteration* based on the operator $\mathcal{F} \colon (\mathbb{Z} \cup \{+\infty\})^V \to (\mathbb{Z} \cup \{+\infty\})^V$ defined for all $x = (x_v)_{v \in V} \in (\mathbb{Z} \cup \{+\infty\})^V$ and all vertices $v \in V$ by

$$\mathcal{F}(x)_v = \begin{cases} 0 & \text{if } v \in T \\ \min_{v' \in E(v)} (\omega(v, v') + x_{v'}) & \text{if } v \in V_{\mathsf{Min}} \\ \max_{v' \in E(v)} (\omega(v, v') + x_{v'}) & \text{if } v \in V_{\mathsf{Max}} \end{cases}$$

We let $f_v^{(0)} = 0$ if $v \in T$ and $+\infty$ otherwise. By monotony of $\mathcal{F}$, the sequence $(f^{(i)} = \mathcal{F}^i(f^{(0)}))_{i \in \mathbb{N}}$ is non-increasing. It is proved to be stationary, and convergent towards $(\mathsf{dVal}(v))_{v \in V}$, the smallest fixed-point of $\mathcal{F}$. The pseudo-polynomial complexity of solving shortest-path games comes from the fact that this sequence may becomes stationary after a pseudo-polynomial (and not polynomial) number of steps: the game of Figure 1 is one of the typical examples.

We introduce a new notion, being the most permissive strategy of Min at each step $i \geqslant 0$ of the computation. It maps each vertex $v \in V_{\mathsf{Min}}$ to the set

$$\widetilde{E}^{(i)}(v) = \{v' \in E(v) \mid \omega(v, v') + f_{v'}^{(i-1)} = f_v^{(i)}\}$$

of vertices that Min can choose. For each such most permissive strategy $\widetilde{E}^{(i)}$, we let $\widetilde{\mathcal{G}}^{(i)}$ be the game graph where we remove all edges (v, v') with $v \in V_{\mathsf{Min}}$ and $v' \notin \widetilde{E}^{(i)}(v)$. This allows us to state the following result, whose proof is in Appendix B.

▶ **Proposition 22.** *The following assertions are equivalent:*

1. Min *has an optimal memoryless deterministic strategy in* $\mathcal{G}$ *(for* dVal*);*
2. Min *has an optimal memoryless (randomised) strategy in* $\mathcal{G}$ *(for* $\overline{\mathsf{mVal}}$*);*

3. $f_v^{(|V|-1)} = f_v^{(|V|)} = \mathsf{dVal}(v)$ *for all vertices* v *(this means that the sequence* $(f^{(i)})$ *is stationary as soon as step* $|V|-1$*), and* Min *can guarantee to reach* T *from all vertices in the game graph* $\widetilde{\mathcal{G}}^{(|V|-1)}$.

This characterisation of the existence of optimal memoryless strategy is testable in polynomial time since it is enough to compute vectors $f^{(|V|-1)}$ and $f^{(|V|)}$, check their equality, compute the sets $\widetilde{E}^{(|V|-1)}(v)$ (this can be done while computing $f^{(|V|)}$) and check whether Min can guarantee reaching the target in $\widetilde{\mathcal{G}}^{(|V|-1)}$ by an attractor computation. The proof of implication $3 \Rightarrow 1$ is constructive and actually allows one to build an optimal memoryless deterministic strategy when it exists.

6 Discussion

This article studies the tradeoff between memoryless and deterministic strategies, showing that Min guarantees the same value when restricted to these two kinds of strategies. We also studied the existence of optimal memoryless strategies, which turns out to be equivalent to the existence of optimal memoryless deterministic strategies, and testable in polynomial time.

We could also define a more general lower and upper values $\underline{\mathsf{Val}}(v)/\overline{\mathsf{Val}}(v)$ when we let Min and Max play unrestricted strategies (randomised and with memory). The Blackwell determinacy results [12] implies that, for such unrestricted strategies, shortest-path games are still determined so that $\overline{\mathsf{Val}}(v) = \underline{\mathsf{Val}}(v) = \mathsf{Val}(v)$. The reasoning of Section 4 only used the vector of values $(\mathsf{mVal}^{\rho}(v))_{v\in V}$ to define the deterministic switching strategy σ, without using anywhere that ρ is memoryless. We thus indeed showed that $\mathsf{dVal}(v) \leqslant \mathsf{Val}(v)$. However, the proof of Section 3 is not directly translatable if we allow Min to use memory and randomisation. In particular, we know nothing anymore about how Max can react, which may break the result of Proposition 10. We leave this further study for future work.

References

1 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

2 Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Math. Oper. Res.*, 16(3):580–595, 1991.

3 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, and Benjamin Monmege. Pseudopolynomial iterative algorithm to solve total-payoff games and min-cost reachability games. *Acta Informatica*, 54, July 2016.

4 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, and Benjamin Monmege. Pseudopolynomial iterative algorithm to solve total-payoff games and min-cost reachability games. *Acta Informatica*, 54(1):85–125, February 2017. doi:10.1007/s00236-016-0276-z.

5 Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. Trading memory for randomness. In *Proceedings of the The Quantitative Evaluation of Systems, First International Conference*, QEST '04, pages 206–217, Washington, DC, USA, 2004. IEEE Computer Society. URL: http://dl.acm.org/citation.cfm?id=1025129.1026090.

6 Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdziński. Mean-payoff parity games. In *Proceedings of the 20th Annual Symposium on Logic in Computer Science (LICS'05)*, pages 178–187. IEEE Computer Society Press, 2005.

7 Krishnendu Chatterjee, Thomas A. Henzinger, and Vinayak S. Prabhu. Trading infinite memory for uniform randomness in timed games. In *Hybrid Systems: Computation and Control, 11th International Workshop, HSCC 2008, St. Louis, MO, USA, April 22-24, 2008. Proceedings*, pages 87–100, 2008. doi:10.1007/978-3-540-78929-1_7.

8 Krishnendu Chatterjee, Mickael Randour, and Jean-François Raskin. Strategy synthesis for multi-dimensional quantitative objectives. *Acta Informatica*, 51:129–163, 2014. doi:10.1007/s00236-013-0182-6.

9 Hugo Gimbert and Wiesław Zielonka. When can you play positionally? In *Proceedings of the 29th International Conference on Mathematical Foundations of Computer Science (MFCS'04)*, volume 3153 of *Lecture Notes in Computer Science*, pages 686–698. Springer, 2004.

10 Erich Grädel, Wolfgang Thomas, and Thomas Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

11 Leonid Khachiyan, Endre Boros, Konrad Borys, Khaled Elbassioni, Vladimir Gurvich, Gabor Rudolf, and Jihui Zhao. On short paths interdiction problems: Total and node-wise limited interdiction. *Theory of Computing Systems*, 43:204–233, 2008.

12 Donald A. Martin. The determinacy of Blackwell games. *The Journal of Symbolic Logic*, 63(4):1565–1581, 1998.

13 John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36(1):48–49, 1950.

A Computations for proof of Proposition 9

A.1 Computations for $\gamma_{0,\mathbb{N}} + \gamma_{<I,\geqslant L} \leqslant \mathsf{dVal}^\sigma(v_0)$

When $\mathsf{dVal}^\sigma(v_0) < 0$ and $\Pi_{<I,\geqslant L} \neq \emptyset$, it remains to show under which conditions over p,

$$S = \sum_{i=1}^{I-1} \sum_{\ell=L}^{\infty} (1-p)^i p^\ell \left(i w^+ + \left\lfloor \frac{\ell - ic - |V|}{c} \right\rfloor (-w^-) + |V| W \right) \leqslant \mathsf{dVal}^\sigma(v_0)$$

Upper-bounding $\left\lfloor \frac{\ell-ic-|V|}{c} \right\rfloor (-w^-)$ by $\left(\frac{\ell-ic-|V|}{c} - 1 \right)(-w^-) = \frac{\ell-ic-|V|-c}{c}(-w^-)$, we can split the double sum S in three parts:

$$S = (w^+ + w^-) \underbrace{\sum_{i=1}^{I-1} \sum_{\ell=L}^{\infty} (1-p)^i p^\ell i}_{S_1} - \frac{w^-}{c} \underbrace{\sum_{i=1}^{I-1} \sum_{\ell=L}^{\infty} (1-p)^i p^\ell \ell}_{S_2} + \left(\frac{-|V|-c}{c}(-w^-) + |V| W \right) \underbrace{\sum_{i=1}^{I-1} \sum_{\ell=L}^{\infty} (1-p)^i p^\ell}_{S_3}$$

Using the fact that $L \geqslant 2$ ($L = I\gamma + \frac{2|\mathsf{dVal}^\sigma(v_0)|+|V|W}{w^-} c + |V| > |V| > 1$ otherwise, for the unique $v \in V$, $\mathsf{dVal}(v) = 0$ or $+\infty$ regarding $v \in T$ or not), we have

$$S_1 \leqslant \sum_{i=1}^{\infty} i(1-p)^i \sum_{\ell=2}^{\infty} p^\ell = \frac{1-p}{p^2} \times \frac{p^2}{1-p} = 1$$

$$S_3 \leqslant \sum_{i=1}^{\infty} (1-p)^i \sum_{\ell=1}^{\infty} p^\ell = \frac{1-p}{p} \times \frac{p}{1-p} = 1$$

and

$$\begin{aligned}
S_2 &= \sum_{i=1}^{I-1}(1-p)^i \sum_{\ell=L}^{\infty} p^\ell \ell \\
&= (1-p)\frac{1-(1-p)^{I-1}}{p} \times \frac{p^L(-Lp+L+p)}{(1-p)^2} \\
&= \frac{(1-(1-p)^{I-1})p^{L-1}(-Lp+L+p)}{1-p} \\
&\geqslant \frac{(1-(1-p)^{I-1})p^L}{1-p} && \text{(since } -Lp+L \geqslant 0\text{)} \\
&\geqslant \frac{1}{4(1-p)} && \text{(since } p \geqslant \tfrac{1}{2^{1/L}} \geqslant \tfrac{1}{2} \text{ and } 1-(1-p)^{I-1} \geqslant \tfrac{1}{2} \text{ by } 0 \leqslant I\text{)}
\end{aligned}$$

Therefore, we obtain

$$S \leqslant (w^+ + w^-) - \frac{w^-}{c}\frac{1}{4(1-p)} + \left(\frac{-|V|-c}{c}(-w^-) + |V|W\right)$$

The right term goes towards $-\infty$ when $p \to 1$. In particular, when

$$p \geqslant 1 - \frac{w^-}{4(cw^+ + 2cw^- + |V|w^- + c|V|W - \mathsf{dVal}^\sigma(v_0)c)}$$

we obtain

$$S \leqslant \mathsf{dVal}^\sigma(v_0)$$

A.2 Computations for $\gamma_{\geqslant I,\mathbb{N}} \leqslant \varepsilon/4$

It remains to show that

$$(1-p)^I\left(\frac{w^+}{p}I + \frac{w^+(1-p)}{p^2} + \frac{|V|W}{p}\right) \leqslant \frac{\varepsilon}{4}$$

We let here $\delta = \frac{2|\mathsf{dVal}^\sigma(v_0)|+|V|W}{w^-}c + |V|$ so that $L = I\gamma + \delta$. Since, $p \geqslant LW/(LW+\varepsilon/4) = (I\gamma W + \delta W)/(I\gamma W + \delta W + \varepsilon/4)$,

$$1-p \leqslant \frac{\varepsilon/4}{I\gamma W + \delta W + \varepsilon/4} = \frac{1}{4I\gamma W/\varepsilon + 4\delta W/\varepsilon + 1}$$

By also using that $p \geqslant 1/2 \geqslant 1/4$, thus $1/p \leqslant 4$ and $1/p^2 \leqslant 4$, we obtain

$$\gamma_{\geqslant I,\mathbb{N}} \leqslant \left(\frac{1}{4I\gamma W/\varepsilon + 4\delta W/\varepsilon + 1}\right)^I \left(4w^+ I + 4(w^+ + |V|W)\right)$$

The value $4I\gamma W/\varepsilon + 4\delta W/\varepsilon + 1$ being greater than 1, we can write

$$\gamma_{\geqslant I,\mathbb{N}} \leqslant \left(\frac{1}{4I\gamma W/\varepsilon + 4\delta W/\varepsilon + 1}\right)^{I-1} \left(4w^+ \frac{I}{4I\gamma W/\varepsilon + 4\delta W/\varepsilon + 1} + 4(w^+ + |V|W)\right)$$

Since $x/(ax+b) \leqslant 1/a$ whenever $a, x, b \geqslant 0$, we have $\frac{I}{4I\gamma W/\varepsilon + 4\delta W/\varepsilon + 1} \leqslant \frac{\varepsilon}{4\gamma W}$. Moreover, $\frac{4I\gamma W}{\varepsilon} + \frac{4\delta W}{\varepsilon} + 1 > \frac{I\gamma W}{2\varepsilon}$ and thus

$$\gamma_{\geqslant I,\mathbb{N}} \leqslant \left(\frac{2\varepsilon}{I\gamma W}\right)^{I-1} \left(\frac{\varepsilon w^+}{\gamma W} + 4(w^+ + |V|W)\right)$$

But

$$\left(\frac{2\varepsilon}{I\gamma W}\right)^{I-1}\left(\frac{\varepsilon w^+}{\gamma W}+4(w^++|V|W)\right)\leqslant\frac{\varepsilon}{4}$$

if and only if

$$\left(\frac{I\gamma W}{2\varepsilon}\right)^{I-1}\geqslant\frac{4w^+}{\gamma W}+\frac{16(w^++|V|W)}{\varepsilon}\geqslant\frac{2w^+}{\gamma W}+\frac{8(w^++|V|W)}{\varepsilon}$$

if and only if

$$(I-1)\ln\left(\frac{I\gamma W}{2\varepsilon}\right)\geqslant\ln\left(\frac{2w^+}{\gamma W}+\frac{8(w^++|V|W)}{\varepsilon}\right)=\ln\left(\frac{\xi\gamma W}{2\varepsilon}\right)$$

where $\xi=\frac{4\varepsilon w^+}{\gamma^2W^2}+\frac{16(w^++|V|W)}{\gamma W}$. Consider ε small enough so that $\gamma W/2\varepsilon\geqslant 1$ and $\xi\gamma W/2\varepsilon\geqslant 2$ (the two terms tend to $+\infty$ when ε tends to 0). Then, $(I-1)\ln\left(\frac{I\gamma W}{2\varepsilon}\right)\geqslant(I-1)\ln(I)$, and it is sufficient to prove that

$$(I-1)\ln(I)\geqslant\ln\left(\frac{\xi\gamma W}{2\varepsilon}\right)$$

Since the mapping $I\mapsto(I-1)\ln(I)$ is increasing, and $I\geqslant\frac{\xi\gamma W}{2\varepsilon}$ (by definition),

$$(I-1)\ln(I)\geqslant\left(\frac{\xi\gamma W}{2\varepsilon}-1\right)\ln\left(\frac{\xi\gamma W}{2\varepsilon}\right)\geqslant\ln\left(\frac{\xi\gamma W}{2\varepsilon}\right)$$

A.3 Lower bound over p

If we gather all the lower bounds over p that we need in the proof, we get that:

- if $\mathsf{dVal}^\sigma(v_0)\geqslant 0$, we must have

$$p\geqslant\max\left(\frac{LW}{LW+\varepsilon/4},\frac{1}{2}\right)$$

- if $\mathsf{dVal}^\sigma(v_0)<0$, we must have

$$\max\left(\frac{LW}{LW+\varepsilon/4},\frac{1}{2^{1/L}},\left(1-\frac{\varepsilon}{2|\mathsf{dVal}^\sigma(v_0)|}\right)^{\frac{1}{|V|}},\right.$$
$$\left.1-\frac{w^-}{4(cw^++2cw^-+|V|w^-+c|V|W+|\mathsf{dVal}^\sigma(v_0)|c)}\right)$$

  with ε small enough so that this bound is less than 1.

B Proof of Proposition 22

Implication $1\Rightarrow 2$ is trivial by the result of Theorem 6.

For implication $3\Rightarrow 1$, consider any memoryless deterministic strategy σ^* that guarantees Min to reach T from all vertices in the game graph $\widetilde{\mathcal{G}}^{(|V|-1)}$. Then, for all vertices v, we show by induction on n, that each play π from v that reaches the target in at most n steps, and conforming to σ^*, has a total-payoff $\mathbf{TP}(\pi)\leqslant\mathsf{dVal}(v)$. This is trivial for $n=0$. If $\pi=v\pi'$ with π' starting in v, then

$$\mathbf{TP}(\pi)=\omega(v,v')+\mathbf{TP}(\pi')\leqslant\omega(v,v')+\mathsf{dVal}(v')=\omega(v,v')+f_v^{(|V|-1)}$$

If $v \in V_{\mathsf{Max}}$, we have

$$\mathbf{TP}(\pi) \leqslant \omega(v,v') + f_v^{(|V|-1)} \leqslant f_v^{(|V|)} = \mathsf{dVal}(v)$$

If $v \in V_{\mathsf{Min}}$, since $v' \in \widetilde{E}^{(|V|-1)}(v)$,

$$\mathbf{TP}(\pi) = f_v^{(|V|)} = \mathsf{dVal}(v)$$

This ends the proof by induction. To conclude that 1 holds, since σ^* guarantees to reach the target, all plays conforming to it reach the target in less than $|V|$ steps, which proves that $\mathsf{dVal}^{\sigma^*}(v) \leqslant \mathsf{dVal}(v)$, showing that σ^* is optimal.

For implication $1 \Rightarrow 3$, consider an optimal deterministic memoryless strategy σ^*, such that for all v, $\mathsf{dVal}^{\sigma^*}(v) = \mathsf{dVal}(v)$.

First, we show that $f_v^{(|V|-1)} = \mathsf{dVal}(v)$ for all vertices v. For that, consider the deterministic strategy τ of Max defined for all finite plays π having $n \leqslant |V|$ vertices, ending in a vertex $v \in V_{\mathsf{Max}}$, by $\tau(\pi) = v'$ such that $\omega(v,v') + f_{v'}^{(|V|-1-n)} = f_v^{(|V|-n)}$. For longer finite plays, we define τ arbitrarily. Then, let π be the play from v conforming to σ^* and τ. Since σ^* ensures reaching the target and is memoryless deterministic, π reaches the target in at most $|V|-1$ steps. Let $\pi = v_0 v_1 v_2 \cdots v_{k-1} v_k$ with $k \leqslant |V|$. Let us show that $\mathbf{TP}(\pi) \geqslant f_v^{(|V|-1)}$. We prove by induction on $0 \leqslant j \leqslant k$ that

$$\sum_{i=j}^{k-1} \omega(v_i, v_{i+1}) \geqslant f_{v_j}^{(|V|-1-j)}$$

When $j = k-1$, the result is trivial since the sum is

$$0 = f_{v_k}^{(0)} \geqslant f_{v_k}^{(|V|-1-(k-1))}$$

Otherwise, by induction hypothesis

$$\sum_{i=j}^{k-1} \omega(v_i, v_{i+1}) \geqslant \omega(v_j, v_{j+1}) + f_{v_{j+1}}^{(|V|-1-(j+1))}$$

If $v_j \in V_{\mathsf{Max}}$, v_{j+1} is chosen by τ so that

$$\omega(v_j, v_{j+1}) + f_{v_{j+1}}^{(|V|-1-(j+1))} = f_{v_j}^{(|V|-1-j)}$$

If $v \in V_{\mathsf{Min}}$, by definition of $\mathcal{F}$,

$$\omega(v_j, v_{j+1}) + f_{v_{j+1}}^{(|V|-1-(j+1))} \geqslant f_{v_j}^{(|V|-1-j)}$$

We can conclude in all cases, so that $f_v^{(|V|-1)} = \mathsf{dVal}(v)$ for all vertices v.

Then, we show that Min can guarantee to reach T from all vertices in the game graph $\widetilde{\mathcal{G}}^{(|V|-1)}$. Let us suppose that this is not the case. Then, there exists a set V' of vertices in which Max can guarantee to keep Min for ever, in the game $\widetilde{\mathcal{G}}^{(|V|-1)}$: for all $v' \in V' \cap V_{\mathsf{Min}}$, $\widetilde{E}^{(|V|-1)}(v') \subseteq V'$, and for all $v' \in V' \cap V_{\mathsf{Max}}$, $E(v) \cap V' \neq \emptyset$. Since σ^* guarantees to reach the target, there exists $v \in V' \cap V_{\mathsf{Min}}$ such that $\sigma^*(v) = v' \notin V'$: then $\omega(v,v') + \mathsf{dVal}(v') > \mathsf{dVal}(v)$ (here we use that $\mathsf{dVal}(v) = f_v^{(|V|-1)} = f_v^{(|V|)}$). Consider an optimal deterministic memoryless strategy τ^* of Max in $\mathcal{G}$. Then, the play π from v conforming to σ^* and τ^* starts by taking the edge (v,v') and continues with a play π'. By optimality, we know that $\mathbf{TP}(\pi) = \mathsf{dVal}(v)$ and $\mathbf{TP}(\pi') = \mathsf{dVal}(v')$. However,

$$\mathbf{TP}(\pi) = \omega(v,v') + \mathbf{TP}(\pi') = \omega(v,v') + \mathsf{dVal}(v') > \mathsf{dVal}(v)$$

which raises a contradiction.

We finish the proof by showing $2 \Rightarrow 1$. For that, consider an optimal memoryless strategy ρ^* for $\overline{\mathsf{mVal}}$. By following the construction of Section 4, we build a memoryless deterministic strategy σ_1. Lemma 21 ensures that σ_1 is an NC-strategy so that every cycle conforming to σ_1 has a negative total weight. Let us show that such a negative cycle cannot exist, which will ensure that all plays conforming to σ_1 reach the target, and thus the optimality of σ_1. Suppose that a cycle $v_1 v_2 \cdots v_k v_1$ conforms to σ_1. By following the notations of the proof of Lemma 18(ii), we suppose that v_1 is a vertex of minimal distance $d(v_1)$ to the target, and that it is owned by V_{Min}. Note that such a vertex exists, otherwise only Max has the minimal distance vertices on the cycle and that contradicts the attractor computation. By minimality of $d(v_1)$ among the vertices of the cycle, $d(v_2) \geqslant d_{v_1}$. Moreover, by the attractor computation, there exists $u \in E(v_1)$ such that $d(u) = d(v_1) - 1 < d(v_1)$. By definition of σ_1, we know for sure that $u \notin \widetilde{E}(v_1)$, so that

$$\omega(v_1, u) + \mathsf{mVal}^{\rho^*}(u) > \omega(v_1, v_2) + \mathsf{mVal}^{\rho^*}(v_2)$$

By (6), we know that in this case

$$\mathsf{mVal}^{\rho^*}(v_1) > \omega(v_1, v_2) + \mathsf{mVal}^{\rho^*}(v_2)$$

By optimality of ρ^*, this rewrites in

$$\overline{\mathsf{mVal}}(v_1) > \omega(v_1, v_2) + \overline{\mathsf{mVal}}(v_2)$$

By Theorem 6, this also rewrites in

$$\mathsf{dVal}(v_1) > \omega(v_1, v_2) + \mathsf{dVal}(v_2) \geqslant \mathcal{F}\big((\mathsf{dVal}(v))_{v \in V}\big)(v_1)$$

(since $v_1 \in V_{\mathsf{Min}}$): this contradicts the fact that the vector $(\mathsf{dVal}(v))_{v \in V}$ is a fixed-point of $\mathcal{F}$.

Characteristic Logics for Behavioural Metrics via Fuzzy Lax Extensions

Paul Wild
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
paul.wild@fau.de

Lutz Schröder
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
lutz.schroeder@fau.de

Abstract

Behavioural distances provide a fine-grained measure of equivalence in systems involving quantitative data, such as probabilistic, fuzzy, or metric systems. Like in the classical setting of crisp bisimulation-type equivalences, the wide variation found in system types creates a need for generic methods that apply to many system types at once. Approaches of this kind are emerging within the paradigm of universal coalgebra, based either on lifting pseudometrics along set functors or on lifting general real-valued (*fuzzy*) relations along functors by means of *fuzzy lax extensions*. An immediate benefit of the latter is that they allow bounding behavioural distance by means of fuzzy bisimulations that need not themselves be (pseudo-)metrics, in analogy to classical bisimulations (which need not be equivalence relations). The known instances of generic pseudometric liftings, specifically the generic Kantorovich and Wasserstein liftings, both can be extended to yield fuzzy lax extensions, using the fact that both are effectively given by a choice of quantitative modalities. Our central result then shows that in fact all fuzzy lax extensions are Kantorovich extensions for a suitable set of quantitative modalities, the so-called *Moss modalities*. For *non-expansive* fuzzy lax extensions, this allows for the extraction of quantitative modal logics that characterize behavioural distance, i.e. satisfy a quantitative version of the Hennessy-Milner theorem; equivalently, we obtain expressiveness of a quantitative version of Moss' coalgebraic logic.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics

Keywords and phrases Modal logic, behavioural distance, coalgebra, bisimulation, lax extension

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.27

Related Version A full version of the paper is available at `https://arxiv.org/abs/2007.01033`.

Funding Work forms part of the DFG project *Probabilistic description logics as a fragment of probabilistic first-order logic* (SCHR 1118/6-2).

1 Introduction

Branching-time equivalences on reactive systems are typically governed by notions of *bisimilarity* [43, 37]. For systems involving quantitative data, such as transition probabilities, fuzzy truth values, or labellings in metric spaces, it is often appropriate to use more fine-grained, *quantitative* measures of behavioural similarity, arriving at notions of *behavioural distance*. Distance-based approaches in particular avoid the problem that small quantitative deviations in behaviour will typically render two given systems inequivalent under two-valued notions of equivalence, losing information about their similarity.

Behavioural distances serve evident purposes in system verification, allowing as they do for a reasonable notion of a specification being satisfied up to an acceptable margin of deviation (e.g. [24]). Applications have also been proposed in differential privacy [9] and conformance testing of hybrid systems [30]. Like their two-valued counterparts, behavioural distances have been introduced for quite a range of system types, such as various forms of

probabilistic labelled transition systems or labelled Markov processes [25, 53, 14, 16]; systems combining nondeterministic and probabilistic branching variously known as nondeterministic probabilistic transition systems [8], probabilistic automata [13], and Markov decision processes [22]; weighted automata [3]; fuzzy transition systems [7] and fuzzy Kripke models [21]; and various forms of metric transition systems [12, 20, 19]. This range of variation creates a need for unifying concepts and methods. The present work contributes to developing such a unified view within the framework of universal coalgebra, which is based on abstracting a wide range of system types (including all the mentioned ones) as set functors.

Specifically, we fix a generic notion of *quantitative bisimulation* via the key notion of *non-expansive (fuzzy) lax extension* of a functor. While existing coalgebraic approaches to behavioural pseudometrics rely on pseudometric liftings of functors [2], fuzzy lax extensions act on unrestricted quantitative relations. Hence, quantitative bisimulations need not themselves be pseudometrics, in analogy to classical bisimulations not needing to be equivalence relations, and thus may serve as small certificates for low behavioural distance. We show that two known systematic constructions of functor liftings from chosen sets of modalities, the generic Wasserstein and Kantorovich liftings, both extend to yield non-expansive fuzzy lax extensions (it is essentially known that the Wasserstein lifting yields a fuzzy lax extension [26]). As our main result, we then establish that *every* fuzzy lax extension of a finitary functor is a Kantorovich extension induced by a suitable set of modalities, the so-called Moss modalities.

This result may be seen as a quantitative version of previous results asserting the existence of separating sets of two-valued modalities for finitary functors [47, 32, 35], which allow for generic Hennessy-Milner-type theorems stating that states in finitely branching systems (coalgebras) are behaviourally equivalent iff they satisfy the same modal formulae [44, 47]. Indeed our main result similarly allows *extracting characteristic quantitative modal logics* from given behavioural metrics, where a logic is *characteristic* or *expressive* if the induced logical distance of states coincides with behavioural distance. This result may equivalently be phrased as expressiveness of a quantitative version of Moss' coalgebraic logic [42], which provides a coalgebraic generalization of the classical relational ∇-modality (which e.g. underlies the $a \to \Psi$ notation used in Walukiewicz's μ-calculus completeness proof [55]). We relax the standard requirement of finite branching, i.e. use of finitary functors, to an approximability condition called *finitary separability*, and hence in particular cover countable probabilistic branching.

Organization. We recall basic concepts on pseudometrics, coalgebraic bisimilarity, and coalgebraic logic in Section 2. The central notion of (nonexpansive) fuzzy lax extension is introduced in Section 3, and the arising principle of quantitative bisimulation in Section 4. The generic Kantorovich and Wasserstein liftings are discussed in Sections 5 and 6, respectively. Our central result showing that every lax extension is a Kantorovich lifting is established in Section 7. In Section 8, we show how our results amount to extracting characteristic modal logics from given non-expansive lax extensions. Proofs are sometimes omitted or only sketched; some additional proofs are in Appendix A, a full version with all proofs is available [56].

Related Work. Probabilistic quantitative characteristic modal logics go back to Desharnais et al. [16]; they relate to fragments of quantitative μ-calculi [29, 38, 40]. A further well-known class of quantitative modal logics are fuzzy modal and description logics (e.g. [41, 23, 49, 34]). Van Breugel and Worrell [53] prove a Hennessy-Milner theorem for quantitative probabilistic modal logic. Quantitative Hennessy-Milner-type theorems have since been established for

fuzzy modal logic with Gödel semantics [21], for systems combining probability and non-determinism [17], and for Heyting-valued modal logics [18] as introduced by Fitting [23]. König and Mika-Michalski [31] provide a quantitative Hennessy-Milner theorem in coalgebraic generality for the case where behavioural distance is induced by the pseudometric Kantorovich lifting defined by the same set of modalities as the logic, a result that we complement by showing that in fact all fuzzy lax extensions are Kantorovich.

Fuzzy lax extensions are a quantitative version of lax extensions [35, 50, 33], which in turn belong to an extended strand of research on relation liftings [28, 50, 33]. They appear to go back to work on monoidal topology [27], and have been used in work on applicative bisimulation [24]; as indicated above, Hofmann [26] effectively already introduces the generic Wasserstein lax extension (without using the term but proving the relevant properties, except non-expansiveness). Our notion of *non-expansive* lax extension, which is central to the connection with characteristic logics, appears to be new. Our method of extracting quantitative modalities from fuzzy lax extensions generalizes the construction of two-valued Moss liftings for (two-valued) lax extensions [32, 35].

2 Preliminaries

We recall basic notions on pseudometrics, universal coalgebra [46], and the generic treatment of two-valued bisimilarity. Basic knowledge of category theory (e.g. [1]) will be helpful.

Pseudometric Spaces. A *(1-bounded) pseudometric* on a set X is a function $d\colon X\times X\to[0,1]$ satisfying $d(x,x)=0$ (reflexivity), $d(x,y)=d(y,x)$ (symmetry), and $d(x,z)\leq d(x,y)+d(y,z)$ (triangle inequality) for $x,y,z\in X$. If moreover $d(x,y)=0$ implies $x=y$, then d is a *metric*. The pair (X,d) is a *(pseudo-)metric space*. The unit interval $[0,1]$ is a metric space under Euclidean distance $d_E(x,y)=|x-y|$. The *supremum distance* of functions $f,g\colon X\to[0,1]$ is $\|f-g\|_\infty=\sup_{x\in X}|f(x)-g(x)|$. A map $f\colon X\to Y$ of pseudometric spaces (X,d_1), (Y,d_2), is *nonexpansive* (notation: $f\colon(X,d_1)\to_1(Y,d_2)$) if $d_2(f(x),f(y))\leq d_1(x,y)$ for all $x,y\in X$.

Universal Coalgebra is a uniform framework for a broad range of state-based system types. It is based on encapsulating the transition type of a system as an (endo-)functor, for the present purposes on the category of sets: A *functor* T assigns to each set X a set TX, and to each map $f\colon X\to Y$ a map $Tf\colon TX\to TY$, preserving identities and composition. We may think of TX as a parametrized datatype; e.g. the *(covariant) powerset functor* $\mathcal{P}$ assigns to each set X its powerset $\mathcal{P}X$, and to $f\colon X\to Y$ the direct image map $\mathcal{P}f\colon\mathcal{P}X\to\mathcal{P}Y, A\mapsto f[A]$; and the *distribution functor* $\mathcal{D}$ maps each set X to the set of discrete probability distributions on X. Recall that a discrete probability distribution on X is given by a *probability mass function* $\mu\colon X\to[0,1]$ such that $\sum_{x\in X}\mu(x)=1$ (implying that the *support* $\{x\in X\mid\mu(x)>0\}$ of μ is at most countable); we abuse μ to denote also the induced probability measure, writing $\mu(A)=\sum_{x\in A}\mu(x)$ for $A\subseteq X$. Moreover, $\mathcal{D}$ maps $f\colon X\to Y$ to $\mathcal{D}f\colon\mathcal{D}X\to\mathcal{D}Y$, $\mu\mapsto\mu f^{-1}$ where the *image measure* μf^{-1} is given by $\mu f^{-1}(B)=\mu(f^{-1}[B])$ for $B\subseteq Y$.

Systems of a transition type T are then cast as *T-coalgebras* (A,α), consisting of a set A of *states* and a *transition function* $\alpha\colon A\to TA$, thought of as assigning to each state a structured collection of successors. E.g. a $\mathcal{P}$-coalgebra $\alpha\colon A\to\mathcal{P}A$ assigns to each state a a set $\alpha(a)$ of successors, so is just a (non-deterministic) transition system. Similarly, a $\mathcal{D}$-coalgebra assigns to each state a distribution over successor states, and thus is a probabilistic transition system or a Markov chain. A *morphism* $f\colon(A,\alpha)\to(B,\beta)$ of T-coalgebras (A,α) and (B,β) is a map $f\colon A\to B$ such that $\beta\circ f=Tf\circ\alpha$, where $\circ$ denotes the usual (applicative) composition of functions; e.g. morphisms of $\mathcal{P}$-coalgebras are functional bisimulations.

A functor T is *finitary* if for each set X and each $t \in TX$, there exists a finite subset $Y \subseteq X$ such that $t = Ti(t')$ for some $t' \in TY$, where $i\colon Y \to X$ is the inclusion map. Intuitively, T is finitary if every element of TX mentions only finitely many elements of X. Every set functor T has a *finitary part* T_ω given by $T_\omega X = \bigcup\{Ti[TY] \mid Y \subseteq X \text{ finite}, i\colon Y \to X \text{ inclusion}\}$. E.g. $\mathcal{P}_\omega$, the *finite powerset functor*, maps a set to the set of its finite subsets, and $\mathcal{D}_\omega$, the *finite distribution functor*, maps a set X to the set of discrete probability distributions on X with finite support. Coalgebras for finitary functors generalize finitely branching systems, and hence feature in Hennessy-Milner type theorems, which typically fail under infinite branching.

Bisimilarity and Lax Extensions. Coalgebras come with a canonical notion of observable equivalence: States $a \in A$, $b \in B$ in T-coalgebras (A, α), (B, β) are *behaviourally equivalent* if there exist a coalgebra (C, γ) and morphisms $f\colon (A, \alpha) \to (C, \gamma)$, $g\colon (B, \beta) \to (C, \gamma)$ such that $f(a) = g(b)$. Behavioural equivalence can often be characterized in terms of bisimulation relations, which may provide small witnesses for behavioural equivalence of states and in particular need not form equivalence relations. The most general known way of treating bisimulation coalgebraically is via *lax extensions* L of the functor T, which map relations $R \subseteq X \times Y$ to $LR \subseteq TX \times TY$ subject to a number of axioms (monotonicity, preservation of relational converse, lax preservation of composition, extension of function graphs) [35]; L *preserves diagonals* if $L\Delta_X = \Delta_{TX}$ for each set X, where for any set Y, Δ_Y denotes the *diagonal* $\{(y, y) \mid y \in Y\}$. The *Barr extension* $\overline{T}$ of T [4, 51] is defined by

$$\overline{T}R = \{(T\pi_1(r), T\pi_2(r)) \mid r \in TR\}$$

for $R \subseteq X \times Y$, where $\pi_1\colon R \to X$ and $\pi_2\colon R \to Y$ are the projections; $\overline{T}$ preserves diagonals, and is a lax extension if T preserves weak pullbacks. E.g., the Barr extension $\overline{\mathcal{P}}$ of the powerset functor $\mathcal{P}$ is the well-known Egli-Milner extension, given by

$$(V, W) \in \overline{\mathcal{P}}(R) \iff (\forall x \in V.\, \exists y \in W.\, (x, y) \in R) \wedge (\forall y \in W.\, \exists x \in V.\, (x, y) \in R)$$

for $R \subseteq X \times Y$, $V \in \mathcal{P}(X)$, $W \in \mathcal{P}(Y)$. An *L-bisimulation* between T-coalgebras (A, α), (B, β) is a relation $R \subseteq A \times B$ such that $(\alpha(a), \beta(b)) \in LR$ for all $(a, b) \in R$; e.g. for $L = \overline{\mathcal{P}}$, we obtain exactly Park/Milner bisimulation on transition systems. If L preserves diagonals, then two states are behaviourally equivalent iff they are related by some L-bisimulation [35].

Coalgebraic Logic serves as a generic framework for the specification of state-based systems [11]. It is based on interpreting custom *modalities* of given finite arity over coalgebras for a functor T as n-ary *predicate liftings*, which are families of maps

$$\lambda_X\colon (2^X)^n \to 2^{TX}$$

(subject to a naturality condition) where $2 = \{\bot, \top\}$ and for any set Y, 2^Y is the set of 2-valued predicates on Y. We do not distinguish notationally between modalities and the associated predicate liftings. Satisfaction of a formula of the form $\lambda(\phi_1, \dots, \phi_n)$ (in some ambient logic) in a state $a \in A$ of a T-coalgebra (A, α) is then defined inductively by

$$a \vDash \lambda(\phi_1, \dots, \phi_n) \text{ iff } \alpha(a) \in \lambda_A([\![\phi_1]\!], \dots, [\![\phi_n]\!]) \tag{1}$$

where for any formula ψ, $[\![\psi]\!] = \{c \in A \mid c \vDash \psi\}$. E.g. the standard diamond modality $\Diamond$ is interpreted over the powerset functor $\mathcal{P}$ by the predicate lifting $\Diamond_X(Y) = \{Z \in \mathcal{P}(X) \mid \exists x \in Z.\, Y(x) = \top\}$, which according to (1) induces precisely the usual semantics of $\Diamond$ over transition systems ($\mathcal{P}$-coalgebras). The standard Hennessy-Milner theorem is generalized

coalgebraically [44, 47] as saying that two states in T-coalgebras are behaviourally equivalent iff they satisfy the same Λ-formulae, provided that T is finitary (which corresponds to the usual assumption of finite branching) and Λ is *separating*, i.e. for any set X, every $t \in TX$ is uniquely determined (within TX) by the set

$$\{(\lambda, Y_1, \dots, Y_n) \mid \lambda \in \Lambda \text{ } n\text{-ary}, Y_1, \dots, Y_n \in 2^X, t \in \lambda(Y_1, \dots, Y_n)\}.$$

For finitary T, a separating set of modalities always exists [47].

3 Fuzzy Relations and Lax Extensions

We next introduce the central notion of the paper, concerning extensions of *fuzzy* (or *real-valued*) relations along a *set functor T, which we fix for the remainder of the paper*. We begin by fixing basic concepts and notation on fuzzy relations. Pseudometrics can be viewed as particular fuzzy relations, forming a quantitative analogue of equivalence relations.

▶ **Definition 3.1.** Let A and B be sets. A *fuzzy relation* between A and B is a map $R\colon A \times B \to [0,1]$, also written $R\colon A \nrightarrow B$. We say that R is *crisp* if $R(a,b) \in \{0,1\}$ for all $a \in A, b \in B$ (and generally apply the term crisp to concepts that live in the standard two-valued setting). The *converse* relation $R^\circ\colon B \nrightarrow A$ is given by $R^\circ(b,a) = R(a,b)$. For $R, S\colon A \nrightarrow B$, we write $R \leq S$ if $R(a,b) \leq S(a,b)$ for all $a \in A, b \in B$.

▶ **Convention 3.2.** Crisp relations are just ordinary relations. However, since we are working in a pseudometric setting, it will be more natural to use the convention that elements $a \in A, b \in B$ are related by a crisp relation R if $R(a,b) = 0$, in which case we write aRb.

▶ **Convention 3.3** (Composition)**.** We write composition of fuzzy relations diagrammatically, using ';'. Explicitly, the composite $R_1; R_2 : A \nrightarrow C$ of $R_1\colon A \nrightarrow B$ and $R_2\colon B \nrightarrow C$ is defined by

$$(R_1; R_2)(a,c) = \inf_{b \in B} R_1(a,b) \oplus R_2(b,c),$$

where $\oplus$ denotes Łukasiewicz disjunction: $x \oplus y = \min(x + y, 1)$. We reserve the applicative composition operator $\circ$ for composition of functions. In particular, $R\colon A \nrightarrow B$ is viewed as a function $A \times B \to [0,1]$ whenever $\circ$ is applied to R.

▶ **Definition 3.4** (Functions as relations)**.** The *ϵ-graph* of a function $f\colon A \to B$ is the fuzzy relation $\mathrm{Gr}_{\epsilon,f}\colon A \nrightarrow B$ given by $\mathrm{Gr}_{\epsilon,f}(a,b) = \epsilon$ if $f(a) = b$, and $\mathrm{Gr}_{\epsilon,f}(a,b) = 1$ otherwise. The ϵ-graph of the identity function id_A is also called the *ϵ-diagonal* of A, and denoted by $\Delta_{\epsilon,A}$. We refer to $\mathrm{Gr}_{0,f}$ simply as the *graph* of f, also denoted Gr_f, and to $\Delta_{0,A}$ as the *diagonal* of A, which we continue to denote as Δ_A.

Using the notation assembled, we can rephrase the definition of pseudometric as follows.

▶ **Lemma 3.5.** *A fuzzy relation $d\colon X \nrightarrow X$ is a pseudometric iff*

$$\begin{aligned} d &\leq \Delta_X && \textit{(reflexivity)} \\ d^\circ &= d && \textit{(symmetry)} \\ d &\leq d; d && \textit{(triangle inequality).} \end{aligned}$$

We now introduce our central notion of non-expansive lax extension:

▶ **Definition 3.6** (Fuzzy lax extensions)**.** A *(fuzzy) relation lifting* L of T maps each fuzzy relation $R\colon A \nrightarrow B$ to a fuzzy relation $LR\colon TA \nrightarrow TB$ such that

(L0) $L(R^\circ) = (LR)^\circ$

for all R. We say that L is a *fuzzy lax extension* if it additionally satisfies

(L1) $R_1 \leq R_2 \Rightarrow LR_1 \leq LR_2$

(L2) $L(R;S) \leq LR;LS$

(L3) $L\mathrm{Gr}_f \leq \mathrm{Gr}_{Tf}$

for all sets A, B, and $R, R_1, R_2\colon A \nrightarrow B$, $S\colon B \nrightarrow C$, $f\colon A \to B$. A fuzzy lax extension L is *non-expansive*, and then briefly called a *non-expansive lax extension*, if

(L4) $L\Delta_{\epsilon,A} \leq \Delta_{\epsilon,TA}$

for all sets A and $\epsilon > 0$.

Axioms (L0)–(L3) are straightforward quantitative generalizations of the axiomatization of two-valued lax extensions [35]; fuzzy lax extensions in this sense have also been called $[0,1]$-*relators* [24, 27] (in the more general setting of quantale-valued relations). Axiom (L4) has no two-valued analogue; its role and the terminology are explained by the following characterization:

▶ **Lemma 3.7.** *Let L be a fuzzy lax extension of T. Then the following are equivalent.*

1. *L satisfies Axiom (L4) (i.e. is non-expansive).*
2. *For all functions $f\colon A \to B$ and all $\epsilon > 0$, $LGr_{\epsilon,f} \leq Gr_{\epsilon,Tf}$.*
3. *For all sets A, B, the map $R \mapsto LR$ is non-expansive w.r.t. the supremum metric on $A \nrightarrow B$.*

This characterization is an important prerequisite for the Hennessy-Milner theorem. Its proof relies on the following basic property [27, Corollary III.1.4.4]:

▶ **Lemma 3.8** (Naturality)**.** *Let L be a fuzzy lax extension of T, let $R\colon A' \nrightarrow B'$ be a fuzzy relation, and let $f\colon A \to A', g\colon B \to B'$. Then $L(R \circ (f \times g)) = LR \circ (Tf \times Tg)$.*

As indicated previously, existing approaches to behavioural metrics (e.g. [53, 2]) are based on lifting functors to pseudometric spaces. Every lax extension induces such a functor lifting:

▶ **Lemma 3.9.** *Let L be a fuzzy lax extension, and let $d\colon X \nrightarrow X$ be a pseudometric. Then Ld is a pseudometric on TX. Moreover, for every non-expansive map $f\colon (X, d_1) \to (Y, d_2)$ of pseudometric spaces, the map $Tf\colon (TX, Ld_1) \to (TY, Ld_2)$ is non-expansive.*

That is, every fuzzy lax extension of $T\colon \mathsf{Set} \to \mathsf{Set}$ gives rise to a functor $\overline{T}\colon \mathsf{PMet} \to \mathsf{PMet}$ on the category PMet of pseudometric spaces and nonexpansive maps that *lifts* T in the sense that $U \circ \overline{T} = T \circ U$, where $U\colon \mathsf{PMet} \to \mathsf{Set}$ is the functor that forgets the pseudometric.

Much of the development will be based on finitary functors; for instance, we need a finitary functor so we can give an explicit syntax for the characterizing logic of a lax extension. The following notion captures a broader class of functors than just the finitary ones.

▶ **Definition 3.10.** A fuzzy lax extension L for the functor T is *finitarily separable* if for every set X, $T_\omega X$ is a dense subset of TX wrt. the pseudometric $L\Delta_X$.

Clearly, any lax extension of a finitary functor is finitarily separable. The prototypical example of a finitarily separable lax extension of a non-finitary functor is the Kantorovich lifting of the discrete distribution functor $\mathcal{D}$ (Example 5.8.1). We conclude the section with a basic example of a non-expansive lax extension, deferring further examples to the sections on systematic constructions of such extensions (Sections 5 and 6):

▶ **Example 3.11** (Hausdorff lifting)**.** The *Hausdorff lifting* is the relation lifting H for the powerset functor $\mathcal{P}$, defined for fuzzy relations $R\colon A \nrightarrow B$ by

$$HR(U,V) = \max(\sup_{a\in U}\inf_{b\in V} R(a,b),\ \sup_{b\in V}\inf_{a\in U} R(a,b)).$$

for $U \subseteq A, V \subseteq B$. The Hausdorff lifting can be viewed as a quantitative analogue of the Egli-Milner extension (Section 2), where sup replaces universal quantification and inf replaces existential quantification. It is shown already in [27] that H is a fuzzy lax extension. Indeed, it is easy to see that H is also non-expansive. These properties will also follow from the results of Section 6, where we show that H is in fact an instance of the Wasserstein lifting. H is not finitarily separable, because for every set X we have $H\Delta_X = \Delta_{\mathcal{P}X}$.

4 Quantitative Bisimulations

We next identify a notion of bisimulation based on a lax extension L of the functor T; similar concepts appear in work on quantitative applicative bisimilarity [24]. We define behavioural distance based on this notion, and show coincidence with the distance defined via the pseudometric lifting induced by L according to Lemma 3.9.

▶ **Definition 4.1.** Let L be a lax extension of T, and let $\alpha\colon A \to TA$ and $\beta\colon B \to TB$ be coalgebras.

1. A fuzzy relation $R\colon A \nrightarrow B$ is an *L-bisimulation* if $LR \circ (\alpha \times \beta) \leq R$.
2. We define *L-behavioural distance* $d^L_{\alpha,\beta}\colon A \nrightarrow B$ to be the infimum of all L-bisimulations:

$$d^L_{\alpha,\beta} = \inf\{R\colon A \nrightarrow B \mid R \text{ is an } L\text{-bisimulation}\}.$$

If $\alpha = \beta$, we write $d^L_\alpha = d^L_{\alpha,\beta}$ instead.

▶ Remark 4.2. Putting Definition 4.1 in other words, an L-bisimulation is precisely a prefix point for the map $F(R) = LR \circ (\alpha \times \beta)$. Note that F is monotone by (L1). This means that, according to the Knaster-Tarski fixpoint theorem, $d^L_{\alpha,\beta}$ is itself a prefix point (i.e. an L-bisimulation), and also the least fixpoint of F, i.e. $d^L_{\alpha,\beta} = Ld^L_{\alpha,\beta} \circ (\alpha \times \beta)$.

As L-behavioural distance is the least L-bisimulation, we get:

▶ **Lemma 4.3.** *For every coalgebra $\alpha\colon A \to TA$, d^L_α is a pseudometric.*

▶ Remark 4.4. As announced above, existing generic notions of behavioural distance defined via functor liftings [2] agree with the one given above (when both apply). Specifically, when applied to the functor lifting induced by a lax extension L of T according to Lemma 3.9, the definition of behavioural distance via functor liftings amounts to taking the same least fixpoint as in Definition 4.1 but only over pseudometrics instead of over fuzzy relations.

▶ Remark 4.5. Every fuzzy lax extension L induces a crisp lax extension L_c, where for any crisp relation R, $L_cR = (LR)^{-1}[\{0\}] \subseteq TA \times TB$ (recall Convention 3.2). It is easily checked that L_c preserves diagonals (Section 2) iff

$$L\Delta_A \text{ is a metric for each set } A. \tag{2}$$

By results on lax extensions cited in Section 2, L_c-bisimilarity coincides with behavioural equivalence in this case, i.e. if L satisfies (2), then L characterizes behavioural equivalence: Two states $a \in A$, $b \in B$ in coalgebras (A,α), (B,β) are behaviourally equivalent iff $d^L_{\alpha,\beta}(a,b) = 0$.

▶ **Example 4.6** (Small bisimulations)**.** We give an example for the functor $TX = [0,1] \times \mathcal{P}X$. Coalgebras for T are Kripke frames where each state is labelled with a number from the unit interval. This T has a non-expansive lax extension L, defined for fuzzy relations $R\colon A \nrightarrow B$ by

$$LR((p,U),(q,V)) = \tfrac{1}{2}(|p-q| + HR(U,V)),$$

where $p,q \in [0,1], U \subseteq A, V \subseteq B$, and H is the Hausdorff lifting (Example 3.11). The motivating idea behind this definition is that the L-behavioural distance of two states is the supremum of the accumulated branching-time differences between state labels over all runs of a process starting at these states. The factor $\frac{1}{2}$ ensures that the total distance is at most 1 by discounting the differences at later stages with exponentially decreasing factors.

Now consider the T-coalgebras (A,α) and (B,β) below:

We put $R(a_1,b_1) = 0.2, R(a_2,b_3) = 0.1, R(a_3,b_2) = 0.05$ and $R(a_i,b_j) = 1$ in all other cases. Then R is an L-bisimulation witnessing that $d^L_{\alpha,\beta}(a_1,b_1) \leq 0.2$, but is neither reflexive nor symmetric, nor transitive on the disjoint union of the systems.

As indicated previously, quantitative Hennessy-Milner theorems can only be expected to hold for non-expansive lax extensions. The key observation is the following. By standard fixpoint theory, L-behavioural distance can be approximated from below by an ordinal-indexed increasing chain. Crucially, if L is non-expansive and finitarily separable, then this chain stabilizes after ω steps. Formally:

▶ **Theorem 4.7.** *Let L be a non-expansive finitarily separable lax extension of T. Given T-coalgebras (A,α), (B,β), define a sequence $(d_n\colon A \nrightarrow B)_{n<\omega}$ and $d_\omega\colon A \nrightarrow B$ by*

$$d_0 = 0, \qquad d_{n+1} = Ld_n \circ (\alpha \times \beta), \qquad d_\omega = \sup_{n<\omega} d_n.$$

Then

(i) *$Ld_\omega \circ (\alpha \times \beta) = d_\omega$, and*

(ii) *L-behavioural distance $d^L_{\alpha,\beta}$ equals d_ω.*

Proof (sketch). In case T is finitary, we can exploit the fact that under restriction to a finite subset of $A \times B$ the pointwise convergence of $(d_n)_{n<\omega}$ becomes uniform, so (i) follows from nonexpansivity of L using Lemma 3.7.3. To generalize to the non-finitary case, one can use an *unravelling* construction and approximate the unravelled T-coalgebra by a T_ω-coalgebra such that the series of accumulated errors converges to a fixed ϵ. Claim (ii) is immediate from (i) by the fixpoint definition of $d^L_{\alpha,\beta}$. ◀

5 The Kantorovich Lifting

As a pseudometric lifting, the Kantorovich lifting is standard in the probabilistic setting: Given a metric d on a set X, the Kantorovich distance $Kd(\mu_1,\mu_2)$ between discrete distributions μ_1,μ_2 on X is defined by

$$Kd(\mu_1,\mu_2) = \sup\{\mathbb{E}_{\mu_1}(f) - \mathbb{E}_{\mu_2}(f) \mid f\colon (X,d) \to ([0,1],d_E) \text{ nonexpansive}\}$$

where $\mathbb{E}$ takes expected values. The coalgebraic generalization of the Kantorovich lifting, both in the pseudometric setting [31] and in the present setting of fuzzy relations, is based on fuzzy predicate liftings, a quantitative analogue of two-valued predicate liftings (Section 2) that goes back to work on coalgebraic fuzzy description logics [48]. Fuzzy predicate liftings will feature in the generic quantitative modal logics that we extract from fuzzy lax extensions (Section 8).

Recall that the *contravariant fuzzy powerset functor* $\mathcal{Q}\colon \mathsf{Set} \to \mathsf{Set}$ is defined on sets X as $\mathcal{Q}X = (X \to [0,1])$ and on functions $f\colon X \to Y$ as $\mathcal{Q}f(h) = h \circ f$.

▶ **Definition 5.1** (Fuzzy predicate liftings)**.** Let $n \in \mathbb{N}$.

1. An *n-ary (fuzzy) predicate lifting* is a natural transformation

$$\lambda\colon \mathcal{Q}^n \Rightarrow \mathcal{Q} \circ T,$$

   where the exponent n denotes n-fold cartesian product.
2. The *dual* of λ is the n-ary predicate lifting $\bar{\lambda}$ given by $\bar{\lambda}(f_1,\dots,f_n) = 1-\lambda(1-f_1,\dots,1-f_n)$.
3. We call λ *monotone* if for all sets X and all functions $f_1,\dots,f_n,g_1,\dots,g_n \in \mathcal{Q}X$ such that $f_i \le g_i$ for all i, $\lambda_X(f_1,\dots,f_n) \le \lambda_X(g_1,\dots,g_n)$.
4. We call λ *nonexpansive* if for all sets X and all functions $f_1,\dots,f_n,g_1,\dots,g_n \in \mathcal{Q}X$,

$$\|\lambda_X(f_1,\dots,f_n) - \lambda_X(g_1,\dots,g_n)\|_\infty \le \max(\|f_1-g_1\|_\infty,\dots,\|f_n-g_n\|_\infty).$$

▶ Remark 5.2. By the Yoneda lemma, unary predicate liftings are equivalent to the *evaluation functions* $e\colon T[0,1] \to [0,1]$ used in work on pseudometric functor liftings [2, 47] and on the generic Wasserstein lifting [26]; more generally, an n-ary predicate lifting is equivalent to a generalized form of evaluation function, of type $T[0,1]^n \to [0,1]$ [47].

Before we can prove that the Kantorovich lifting is a lax extension, we first need to generalize it so that it lifts arbitrary fuzzy relations instead of just pseudometrics. To this end, we introduce the notion of nonexpansive pairs (a similar idea appears already in [54, Section 5]):

▶ **Definition 5.3.** Let $R\colon A \nrightarrow B$. A pair (f,g) of functions $f\colon A \to [0,1]$ and $g\colon B \to [0,1]$ is *R-nonexpansive* if $f(a) - g(b) \le R(a,b)$ for all $a \in A, b \in B$.

Given a function and a fuzzy relation, we can construct a *nonexpansive companion*:

▶ **Definition 5.4.** Let $R\colon A \nrightarrow B$ and $f\colon A \to [0,1]$. Then we define $R[f]\colon B \to [0,1]$ by

$$R[f](b) = \sup_{a\in A} f(a) \ominus R(a,b),$$

where for $x,y \in [0,1]$, $x \ominus y = \max(x-y,0)$.

▶ **Definition 5.5.** Let Λ be a set of monotone predicate liftings that is closed under duals. The *Kantorovich lifting* K_Λ is defined as follows: for $R\colon A \nrightarrow B$, $K_\Lambda R\colon TA \nrightarrow TB$ is given by

$$K_\Lambda R(t_1,t_2) = \sup\{\lambda_A(f_1,\dots,f_n)(t_1) - \lambda_B(g_1,\dots,g_n)(t_2) \mid \lambda \in \Lambda\ n\text{-ary}, (f_1,g_1),\dots(f_n,g_n)\ R\text{-nonexpansive}\}.$$

We show in the appendix that closure under duals guarantees that $K_\Lambda R(t_1,t_2) \ge 0$ always.

▶ **Theorem 5.6.** *Let Λ be a set of monotone predicate liftings that is closed under duals. The Kantorovich lifting K_Λ is a lax extension. If all $\lambda \in \Lambda$ are nonexpansive, then K_Λ is nonexpansive as well.*

Proof (sketch). We sketch the proofs for (L2) and (L4). For (L2), one observes that given a nonexpansive pair (f,h) for $R;S$, one can obtain nonexpansive pairs (f,g) for R and (g,h) for S using the nonexpansive companion $g = R[f]$. For (L4), we note that $\Delta_{\epsilon,A}$-nonexpansivity of (f,g) implies that $f(a) - g(a) \leq \epsilon$ for all $a \in A$. By monotonicity of predicate liftings, we can assume that w.l.o.g. $g(a) = f(a) \ominus \epsilon$. In this case, for every $\lambda \in \Lambda$ (for simplicity, unary) and $t \in TA$,

$$\lambda(f)(t) - \lambda(g)(t) \leq \|\lambda(f) - \lambda(g)\|_\infty \leq \|f - g\|_\infty \leq \epsilon. \qquad \blacktriangleleft$$

▶ Remark 5.7 (Kantorovich for pseudometrics). On pseudometrics, the Kantorovich lifting K_Λ as given by Definition 5.5 agrees with the usual Kantorovich distance $-^{\uparrow T}$ [2, Definition 5.4] defined for pseudometrics. If $d\colon A \nrightarrow A$ is a pseudometric, then $d^{\uparrow T}(t_1,t_2)$ equals

$$\sup\{|\lambda_A(f_1,\dots,f_n)(t_1) - \lambda_A(f_1,\dots,f_n)(t_2)| \mid \lambda \in \Lambda, f_1,\dots,f_n\colon (A,d) \to_1 ([0,1],d_E)\}$$

▶ **Example 5.8** (Kantorovich liftings)**.**

1. The standard Kantorovich lifting K of the discrete distribution functor $\mathcal{D}$ is an instance of the generic one, for the single predicate lifting $\Diamond(f)(\mu) = \mathbb{E}_\mu(f)$. Crucially, K is finitarily separable, by the observation that for every discrete distribution $\mu \in \mathcal{D}X$ and every $\epsilon > 0$, there are only finitely many points x with $\mu(x) > \epsilon$.
2. The *fuzzy neighbourhood functor* is the (covariant) functor $\mathcal{N} = \mathcal{Q} \circ \mathcal{Q}$; the elements of $\mathcal{N}X$ are called *fuzzy neighbourhood systems*, and their coalgebras *fuzzy neighbourhood frames* [45, 10]. The *monotone (nonexpansive) fuzzy neighbourhood functor* $\mathcal{M}$ is the subfunctor $\mathcal{M}$ of $\mathcal{N}$ given by $\mathcal{M}X$ consisting of the fuzzy neighbourhood systems that are monotone and nonexpansive as maps $A\colon \mathcal{Q}X \to [0,1]$. We put

   $$LR(A,B) = \max(\sup_{f \in \mathcal{Q}X} A(f) \ominus B(R[f]), \sup_{g \in \mathcal{Q}X} B(g) \ominus A(R^\circ[g]))$$

   for $R\colon A \nrightarrow B$, $A \in \mathcal{M}X$, $B \in \mathcal{M}Y$ (recall Definition 5.4). Then L is a nonexpansive lax extension of $\mathcal{M}$; specifically, $L = K_{\{\lambda\}}$ where λ is the predicate lifting given by $\lambda_X(f)(A) = A(f)$.

6 The Wasserstein Lifting

The other generic construction for lax extensions arises in a similar way, by generalizing the generic Wasserstein lifting for pseudometrics [2] to lift arbitrary fuzzy relations instead of just pseudometrics; our construction slightly generalizes one given by Hofmann [26]. Compared to the case of the Kantorovich lifting, where we needed to work with nonexpansive pairs, the generalization from pseudometric lifting to relation lifting is much more direct. In the same way as for the original construction of pseudometric Wasserstein liftings, additional constraints, both on the functor and the set of predicate liftings involved, are needed for the Wasserstein lifting to be a lax extension. Indeed, the Wasserstein lifting may be seen as a quantitative analogue of the two-valued Barr extension (Section 2), and like the latter works only for functors that preserve weak pullbacks. In particular, Wasserstein liftings are based on the central notion of coupling:

▶ **Definition 6.1.** Let $t_1 \in TA, t_2 \in TB$ for sets A, B. The set of *couplings* of t_1 and t_2 is $\mathsf{Cpl}(t_1,t_2) = \{t \in T(A \times B) \mid T\pi_1(t) = t_1, T\pi_2(t) = t_2\}$.

Like the Kantorovich lifting, the Wasserstein lifting is based on a choice of predicate liftings. It is, however, built in a quite different manner, and in particular appears to make sense only for unary predicate liftings, so unlike in some other places in the paper, the restriction to unary liftings in the next definition is not just for readability.

▶ **Definition 6.2** (Wasserstein lifting)**.** Let Λ be a set of unary predicate liftings. The *generic Wasserstein lifting* is the relation lifting W_Λ of T defined for $R\colon A \nrightarrow B$ by

$$W_\Lambda R(t_1, t_2) = \sup_{\lambda\in\Lambda} \inf\{\lambda_{A\times B}(R)(t) \mid t \in \mathsf{Cpl}(t_1, t_2)\}.$$

This construction is similar to [26, Definition 3.4] except that we admit more than one modality. On pseudometrics, the Wasserstein lifting coincides with the pseudometric lifting $-^{\downarrow T}$ as defined in [2, Definition 5.12]. We will see that the following conditions ensure that the Wasserstein lifting is a fuzzy lax extension:

▶ **Definition 6.3.** Let λ be a unary predicate lifting.

1. λ is *subadditive* if for all sets X and all $f, g \in \mathcal{Q}X$, $\lambda_X(f \oplus g) \leq \lambda_X(f) \oplus \lambda_X(g)$.
2. λ *preserves the zero function* if for all sets X, $\lambda_X(0_X) = 0_{TX}$, where $0_X\colon x \mapsto 0$.
3. λ is *standard* if it is monotone, subadditive, and preserves the zero function.

▶ Remark 6.4. Baldan et al. give conditions under which the Wasserstein lifting arising from some set of evaluation functions (Remark 5.2) preserves pseudometrics. For this purpose they consider the notion of a *well-behaved evaluation function* [2, Definition 5.14]. We show in Appendix A that this amounts to a slightly stronger condition than standardness of the corresponding predicate lifting. Similar conditions also feature in Hofmann's *topological theories* [26, Definition 3.1], which consist of a monad acting on a quantale via an evaluation function and on which his generic Wasserstein extension is based. We show in the appendix that, ignoring some monad-specific axioms, the conditions imposed on the functor and evaluation function are equivalent to standardness of the associated predicate lifting.
Now indeed we have

▶ **Theorem 6.5.** *If T preserves weak pullbacks and Λ is a set of standard predicate liftings, then the Wasserstein lifting W_Λ is a lax extension. If additionally all $\lambda \in \Lambda$ are nonexpansive, then W_Λ is nonexpansive as well.*

Proof (sketch). The proofs of (L0)–(L3) are similar to [26, Theorem 3.5]. In particular, (L3) follows by preservation of the zero function, and(L2) is based on subadditivity of predicate liftings and weak pullback preservation of T. The latter is a prerequisite for the so-called gluing lemma (e.g. [2, Lemma 5.18]), which gives a canonical way of producing couplings $t_{13} \in \mathsf{Cpl}(t_1, t_3)$ from couplings $t_{12} \in \mathsf{Cpl}(t_1, t_2)$ and $t_{23} \in \mathsf{Cpl}(t_2, t_3)$. The proof of (L4) is by nonexpansivity of predicate liftings. ◀

▶ **Example 6.6** (Wasserstein liftings)**.**

1. The Hausdorff lifting H (Example 3.11) is the Wasserstein lifting $W_{\{\lambda\}}$ for $\mathcal{P}$, where $\lambda_X(f)(A) = \sup f[A]$ for $A \subseteq X$.
2. The convex powerset functor $\mathcal{C}$, whose coalgebras combine probabilistic branching and nondeterminism [6], maps a set X to the set of nonempty convex subsets of $\mathcal{D}X$. The Wasserstein lifting $W_{\{\lambda\}}$, where $\lambda_X(f)(A) = \sup_{\mu\in A} \mathbb{E}_\mu(f)$ for $A \in \mathcal{C}X$, is a non-expansive lax extension of $\mathcal{C}$. Of course, λ is just the composite of the predicate liftings respectively defining the standard Kantorovich and Hausdorff liftings. As we show in the appendix, $W_{\{\lambda\}}$ indeed coincides with the composite of these liftings (for which a quantitative equational axiomatization has recently been given by Mio and Vignudelli [39]).

7 Lax Extensions as Kantorovich Liftings

We proceed to establish the central result that every fuzzy lax extension is a Kantorovich lifting for some suitable set Λ of predicate liftings, and moreover we characterize the Kantorovich liftings induced by non-expansive predicate liftings as precisely the non-expansive lax extensions. For a given fuzzy lax extension L, the equality $K_\Lambda R = LR$ splits into two inequalities, one of which is characterized straightforwardly:

▶ **Definition 7.1.** An n-ary predicate lifting λ *preserves nonexpansivity* if for all fuzzy relations R and all R-nonexpansive pairs $(f_1, g_1), \dots, (f_n, g_n)$, the pair $(\lambda_A(f_1, \dots, f_n), \lambda_B(g_1, \dots, g_n))$ is LR-nonexpansive. A set Λ of predicate liftings *preserves nonexpansivity* if all $\lambda \in \Lambda$ preserve nonexpansivity.

▶ **Lemma 7.2.** *We have $K_\Lambda R \le LR$ if and only if Λ preserves nonexpansivity.*

▶ **Definition 7.3** (Separation)**.** A set Λ of predicate liftings is *separating* for L if $K_\Lambda R \ge LR$ for all fuzzy relations R.

To motivate Definition 7.3, recall from Section 2 that in the two-valued setting a set Λ of predicate liftings (for simplicity, assumed to be unary) is separating if

$$t_1 \neq t_2 \implies \exists \lambda \in \Lambda, A' \subseteq A \text{ such that } t_1 \in \lambda_A(A') \not\Leftrightarrow t_2 \in \lambda_A(A')$$

for $t_1, t_2 \in TA$. Analogously, unfolding definitions in the inequality $K_\Lambda R \ge LR$ (and again assuming unary liftings), we arrive at the condition that for all $t_1 \in TA, t_2 \in TB$, $\epsilon > 0$,

$$LR(t_1, t_2) > \epsilon \implies \exists \lambda \in \Lambda, (f, g) \ R\text{-nonexpansive such that } \lambda_A(f)(t_1) - \lambda_B(g)(t_2) > \epsilon.$$

We are now ready to state our main result, which says that all lax extensions are Kantorovich:

▶ **Theorem 7.4.** *If L is a finitarily separable lax extension of T, then there exists a set Λ of monotone predicate liftings that preserves nonexpansivity and is separating for L, i.e. $L = K_\Lambda$. Moreover, L is nonexpansive iff Λ can be chosen in such a way that all $\lambda \in \Lambda$ are nonexpansive.*

This result can be seen as a fuzzy version of the statements that every finitary functor has a separating set of two-valued modalities (and hence an expressive two-valued coalgebraic modal logic) [47, Corollary 45], and that more specifically, every finitary functor equipped with a diagonal-preserving lax extension has a separating set of two-valued *monotone* predicate liftings [35, Theorem 14]. We will detail in Section 8 how Theorem 7.4 implies the existence of characteristic modal logics. The proof of Theorem 7.4 uses a quantitative version of the so-called Moss modalities [32, 35]. The construction of these modalities relies on the fact that T_ω can be presented by algebraic operations of finite arity:

▶ **Definition 7.5.** A *finitary presentation* of T_ω consists of a *signature* Σ of operations with given finite arities, and for each $\sigma \in \Sigma$ of arity n a natural transformation $\sigma\colon (-)^n \Rightarrow T_\omega$ such that every element of $T_\omega X$ has the form $\sigma_X(x_1, \dots, x_n)$ for some $\sigma \in \Sigma$.

For the remainder of this section, we fix a finitary presentation of T_ω (such a presentation always exists) and assume a finitarily separable fuzzy lax extension L of T.

▶ **Definition 7.6.** Let $\sigma \in \Sigma$ be n-ary. The *Moss lifting* $\mu^\sigma\colon \mathcal{Q}^n \Rightarrow \mathcal{Q} \circ T$ is defined by

$$\mu^\sigma_X(f_1, \dots, f_n)(t) = L\mathsf{ev}_X(\sigma_{\mathcal{Q}X}(f_1, \dots, f_n), t),$$

where $\mathsf{ev}_X\colon \mathcal{Q}X \nrightarrow X$ is given by $\mathsf{ev}_X(f, x) = f(x)$.

We take Λ to be the set of all Moss liftings and their duals:

$$\Lambda = \{\mu^\sigma \mid \sigma \in \Sigma\} \cup \{\overline{\mu^\sigma} \mid \sigma \in \Sigma\}$$

Now Theorem 7.4 is immediate from

▶ **Lemma 7.7.** *Λ is a set of monotone predicate liftings that preserves nonexpansivity and is separating for L. If L is nonexpansive, then so are all predicate liftings in Λ.*

Proof (sketch).

- *Λ is separating*: Let $s\colon B \to \mathcal{Q}A, s(b)(a) = R(a,b)$ and $\epsilon > 0$. Because the set of Σ-terms over $\mathcal{Q}A$ generates $T_\omega \mathcal{Q}A$ and L is finitarily separable, there exists some $\sigma \in \Sigma$ (for simplicity unary) and some $f \in \mathcal{Q}A$ such that $L\Delta_{\mathcal{Q}A}(\sigma(f), Ts(t_2)) \le \epsilon$. If we put $g = R[f]$, then $|\mu^\sigma(f)(t_1) - LR(t_1,t_2)| \le \epsilon$ and $\mu^\sigma(g)(t_2) \le \epsilon$. Then let $\epsilon \to 0$.
- *Nonexpansivity of Moss liftings*: This is based on the observation that for any set A and any $f, g \in \mathcal{Q}A$, $\|f - g\|_\infty \le \epsilon$ iff both (f,g) and (g,f) are $\Delta_{\epsilon,A}$-nonexpansive pairs.

For the remaining properties we refer to the full version of this paper [56]. ◀

8 Real-valued Coalgebraic Modal Logic

We next recall the generic framework of *real-valued coalgebraic modal logic*, which lifts two-valued coalgebraic modal logic (Section 2) to the quantitative setting, and will yield characteristic quantitative modal logics for all non-expansive lax extensions. The framework goes back to work on fuzzy description logics [48]. The present version, characterized by a specific choice of propositional operators, appears in work on the coalgebraic quantitative Hennessy-Milner theorem [31], and generalizes quantitative probabilistic modal logic [53].

Given a set Λ of (fuzzy) predicate liftings, the set $\mathcal{L}_\Lambda$ of modal (Λ)-formulae is given by

$$\phi, \psi ::= c \mid \phi \ominus c \mid \neg\phi \mid \phi \wedge \psi \mid \lambda(\phi_1, \dots, \phi_n) \tag{3}$$

where $c \in \mathbb{Q} \cap [0,1]$ and $\lambda \in \Lambda$ has arity n. The semantics assigns to each formula ϕ and each coalgebra (A, α) a real-valued map $[\![\phi]\!]_{A,\alpha}\colon A \to [0,1]$, or just $[\![\phi]\!]$, defined by

$$[\![c]\!](a) = c \qquad [\![\phi \wedge \psi]\!](a) = \min([\![\phi]\!](a), [\![\psi]\!](a))$$
$$[\![\phi \ominus c]\!](a) = \max([\![\phi]\!](a) - c, 0) \qquad [\![\lambda(\phi_1, \dots, \phi_n)]\!](a) = \lambda_A([\![\phi_1]\!], \dots, [\![\phi_n]\!])(\alpha(a))$$
$$[\![\neg\phi]\!](a) = 1 - [\![\phi]\!](a)$$

▶ Remark 8.1. We thus adopt what is often called *Zadeh semantics* for the propositional operators. This choice is pervasive in characteristic logics for behavioural distances (including [53, 31, 57]) – in particular, the more general *Łukasiewiecz semantics* fails to be nonexpansive w.r.t. behavioural distance, and indeed induces a discrete logical distance [57].

In the two-valued setting, one can sometimes restrict the propositional base in characteristic logics; notably, two-valued probabilistic modal logic characterizes (event) bisimilarity of probabilistic transition systems even with conjunction as the only propositional connective [15]. No similar results appear to be known in the quantitative case; e.g. van Breugel and Worrell's characteristic logic for behavioural distance of probabilistic transition systems [53] does feature essentially the same propositional operators as our grammar (3).

▶ **Example 8.2.**

1. *Fuzzy modal logic* may be seen as a basic fuzzy description logic [34]. Eliding propositional atoms for brevity (they may be added as nullary modalities), we take $\Lambda = \{\Diamond\}$. Models are fuzzy relational structures, i.e. coalgebras for the *covariant* fuzzy powerset functor $\mathcal{F}$ given

by $\mathcal{F}X = [0,1]^X$ and $\mathcal{F}f(g)(y) = \sup_{f(x)=y} g(x)$, and $\Diamond$ is interpreted as the predicate lifting $\Diamond_A(f)(g) = \sup_{a\in A} \min(g(a), f(a))$. Hennessy-Milner-type results necessarily apply only to finitely branching models, i.e. coalgebras for $\mathcal{F}_\omega$.

2. *Probabilistic modal logic:* Take models to be probabilistic transition systems with possible deadlocks, i.e. coalgebras for the functor $1+\mathcal{D}$, where $\mathcal{D}A$ is the set of discrete probability distributions on A (Section 2); and $\Lambda = \{\Diamond\}$, with

$$\Diamond_A(f)(*) = 0 \quad \text{for } * \in 1, \text{ and} \qquad \Diamond_A(f)(\mu) = \mathbb{E}_\mu(f) = \textstyle\sum_{a\in A} \mu(a)\cdot f(a).$$

When extended with propositional atoms, this induces (up to restricting to discrete probabilities) van Breugel et al.'s contraction-free quantitative probabilistic modal logic [52].

In the two-valued setting, modal logic is typically invariant under bisimulation, i.e. bisimilar states satisfy the same modal formulae. In the quantitative setting, this corresponds to non-expansiveness of formula evaluation, which may be phrased as saying that logical distance is below behavioural distance:

▶ **Definition 8.3.** The Λ-*logical distance* between states $a \in A$, $b \in B$ in T-coalgebras (A,α), (B,β) is $d^\Lambda(a,b) = \sup\{|[\![\phi]\!](a) - [\![\phi]\!](b)| \mid \phi \in \mathcal{L}_\Lambda\}$.

▶ **Lemma 8.4** (Non-expansiveness of quantitative modal logic)**.** *If Λ preserves non-expansiveness w.r.t. a lax extension L, then $d^\Lambda \le d^L$.*

Finally, we show how the characterization of lax extensions as Kantorovich extensions can be used to define characteristic logics for nonexpansive lax extensions. We use a Hennessy-Milner result by König and Mika-Michalski [31], for the (pseudometric) Kantorovich lifting:

▶ **Theorem 8.5.** *Let Λ be a set of predicate liftings such that iterative approximation of the fixpoint d^{K_Λ} as in Theorem 4.7 stabilizes in ω steps. Then $d^\Lambda \ge d^{K_\Lambda}$.*

We combine this result with our Theorems 4.7 and 5.6 to obtain, complementing Lemma 8.4, a criterion phrased directly in terms of conditions on the lax extension and the modalities:

▶ **Corollary 8.6** (Coalgebraic quantitative Hennessy-Milner theorem)**.** *Let L be a finitarily separable fuzzy lax extension, and let Λ be a separating set of monotone non-expansive predicate liftings for L. Then $d^\Lambda \ge d^L$.*

▶ **Example 8.7.** Since we only require L to be finitarily separable (rather than T finitary), Example 5.8.1 implies that we recover expressiveness [52, 53] of quantitative probabilistic modal logic over countably branching discrete probabilistic transition systems (Example 8.2.2) as an instance of Corollary 8.6.

▶ Remark 8.8. In [31], Theorem 8.5 is in fact only shown for the case of distances d^L_α defined on a single coalgebra. The general case of distances $d^L_{\alpha,\beta}$ between two possibly distinct coalgebras can be recovered by working on their coproduct (disjoint union), using that both L-behavioural distance and formula evaluation are preserved under morphisms.

Applying Lemma 8.4 and Corollary 8.6 to $L = K_\Lambda$ and using our result that all lax extensions are Kantorovich extensions for their Moss liftings (Theorem 7.4), which moreover are monotone and nonexpansive in case L is nonexpansive (Lemma 7.7), we obtain expressive logics for finitarily separable nonexpansive lax extensions:

▶ **Corollary 8.9.** *If L is a finitarily separable nonexpansive lax extension of a functor T, then $d^L = d^\Lambda$ for the set Λ of Moss liftings.*

We can see the coalgebraic modal logic of Moss liftings as concrete syntax for a more abstract logic where we incorporate functor elements into the syntax directly, as in Moss' coalgebraic logic [42]. The set $\mathcal{L}_L$ of formulae in the arising *quantitative Moss logic* is generated by the same propositional operators as above, and additionally by a modality Δ that applies to $\Phi \in T\mathcal{L}_0$ for finite $\mathcal{L}_0 \subseteq \mathcal{L}_L$, with semantics

$$[\![\Delta\Phi]\!](a) = L\mathsf{ev}_A(\Phi, \alpha(a)).$$

The dual of Δ is denoted ∇, and behaves like a quantitative analogue of Moss' two-valued ∇. From Corollary 8.9, it is immediate that this logic is expressive:

▶ **Corollary 8.10** (Expressiveness of quantitative Moss logic). *Let L be a finitarily separable nonexpansive lax extension of a functor T. Then L-behavioural distance d^L coincides with logical distance in quantitative Moss logic, i.e. for all states $a \in A$, $b \in B$ in $(A, \alpha), (B, \beta)$ of coalgebras, and all $a \in A, b \in B$, $d^L_{\alpha,\beta}(a, b) = \sup\{|[\![\phi]\!](a) - [\![\phi]\!](b)| \mid \phi \in \mathcal{L}_L\}$.*

▶ **Example 8.11.**

1. We equip the finite fuzzy powerset functor $\mathcal{F}_\omega$ with the Wasserstein lifting $W_\Diamond$ for $\Diamond$ as in Example 8.2.1, in analogy to the Hausdorff lifting (Example 6.6.1). Then ∇ applies to finite fuzzy sets Φ of formulae, and

   $$[\![\nabla\Phi]\!](a) = \sup_{t \in \mathsf{Cpl}(\Phi, \alpha(a))} \inf_{(\phi, a') \in \mathcal{L}_L \times A} \max(1 - t(\phi, a'), \phi(a'))$$

   for a state a in an $\mathcal{F}$-coalgebra (A, α), i.e. in a finitely branching fuzzy relational structure.
2. Let C_{fg} be the subfunctor of the convex powerset functor $\mathcal{C}$ given by the finitely generated convex sets of (not necessarily finite) discrete distributions, equipped with the Wasserstein lifting described in Example 6.6.2. Then ∇ applies to finite sets of finite distributions on formulae, understood as spanning a convex polytope. By Corollary 8.10, the arising instance of quantitative Moss logic is expressive for all C_{fg}-coalgebras.

9 Conclusions

We have developed a systematic theory of behavioural distances based on fuzzy lax extensions, identifying the key notion of *non-expansive lax extension*, which we believe has good claims to being the right notion of quantitative relation lifting in this context. We give two general constructions of non-expansive lax extensions, respectively generalizing the classical Kantorovich and Wasserstein distances and strengthening previous generalizations where only pseudometrics are lifted [2]. Our construction of the Kantorovich lifting is based in particular on the key notion of non-expansive pair (implicit in recent work on optimal transportation [54]). Our main result shows that every non-expansive lax extension is a Kantorovich lifting for a suitable choice of modalities, the so-called Moss modalities. Moreover, one can extract from a given non-expansive lax extension a characteristic modal logic satisfying a strong form of quantitative Hennessy-Milner property. Future work will concern the extension of the systematic study of behavioural distances beyond branching-time distances as exemplified in previous work on the concrete case of metric transition systems [20], possibly using a quantitative variant of graded monads [36]; as well as a further generalization to quantale-valued metrics.

References

1 Jiří Adámek, Horst Herrlich, and George Strecker. *Abstract and Concrete Categories*. Wiley Interscience, 1990. Available as *Reprints Theory Appl. Cat.* 17 (2006), pp. 1-507.

2 Paolo Baldan, Filippo Bonchi, Henning Kerstan, and Barbara König. Coalgebraic Behavioral Metrics. *Logical Methods in Computer Science*, Volume 14, Issue 3, September 2018. doi:10.23638/LMCS-14(3:20)2018.

3 Borja Balle, Pascale Gourdeau, and Prakash Panangaden. Bisimulation metrics for weighted automata. In *International Colloquium on Automata, Languages, and Programming, ICALP 2017*, volume 80 of *LIPIcs*, pages 103:1–103:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.103.

4 Michael Barr. Relational algebras. In *Proc. Midwest Category Seminar*, volume 137 of *LNM*. Springer, 1970.

5 Michael Barr. Terminal coalgebras in well-founded set theory. *Theoret. Comput. Sci.*, 114:299–315, 1993.

6 Filippo Bonchi, Alexandra Silva, and Ana Sokolova. The power of convex algebras. In *Concurrency Theory, CONCUR 2017*, volume 85 of *LIPIcs*, pages 23:1–23:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CONCUR.2017.23.

7 Yongzhi Cao, Sherry Sun, Huaiqing Wang, and Guoqing Chen. A behavioral distance for fuzzy-transition systems. *IEEE Trans. Fuzzy Systems*, 21(4):735–747, 2013. doi:10.1109/TFUZZ.2012.2230177.

8 Valentina Castiglioni, Daniel Gebler, and Simone Tini. Logical characterization of bisimulation metrics. In *Quantitative Aspects of Programming Languages and Systems, QAPL 2016*, volume 227 of *EPTCS*, pages 44–62, 2016. doi:10.4204/EPTCS.227.

9 Konstantinos Chatzikokolakis, Daniel Gebler, Catuscia Palamidessi, and Lili Xu. Generalized bisimulation metrics. In *Concurrency Theory, CONCUR 2014*, volume 8704 of *LNCS*, pages 32–46. Springer, 2014. doi:10.1007/978-3-662-44584-6.

10 Petr Cintula, Carles Noguera, and Jonas Rogger. From Kripke to neighborhood semantics for modal fuzzy logics. In *Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU 2016*, volume 611 of *CCIS*, pages 95–107. Springer, 2016. doi:10.1007/978-3-319-40581-0_9.

11 Corina Cîrstea, Alexander Kurz, Dirk Pattinson, Lutz Schröder, and Yde Venema. Modal logics are coalgebraic. *Comput. J.*, 54:31–41, 2011. doi:10.1093/comjnl/bxp004.

12 Luca de Alfaro, Marco Faella, and Mariëlle Stoelinga. Linear and branching system metrics. *IEEE Trans. Software Eng.*, 35(2):258–273, 2009. doi:10.1109/TSE.2008.106.

13 Yuxin Deng, Tom Chothia, Catuscia Palamidessi, and Jun Pang. Metrics for action-labelled quantitative transition systems. In *Quantitative Aspects of Programming Languages, QAPL 2005*, volume 153 of *ENTCS*, pages 79–96. Elsevier, 2006. doi:10.1016/j.entcs.2005.10.033.

14 Josée Desharnais. *Labelled Markov processes*. PhD thesis, McGill University, Montreal, November 1999.

15 Josee Desharnais, Abbas Edalat, and Prakash Panangaden. A logical characterization of bisimulation for labeled Markov processes. In *Logic in Computer Science, LICS 1998*, pages 478–487. IEEE Computer Society, 1998.

16 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled Markov processes. *Theor. Comput. Sci.*, 318:323–354, 2004.

17 Wenjie Du, Yuxin Deng, and Daniel Gebler. Behavioural pseudometrics for nondeterministic probabilistic systems. In *Dependable Software Engineering: Theories, Tools, and Applications, SETTA 2016*, volume 9984 of *LNCS*, pages 67–84. Springer, 2016. doi:10.1007/978-3-319-47677-3.

18 Pantelis Eleftheriou, Costas Koutras, and Christos Nomikos. Notions of bisimulation for Heyting-valued modal languages. *J. Log. Comput.*, 22(2):213–235, 2012.

19 Uli Fahrenberg and Axel Legay. The quantitative linear-time-branching-time spectrum. *Theor. Comput. Sci.*, 538:54–69, 2014. doi:10.1016/j.tcs.2013.07.030.

20 Uli Fahrenberg, Axel Legay, and Claus Thrane. The quantitative linear-time–branching-time spectrum. In *Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2011*, volume 13 of *LIPIcs*, pages 103–114. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2011. doi:10.4230/LIPIcs.FSTTCS.2011.103.

21 T. Fan. Fuzzy bisimulation for Gödel modal logic. *IEEE Trans. Fuzzy Sys.*, 23:2387–2396, December 2015. doi:10.1109/TFUZZ.2015.2426724.

22 Norm Ferns, Prakash Panangaden, and Doina Precup. Metrics for finite Markov decision processes. In *Uncertainty in Artificial Intelligence, UAI 2004*, pages 162–169. AUAI Press, 2004.

23 Melvin Fitting. Many-valued modal logics. *Fundam. Inform.*, 15:235–254, 1991.

24 Francesco Gavazzo. Quantitative behavioural reasoning for higher-order effectful programs: Applicative distances. In *Logic in Computer Science, LICS 2018*, pages 452–461. ACM, 2018. doi:10.1145/3209108.

25 Alessandro Giacalone, Chi-Chang Jou, and Scott Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Programming concepts and methods, PCM 1990*, pages 443–458. North-Holland, 1990.

26 Dirk Hofmann. Topological theories and closed objects. *Adv. Math.*, 215(2):789–824, 2007. doi:10.1016/j.aim.2007.04.013.

27 Dirk Hofmann, Gavin Seal, and Walter Tholen, editors. *Monoidal Topology: A Categorical Approach to Order, Metric, and Topology*. Cambridge University Press, 2014.

28 Jesse Hughes and Bart Jacobs. Simulations in coalgebra. *Theor. Comput. Sci.*, 327(1-2):71–108, 2004. doi:10.1016/j.tcs.2004.07.022.

29 Michael Huth and Marta Kwiatkowska. Quantitative analysis and model checking. In *Logic in Computer Science, LICS 1997*, pages 111–122. IEEE, 1997.

30 Narges Khakpour and Mohammad Mousavi. Notions of conformance testing for cyber-physical systems: Overview and roadmap (invited paper). In *Concurrency Theory, CONCUR 2015*, volume 42 of *LIPIcs*, pages 18–40. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CONCUR.2015.18.

31 Barbara König and Christina Mika-Michalski. (Metric) bisimulation games and real-valued modal logics for coalgebras. In Sven Schewe and Lijun Zhang, editors, *Concurrency Theory, CONCUR 2018*, volume 118 of *LIPIcs*, pages 37:1–37:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.37.

32 Alexander Kurz and Raul Leal. Equational coalgebraic logic. In *Mathematical Foundations of Programming Semantics, MFPS 2009*, volume 249 of *ENTCS*, pages 333–356. Elsevier, 2009.

33 Paul Levy. Similarity quotients as final coalgebras. In *Foundations of Software Science and Computational Structures, FOSSACS 2011*, volume 6604 of *LNCS*, pages 27–41. Springer, 2011. doi:10.1007/978-3-642-19805-2.

34 Thomas Lukasiewicz and Umberto Straccia. Managing uncertainty and vagueness in description logics for the semantic web. *J. Web Sem.*, 6(4):291–308, 2008. doi:10.1016/j.websem.2008.04.001.

35 Johannes Marti and Yde Venema. Lax extensions of coalgebra functors and their logic. *J. Comput. Syst. Sci.*, 81(5):880–900, 2015.

36 Stefan Milius, Dirk Pattinson, and Lutz Schröder. Generic trace semantics and graded monads. In *Algebra and Coalgebra in Computer Science, CALCO 2015*, Leibniz International Proceedings in Informatics, 2015.

37 Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

38 Matteo Mio and Alex Simpson. Łukasiewicz μ-calculus. *Fund. Inf.*, 150(3–4):317–346, 2017.

39 Matteo Mio and Valeria Vignudelli. Monads and quantitative equational theories for non-determinism and probability, 2020. This volume; full version available as arXiv e-print arXiv:2005.07509. doi:10.4230/LIPIcs.CONCUR.2020.28.

40 Carroll Morgan and Annabelle McIver. A probabilistic temporal calculus based on expectations. In Lindsay Groves and Steve Reeves, editors, *Formal Methods Pacific, FMP 1997*. Springer, 1997.

41 Charles Morgan. Local and global operators and many-valued modal logics. *Notre Dame J. Formal Log.*, 20:401–411, 1979.

42 Lawrence Moss. Coalgebraic logic. *Ann. Pure Appl. Logic*, 96:277–317, 1999.

43 David Park. Concurrency and automata on infinite sequences. In *Theoretical Computer Science, 5th GI-Conference*, volume 104 of *LNCS*, pages 167–183. Springer, 1981. doi:10.1007/BFb0017288.

44 Dirk Pattinson. Expressive logics for coalgebras via terminal sequence induction. *Notre Dame J. Formal Log.*, 45:19–33, 2004.

45 Ricardo Rodriguez and Lluis Godo. Modal uncertainty logics with fuzzy neighborhood semantics. In *Weighted Logics for Artiticial Intelligence, WL4AI 2013 (Workshop at IJCAI 2013)*, pages 79–86, 2013.

46 Jan Rutten. Universal coalgebra: A theory of systems. *Theoret. Comput. Sci.*, 249:3–80, 2000.

47 Lutz Schröder. Expressivity of coalgebraic modal logic: The limits and beyond. *Theoret. Comput. Sci.*, 390:230–247, 2008.

48 Lutz Schröder and Dirk Pattinson. Description logics and fuzzy probability. In Toby Walsh, editor, *Int. Joint Conf. Artificial Intelligence, IJCAI 2011*, pages 1075–1081. AAAI, 2011.

49 Umberto Straccia. A fuzzy description logic. In *Artificial Intelligence, AAAI 1998*, pages 594–599. AAAI Press / MIT Press, 1998.

50 Albert Thijs. *Simulation and fixpoint semantics*. PhD thesis, University of Groningen, 1996.

51 Věra Trnková. General theory of relational automata. *Fund. Inform.*, 3:189–234, 1980.

52 Franck van Breugel, Claudio Hermida, Michael Makkai, and James Worrell. Recursively defined metric spaces without contraction. *Theor. Comput. Sci.*, 380(1-2):143–163, 2007. doi:10.1016/j.tcs.2007.02.059.

53 Franck van Breugel and James Worrell. A behavioural pseudometric for probabilistic transition systems. *Theor. Comput. Sci.*, 331:115–142, 2005.

54 Cédric Villani. *Optimal Transport: Old and New*. Springer, 2008.

55 Igor Walukiewicz. Completeness of Kozen's axiomatisation of the propositional μ-calculus. In *Logic in Computer Science, LICS 1995*, pages 14–24. IEEE Computer Society, 1995. doi:10.1109/LICS.1995.523240.

56 Paul Wild and Lutz Schröder. Characteristic logics for behavioural metrics via fuzzy lax extensions, 2020. arXiv e-print. arXiv:2007.01033.

57 Paul Wild, Lutz Schröder, Dirk Pattinson, and Barbara König. A van Benthem theorem for fuzzy modal logic. In *Logic in Computer Science, LICS 2018*, pages 909–918. ACM, 2018.

A Appendix

We assume w.l.o.g. that all functors preserve injective maps [5].

Proof of Lemma 3.7

- 1. $\Longleftrightarrow$ 2.: The implication "$\Leftarrow$" is trivial; we prove "$\Rightarrow$". We have

$$\begin{aligned} L\mathrm{Gr}_{\epsilon,f} &= L(\Delta_{\epsilon,B} \circ (\mathsf{id}_B \times f)) \\ &= L\Delta_{\epsilon,B} \circ (\mathsf{id}_{TB} \times Tf) && \text{(Lemma 3.8)} \\ &\leq \Delta_{\epsilon,TB} \circ (\mathsf{id}_{TB} \times Tf) \\ &= \mathrm{Gr}_{\epsilon,Tf} \end{aligned}$$

- 1. $\Longrightarrow$ 3.: Let $R_1, R_2 \colon A \nrightarrow B$ and $\epsilon > 0$ such that $\|R_1 - R_2\|_\infty \leq \epsilon$; we need to show that $\|LR_1 - LR_2\|_\infty \leq \epsilon$. The assumption implies $R_1 \leq R_2; \Delta_{\epsilon,B}$, hence $LR_1 \leq L(R_2; \Delta_{\epsilon,B}) \leq LR_2; L\Delta_{\epsilon,B} \leq LR_2; \Delta_{\epsilon,TB}$ using (L1), (L2), and (L4). Symmetrically, we show $LR_2 \leq LR_1; \Delta_{\epsilon,TB}$, so that $\|LR_1 - LR_2\|_\infty \leq \epsilon$.
- 3. $\Longrightarrow$ 1.: We have $\|\Delta_{\epsilon,A} - \Delta_A\|_\infty = \epsilon$, and hence by assumption $\|L\Delta_{\epsilon,A} - L\Delta_A\|_\infty \leq \epsilon$. In particular, $L\Delta_{\epsilon,A} - L\Delta_A \leq \Delta_{\epsilon,TA}$, so

  $$L\Delta_{\epsilon,A} \leq L\Delta_A; \Delta_{\epsilon,TA} \leq \Delta_{TA}; \Delta_{\epsilon,TA} = \Delta_{\epsilon,TA}$$

  using (L3).

Proof of Theorem 4.7

By the fixpoint definition of $d^L_{\alpha,\beta}$, (ii) is immediate from (i). We prove (i), i.e. that $Ld_\omega(\alpha(a), \beta(b)) = d_\omega(a, b)$ for all $a \in A$, $b \in B$. We begin by assuming that T is finitary, and generalise to the non-finitary case later.

Since T is finitary, there exist finite subsets $A_0 \subseteq A$, $B_0 \subseteq B$ and $s \in TA_0$, $t \in TB_0$ such that $\alpha(a) = Ti(s)$ and $\beta(b) = Tj(t)$, where $i \colon A_0 \to A$ and $j \colon B_0 \to B$ are the inclusion maps. We then have $Ld_\omega(\alpha(a), \beta(b)) = L(d_\omega \circ (i \times j))(s, t)$ by naturality (Lemma 3.8). Now the $d_n \circ (i \times j)$ converge to $d_\omega \circ (i \times j)$ pointwise, and therefore also under the supremum metric (i.e. uniformly), since $A_0 \times B_0$ is finite. Since the assumptions imply that L is continuous w.r.t. the supremum metric, it follows that

$$\begin{aligned}
&L(d_\omega \circ (i \times j))(s, t) \\
&= \sup_{n<\omega} L(d_n \circ (i \times j))(s, t) \\
&= \sup_{n<\omega} Ld_n(\alpha(a), \beta(b)) && \text{(naturality)} \\
&= \sup_{n<\omega} d_{n+1}(a, b) = d_\omega(a, b).
\end{aligned}$$

For non-finitary T, we refer to the full version.

Details for Remark 5.2

An evaluation function $e \colon T[0,1] \to [0,1]$ gives rise to a unary predicate lifting λ_e by putting $\lambda_e(f) = e \circ Tf$. Conversely, an evaluation function for $\lambda \colon \mathcal{Q} \Rightarrow \mathcal{Q} \circ T$ can be defined via $e_\lambda = \lambda_{[0,1]}(\mathsf{id})$.

Generalizing to higher arities, an n-ary evaluation function is a map $e \colon T([0,1]^n) \to [0,1]$, and gives rise to a predicate lifting $\lambda_e(f_1, \dots, f_n) = e \circ T\langle f_1, \dots f_n\rangle$, while for each n-ary predicate lifting λ the corresponding evaluation function is $e_\lambda = \lambda_{[0,1]^n}(\pi_1, \dots, \pi_n)$.

Details for Definition 5.5

From the definition it is clear that $K_\Lambda R(t_1, t_2) \in [-1, 1]$ for all t_1 and t_2. To see that $K_\Lambda R(t_1, t_2) \geq 0$, consider the maps $h_X \colon X \to [0,1], x \mapsto \frac{1}{2}$ for any set X. The pair (h_A, h_B) is clearly R-nonexpansive and so, for some arbitrary unary $\lambda \in \Lambda$

$$\begin{aligned}
K_\Lambda R(t_1, t_2) &\geq \max(\lambda_A(h_A)(t_1) - \lambda_B(h_B)(t_2), \bar{\lambda}_A(h_A)(t_1) - \bar{\lambda}_B(h_B)(t_2)) \\
&= |\lambda_A(h_A)(t_1) - \lambda_B(h_B)(t_2)| \geq 0.
\end{aligned}$$

(If λ has higher arity, just supply more copies of h_A and h_B.)

Proof of Theorem 5.6

The following lemma will be used in the proof of (L2):

▶ **Lemma A.1.** *Let $R\colon A \nrightarrow B, S\colon B \nrightarrow C$. Then for every $(R;S)$-nonexpansive pair (f,h) there exists some function $g\colon B \to [0,1]$ such that (f,g) is R-nonexpansive and (g,h) is S-nonexpansive.*

Proof. For each $b \in B$ the value $g(b)$ can be chosen arbitrarily in the interval

$$[\sup_{a\in A} f(a) \ominus R(a,b), \inf_{c\in C} h(c) \oplus S(b,c)],$$

so for instance we can use the nonexpansive companion $g := R[f]$ (Definition 5.4). This interval is non-empty because by assumption

$$\begin{aligned} f(a) - h(c) &\le (R;S)(a,c) \\ &\le \inf_{b'\in B} R(a,b') + S(b',c) \\ &\le R(a,b) + S(b,c) \end{aligned}$$

for all $a \in A, c \in C$, so $f(a) - R(a,b) \le h(c) + S(b,c)$ by rearranging. Similar rearranging also shows that choosing $g(b)$ in this way ensures that (f,g) is R-nonexpansive and (g,h) is S-nonexpansive. ◀

Now we are ready for the main proof.

Proof. For readability, we pretend that all $\lambda \in \Lambda$ are unary although the proof works just as well for unrestricted arities, whose treatment requires no more than adding indices. We show the five properties one by one:

- (L0): Let $R\colon A \nrightarrow B$ and $t_1 \in TA, t_2 \in TB$. Note that a pair (g,f) is R°-nonexpansive iff $(1-f, 1-g)$ is R-nonexpansive. Now, using that Λ is closed under duals,

$$\begin{aligned} K_\Lambda(R^\circ)(t_2,t_1) &= \sup\{\lambda_B(g)(t_2) - \lambda_A(f)(t_1) \mid \lambda \in \Lambda, (g,f)\ R^\circ\text{-nonexp.}\} \\ &= \sup\{\bar\lambda_A(f)(t_1) - \bar\lambda_B(g)(t_2) \mid \lambda \in \Lambda, (f,g)\ R\text{-nonexp.}\} = K_\Lambda R(t_1,t_2) \end{aligned}$$

- (L1): Let $R_1 \le R_2$. Then every R_1-nonexpansive pair is also R_2-nonexpansive. Thus $K_\Lambda R_1 \le K_\Lambda R_2$, because the supremum on the left side is taken over a subset of that on the right side.
- (L2): Let $R\colon A \nrightarrow B, S\colon B \nrightarrow C$ and $t_1 \in TA, t_2 \in TB, t_3 \in TC$. Let $\lambda \in \Lambda$ and let (f,h) be $(R;S)$-nonexpansive. Let g be given by Lemma A.1. Then it is enough to observe that:

$$\begin{aligned} \lambda_A(f)(t_1) - \lambda_C(h)(t_3) &= (\lambda_A(f)(t_1) - \lambda_B(g)(t_2)) + (\lambda_B(g)(t_2) - \lambda_C(h)(t_3)) \\ &\le K_\Lambda R(t_1,t_2) + K_\Lambda S(t_2,t_3). \end{aligned}$$

- (L3): Let $h\colon A \to B$ and $t \in TA$. We need to show that $K_\Lambda \mathrm{Gr}_h(t, Th(t)) = 0$. Let $\lambda \in \Lambda$ and let (f,g) be Gr_h-nonexpansive, implying $f \le g \circ h$. Then

$$\lambda_A(f)(t) \le \lambda_A(g\circ h)(t) = \lambda_B(g)(Th(t)),$$

  by monotonicity and naturality of λ.
- (L4): Let A be a set, $t \in TA$ and $\epsilon > 0$. We need to show that $K_\Lambda \Delta_{\epsilon,A}(t,t) \le \epsilon$. Let $\lambda \in \Lambda$ and let (f,g) be $\Delta_{\epsilon,A}$-nonexpansive, implying $f(a) - g(a) \le \epsilon$ for all $a \in A$. By monotonicity of λ, we can restrict our attention to the case $g(a) = f(a) \ominus \epsilon$. In this case,

$$\lambda(f)(t) - \lambda(g)(t) \le \|\lambda(f) - \lambda(g)\|_\infty \le \|f - g\|_\infty \le \epsilon.$$ ◀

Details for Remark 5.7

First, note that if (f,g) with $f,g\colon A\to[0,1]$ is d-nonexpansive, then $f(a)-g(a)\le d(a,a)=0$ for all $a\in A$, so $f\le g$. By monotonicity of the $\lambda\in\Lambda$ the value of the supremum in Definition 5.5 does not change if we restrict the choice of (f,g) to the case $f=g$. Finally, in the case $f=g$, d-nonexpansivity implies that $f(a)-f(b)\le d(a,b)$ and $f(b)-f(a)\le d(b,a)=d(a,b)$ for every $a,b\in A$, which means that f is in fact a nonexpansive map $f\colon(A,d)\to_1([0,1],d_E)$. Also the supremum does not change when taking the absolute value, because f is nonexpansive iff $1-f$ is and Λ is closed under duals.

Details for Example 5.8.1

We show that K is finitarily separable. Let $\mu\in\mathcal{D}X$ and $\epsilon>0$. We need to find $\mu_\epsilon\in\mathcal{D}X$ with finite support such that $K\Delta_X(\mu,\mu_\epsilon)\le\epsilon$. Note that a pair (f,g) is Δ_X-nonexpansive iff $f\le g$, so by monotonicity

$$K\Delta_X(\mu,\mu_\epsilon)=\sup\{\textstyle\sum_{x\in X}f(x)(\mu(x)-\mu_\epsilon(x))\mid f\colon X\to[0,1]\}\le\sum_{x\in X}|\mu(x)-\mu_\epsilon(x)|.$$

Because μ is discrete, there exists a finite set $Y\subseteq X$ with $\sum_{x\in Y}\mu(x)\ge 1-\frac{\epsilon}{2}$. If $Y=X$, then we can just put $\mu_\epsilon=\mu$. Otherwise, let $x_0\in X\setminus Y$. Then we define μ_ϵ as follows: $\mu_\epsilon(x_0)=\mu(Y)$, $\mu_\epsilon(x)=\mu(x)$ for $x\in Y$, and $\mu_\epsilon(x)=0$ otherwise. In this case,

$$\textstyle\sum_{x\in X}|\mu(x)-\mu_\epsilon(x)|\le 2\mu(Y)\le\epsilon.$$

Details for Remark 6.4

We recall the definition of well-behaved evaluation functions from [2]:

▶ **Definition A.2.** An evaluation function $e\colon T[0,1]\to[0,1]$ is *well-behaved* if it satisfies the following:

1. The predicate lifting λ_e is monotone.
2. For all $t\in T([0,1]^2)$, we have $d_E(e(t_1),e(t_2))\le\lambda_e(d_E)(t)$, where $t_1=T\pi_1(t)$ and $t_2=T\pi_2(t)$.
3. $e^{-1}[\{0\}]=Ti[T\{0\}]$, where $i\colon\{0\}\to[0,1]$ is the inclusion map.

This notion is almost equivalent to that of a standard predicate lifting in the following sense:

▶ **Lemma A.3.** *e is a well-behaved evaluation function iff the predicate lifting λ_e is standard and $e^{-1}[\{0\}]\subseteq Ti[T\{0\}]$.*

Proof. First, note that monotonicity of λ_e features in both notions and λ_e preserves zero iff $e^{-1}[\{0\}]\supseteq Ti[T\{0\}]$. It remains to relate Item 2 of Definition A.2 with subadditivity of λ. Reformulating in terms of λ_e gives

$$|\lambda_e(\pi_1)(t)-\lambda_e(\pi_2)(t)|\le\lambda_e(d_E)(t)\quad\text{for } t\in T([0,1]^2).\tag{4}$$

We show that (4) is equivalent to subadditivity of λ_e, given that λ_e is monotone:

- "⇒": Let $f,g\in\mathcal{Q}X,t\in TX$. Put $t':=T\langle f\oplus g,f\rangle(t)\in T([0,1]^2)$. Then, by naturality, we have $\lambda_e(\pi_1)(t')=\lambda_e(f\oplus g)(t)$ and $\lambda_e(\pi_2)(t')=\lambda_e(f)(t)$ and

  $$\lambda_e(d_E)(t')=\lambda_e(d_E\circ\langle f\oplus g,f\rangle)(t)\le\lambda_e(g)(t),$$

  where we used monotonicity of λ_e in the last step. Therefore, $\lambda(f\oplus g)(t)-\lambda(f)(t)\le\lambda(g)(t)$ by (4).

- "$\Leftarrow$": Put $f = d_E, g = \pi_1 : [0,1]^2 \to [0,1]$. Then it is easily checked that $f \oplus g \geq \pi_2$ and therefore

  $$\lambda_e(\pi_2) \leq \lambda_e(f \oplus g) \leq \lambda_e(f) + \lambda_e(g) = \lambda_e(d_E) + \lambda_e(\pi_1)$$

  by monotonicity and subadditivity of λ_e, so $\lambda_e(\pi_1) - \lambda_e(\pi_2) \leq \lambda_e(d_E)$. Similarly, we can show that $\lambda_e(\pi_2) - \lambda_e(\pi_1) \leq \lambda_e(d_E)$ by swapping the roles of π_1 and π_2. ◀

In [26, Definition 3.1], topological theories are defined as triples consisting of a monad T, a quantale V, and a map $\xi : TV \to V$ satisfying a number of axioms. We only consider the case of the quantale $[0,1]^{\mathrm{op}}$, with the order given by $\geq$ and the monoid structure by $\oplus$. The first two axioms state that ξ is a T-algebra and can be ignored for our purposes. The remaining axioms instantiate as follows, where as usual $\lambda_\xi(f) = \xi \circ Tf$ is the predicate lifting associated with ξ:

- $(Q_\otimes)$ $\quad \oplus \circ \langle \lambda_\xi(\pi_1), \lambda_\xi(\pi_2) \rangle \geq \lambda_\xi(\oplus)$
- (Q_k) $\quad 0 \geq \lambda_\xi(0_1)(t)$ for every $t \in T1$, where 1 is a singleton set
- $(Q'_\vee)$ $\quad \lambda_\xi$ is a monotone natural transformation

Using a similar idea as in Lemma A.3, we see that $(Q_\otimes)$ is equivalent to subadditivity of λ_ξ and (Q_k) is equivalent to preservation of the zero function. Finally note that [26, Theorem 3.5 (d)] (which states that the Wasserstein lifting satisfies (L2)) requires that the functor satisfies the *Beck-Chevalley condition*, i.e. preserves weak pullbacks.

Proof of Example 6.6.1

Let $R : A \nrightarrow B$, and let $U \subseteq A$ and $V \subseteq B$. We show $HR(U,V) = W_{\{\lambda\}}R(U,V)$. There are two inequalities:

- "$\leq$": Let $Z \in \mathsf{Cpl}(U,V)$. Then for every $a \in U$ there exists $b \in V$ such that $(a,b) \in Z$, so $\inf_{b \in V} R(a,b) \leq \sup R[Z]$. Thus, we have $\sup_{a \in U} \inf_{b \in V} R(a,b) \leq \sup R[Z]$, and, by a symmetrical argument, $\sup_{b \in V} \inf_{a \in U} R(a,b) \leq \sup R[Z]$.
- "$\geq$": It is enough to find for each $\epsilon > 0$ a coupling $Z \in \mathsf{Cpl}(U,V)$ such that $\sup R[Z] \leq HR(U,V) + \epsilon$. So let $\epsilon > 0$. We construct functions $f : U \to V$ and $g : V \to U$ as follows: For each $a \in U$ choose $f(a) \in V$ such that $R(a, f(a)) \leq \inf_{b \in V} R(a,b) + \epsilon$. Similarly, for each $b \in V$ choose $g(b) \in U$ such that $R(g(b), b) \leq \inf_{a \in U} R(a,b) + \epsilon$. Now put $Z = \{(a, f(a)) \mid a \in U\} \cup \{(g(b), b) \mid b \in V\}$. Clearly, $Z \in \mathsf{Cpl}(U,V)$ and by construction,

  $$\sup R[Z] = \max(\sup_{a \in U} R(a, f(a)), \sup_{b \in V} R(g(b), b)) \leq HR(U,V) + \epsilon.$$

Details for Example 6.6.2

We denote the Wasserstein lifting of the distribution functor $\mathcal{D}$ by W. Let $R : A \nrightarrow B$, and let $U \in \mathcal{C}A$ and $V \in \mathcal{C}B$. We show $W_{\{\lambda\}}(R)(U,V) = HW(R)(U,V)$. There are two inequalities:

- "$\geq$": Let $Z \in \mathsf{Cpl}_{\mathcal{C}}(U,V)$. We put $Y = \mathcal{P}\langle \mathcal{D}\pi_1, \mathcal{D}\pi_2 \rangle(Z)$. Then $\mathcal{P}\pi_1(Y) = \mathcal{P}\mathcal{D}\pi_1(Z) = \mathcal{C}\pi_1(Z) = U$ and similarly $\mathcal{P}\pi_2(Y) = V$, so that $Y \in \mathsf{Cpl}_{\mathcal{P}}(U,V)$. Now, note that for every $\mu \in \mathcal{D}(A \times B)$ we have that $\mathbb{E}_\mu(R) \geq WR(\mathcal{D}\pi_1(\mu), \mathcal{D}\pi_2(\mu))$ and therefore

  $$\sup_{\mu \in Z} \mathbb{E}_\mu(R) \geq \sup_{(\mu_1,\mu_2) \in Y} WR(\mu_1, \mu_2) \geq HW(R)(U,V).$$

- "$\leq$": Let $Y \in \mathsf{Cpl}_{\mathcal{P}}(U,V)$. It is enough to find for each $\epsilon > 0$ some $Z \in \mathsf{Cpl}_{\mathcal{C}}(\mu_1, \mu_2)$ such that

  $$\sup_{\mu \in Z} \mathbb{E}_\mu(R) \leq \sup_{(\mu_1,\mu_2) \in Y} WR(\mu_1, \mu_2) + \epsilon.$$

For every $(\mu_1, \mu_2) \in \mathcal{D}A \times \mathcal{D}B$ there exists some $\mu \in \mathsf{Cpl}_{\mathcal{D}}(U, V)$ such that $\mathbb{E}_\mu(R) \leq WR(\mu_1, \mu_2) + \epsilon$. Let Z' be a set consisting of one such μ for every pair $(\mu_1, \mu_2) \in Y$ and put $Z = \mathrm{conv}(Z')$, where conv is convex hull. Then we have

$$\mathcal{C}\pi_1(Z) = \mathcal{P}\mathcal{D}\pi_1(\mathrm{conv}(Z')) = \mathrm{conv}(\mathcal{P}\mathcal{D}\pi_1(Z')) = \mathrm{conv}(U) = U.$$

Here we made use of the fact that $\mathcal{D}\pi_1$ is linear when considered as a map $\mathbb{R}^{A\times B} \to \mathbb{R}^A$, and linear maps preserve convex sets. We similarly get $\mathcal{C}\pi_2(Z) = V$, so that $Z \in \mathsf{Cpl}_{\mathcal{C}}(U, V)$. Finally, we note that taking expected values is a linear operation, so if $\mu = \sum_{i=1}^n p_i\mu_i$ is a convex combination of probability measures, then $\mathbb{E}_\mu = \sum_{i=1}^n p_i\mathbb{E}_{\mu_i} \leq \max_{i=1}^n \mathbb{E}_{\mu_i}$. Therefore we have, as desired,

$$\sup_{\mu\in Z} \mathbb{E}_\mu(R) = \sup_{\mu\in Z'} \mathbb{E}_\mu(R) \leq \sup_{(\mu_1,\mu_2)\in Y} WR(\mu_1, \mu_2) + \epsilon.$$

Proof of Lemma 8.4

Immediate from the following lemma:

▶ **Lemma A.4.** *Let ϕ be a modal Λ-formula, and let $a \in A$, $b \in B$ be states in T-coalgebras (A, α), (B, β). Then $[\![\phi]\!]_{A,\alpha}(a) - [\![\phi]\!]_{B,\beta}(b)| \leq d^{K_\Lambda}_{\alpha,\beta}(a, b)$.*

Proof. Induction on ϕ, with trivial Boolean cases (in Zadeh semantics, all propositional operators on $[0, 1]$ are non-expansive). For the modal case, we have (for readability, restricting to unary $\lambda \in \Lambda$ and omitting subscripts)

$$\begin{aligned}
|[\![\lambda(\phi)]\!](a) - [\![\lambda(\phi)]\!](b)| &= |\lambda_A([\![\phi]\!])(\alpha(a)) - \lambda_B([\![\phi]\!](\beta(b))| \\
&\leq K_\Lambda d^{K_\Lambda}(\alpha(a), \beta(b)) && \text{(definition, IH)} \\
&= d^{K_\Lambda}(a, b) && \text{(definitionup)}
\end{aligned}$$

◀

Proof of Corollary 8.10

Immediate from Corollary 8.9 once one notes that the closure under duals incorporated in the Definition of the Kantorovich distance is ensured by the presence of negation in quantitative Moss logic.

Monads and Quantitative Equational Theories for Nondeterminism and Probability

Matteo Mio
Université Lyon, CNRS, ENS Lyon, UCB Lyon 1, LIP, France

Valeria Vignudelli
Université Lyon, CNRS, ENS Lyon, UCB Lyon 1, LIP, France

Abstract

The monad of convex sets of probability distributions is a well-known tool for modelling the combination of nondeterministic and probabilistic computational effects. In this work we lift this monad from the category of sets to the category of extended metric spaces, by means of the Hausdorff and Kantorovich metric liftings. Our main result is the presentation of this lifted monad in terms of the quantitative equational theory of convex semilattices, using the framework of quantitative algebras recently introduced by Mardare, Panangaden and Plotkin.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Axiomatic semantics; Theory of computation → Categorical semantics

Keywords and phrases Computational Effects, Monads, Metric Spaces, Quantitative Algebras

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.28

Funding Both authors have been partially supported by the French project ANR-16-CE25-0011 REPAS. Vignudelli has been partially supported by the European Research Council (ERC) under the European Union's Horizon 2020 programme (CoVeCe, grant agreement No 678157) and by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

Acknowledgements The authors are grateful to the anonymous reviewers and to the authors of [7] for their useful comments and suggestions.

1 Introduction

In the theory of programming languages the categorical concept of *monad* is used to handle computational effects [43, 44]. As main examples, the *powerset monad* ($\mathcal{P}$) and the *probability distribution monad* ($\mathcal{D}$) are used to handle nondeterministic and probabilistic behaviours, respectively. It is of course desirable to handle the combination of these two effects to model, for instance, concurrent randomised protocols where nondeterminism arises from the action of an unpredictable scheduler and probability from the use of randomised procedures such as coin tosses. However, the composite functor $\mathcal{P} \circ \mathcal{D}$ is not a monad (see, e.g., [52]).

A well–known way to handle this technical issue is to use instead the *convex powerset of distributions monad* ($\mathcal{C}$) which restricts $\mathcal{P}\circ\mathcal{D}$ by only admitting sets of probability distributions that are closed under the formation of *convex combinations* (see [50, 29, 28, 42, 41, 33, 39] and Section 2). Restricting $\mathcal{P} \circ \mathcal{D}$ to $\mathcal{C}$ is not only mathematically convenient, because it leads to a monad, but also natural as convexity captures the possibility of a scheduler to make probabilistic choices, as originally observed by Segala [46]. Suppose indeed that a scheduler can select between two probabilistic behaviours $\{d_1, d_2\}$ for execution. It is reasonable to assume that said scheduler can also, with the aid of a (biased) coin, choose d_1 with probability p and d_2 with probability $1 - p$. Hence, effectively, the scheduler can choose any behaviour in $\{p \cdot d_1 + (1 - p) \cdot d_2 \mid p \in [0, 1]\}$, which is indeed a convex set of distributions.

In a recent work [13] the authors provide a proof for the following result: the equational theory $\mathtt{Th}_{CS}$ of convex semilattices is a *presentation* of the **Set** monad $\mathcal{C}$. This means (see Section 2 for details) that the category $\mathbf{A}(\mathtt{Th}_{CS})$ of convex semilattices and their homomorphisms is isomorphic to the category $\mathbf{EM}(\mathcal{C})$ of Eilenberg-Moore algebras for $\mathcal{C}$.

Presentation results of this kind have a number of applications in computer science due to the interplay between the structure (syntax) and the dynamics (behaviour) of systems. For example, it follows from the presentation result of [13] that the free convex semilattice with set of generators X is isomorphic to $\mathcal{C}(X)$. This allows us to manipulate elements of $\mathcal{C}(X)$ as convex semilattice terms modulo the equations of $\mathtt{Th}_{CS}$ and, similarly, to perform equational reasoning steps using facts (e.g., from geometry) related to the mathematical structure of $\mathcal{C}(X)$. Applications in the field of program semantics and concurrency theory arise by combining coalgebraic reasoning methods, associated with the use of monads as behaviour functors, and algebraic methods, which are made available by presentation theorems. Well known examples include *bisimulation up–to techniques* (e.g., up–to congruence [11]) and the categorical approach to structural operational semantics, introduced by Turi and Plotkin in [51] (see also [35]) and based on the notion of *bialgebras*.

The category **EMet**, having extended metric spaces as objects and non–expansive maps as morphisms, is a natural mathematical setting[1] which can replace the category **Set** when it is desirable to switch from the concept of *program equivalence* to that of *program distance*. This has been a very active topic of research in the last two decades (see, e.g, [45, 27, 15, 23, 16]). In this context, it is necessary to deal with monads on **EMet**. Variants of the **Set** monads $\mathcal{P}$ and $\mathcal{D}$ have been proposed on **EMet** (see, e.g., [15, 8] and Section 3), and are technically based on different types of *metric liftings*, due to Hausdorff and Kantorovich.

Contributions of this work. In this work we investigate a **EMet** variant of the **Set** monad $\mathcal{C}$, which we denote by $\hat{\mathcal{C}}$. As a functor, $\hat{\mathcal{C}} : \mathbf{EMet} \to \mathbf{EMet}$ maps a metric space (X, d) to the metric space $(\mathcal{C}(X), HK(d))$, the collection of non–empty, finitely generated convex sets of finitely supported probability distributions on X endowed with the metric $H(K(d))$, the Hausdorff lifting of the Kantorovich lifting of the metric d.

$$\hat{\mathcal{C}} : \mathbf{EMet} \to \mathbf{EMet} \qquad (X, d) \mapsto \Big(\mathcal{C}(X), H(K(d))\Big).$$

As a first contribution, in Section 4 we give a direct proof of the fact that $\hat{\mathcal{C}}$ is indeed a monad on **EMet**. This result does not seem straightforward to prove. Most notably, establishing the non–expansiveness of the monad multiplication $\mu^{\hat{\mathcal{C}}}$ requires some detailed calculations.

Our second and main result concerns the presentation of the **EMet** monad $\hat{\mathcal{C}}$. Presentations of monads in **Set** are given in terms of categories of algebras (in the sense of universal algebra) and their homomorphisms, but these are not adequate in the metric setting. For this reason we use, instead, the recently introduced apparatus of *quantitative algebras* and *quantitative equational theories* of [36] (see also [37, 7, 5, 4]). This framework generalises that of universal algebra and equational reasoning by dealing with quantitative algebras, which are metric spaces equipped with non–expansive operations over a signature, and quantitative equations of the form $s =_\epsilon t$, intuitively expressing that the distance between terms s and t is less than or equal to ϵ. In Section 4 we define the quantitative equational theory $\mathtt{QTh}_{CS}$ of

[1] The category **EMet** of extended metric spaces carries additional categorical structure compared to the category **Met** of ordinary metric spaces such as, e.g., the existence of coproducts. This structure is often useful in the field of program semantics. All the results obtained in this paper can be easily adapted to hold in the category **Met**.

quantitative convex semilattices, and in Section 5 we prove the presentation result (Theorem 36): the category $\mathbf{EM}(\hat{\mathcal{C}})$ of Eilenberg-Moore algebras for $\hat{\mathcal{C}}$ is isomorphic to the category $\mathbf{QA}(\mathtt{QTh}_{CS})$ of quantitative convex semilattices and their non–expansive homomorphims.

Relation with other works. This work continues the research path opened in the seminal [36] (see also subsequent works [37, 7, 5, 4]) where the authors investigated the connection between the quantitative theories of semilattices ($\mathtt{QTh}_{SL}$) and convex algebras ($\mathtt{QTh}_{CA}$) and the monads $\hat{\mathcal{P}}$ and $\hat{\mathcal{D}}$, which are **EMet** variants of $\mathcal{P}$ and $\mathcal{D}$, respectively. Hence, our work constitutes a natural step forward. From a technical standpoint, there is a difference between our main presentation result and those of [36] regarding $\mathtt{QTh}_{SL}$ and $\mathtt{QTh}_{CA}$ (corollaries 9.4 and 10.6 respectively in [36]). Indeed, in [36] the authors only provide representations of the free objects in the categories $\mathbf{QA}(\mathtt{QTh}_{SL})$ and $\mathbf{QA}(\mathtt{QTh}_{CA})$. While this suffices in many applications, we believe that proving a full presentation, in the sense introduced and investigated in this work, provides a more general and useful result, giving a representation for the whole categorical structure and not just for free objects. This said, the technical machinery developed in [36] suffices, with minor additional work[2], to establish the following presentation results in our sense: $\mathbf{QA}(\mathtt{QTh}_{SL}) \cong \mathbf{EM}(\hat{\mathcal{P}})$ and $\mathbf{QA}(\mathtt{QTh}_{CA}) \cong \mathbf{EM}(\hat{\mathcal{D}})$.

2 Monads on Sets and Equational Theories

In this section we present basic definitions and results regarding monads. We assume the reader is familiar with the basic concepts of category theory (see [3] as a reference).

▶ **Definition 1.** *Given a category* $\mathbf{C}$*, a monad on* $\mathbf{C}$ *is a triple* $(\mathcal{M}, \eta, \mu)$ *composed of a functor* $\mathcal{M}\colon \mathbf{C} \to \mathbf{C}$ *together with two natural transformations: a unit* $\eta\colon id \Rightarrow \mathcal{M}$*, where* id *is the identity functor on* $\mathbf{C}$*, and a multiplication* $\mu\colon \mathcal{M}^2 \Rightarrow \mathcal{M}$*, satisfying the two laws* $\mu \circ \eta\mathcal{M} = \mu \circ \mathcal{M}\eta = id$ *and* $\mu \circ \mathcal{M}\mu = \mu \circ \mu\mathcal{M}$*.*

We now introduce three relevant monads on the category **Set** of sets and functions.

▶ **Definition 2.** *The* non–empty finite powerset *monad* $(\mathcal{P}, \eta^{\mathcal{P}}, \mu^{\mathcal{P}})$ *on* **Set** *is defined as follows. Given an object* X *in* **Set**, $\mathcal{P}(X) = \{X' \subseteq X \mid X' \neq \emptyset \text{ and } X' \text{ is finite}\}$*. Given an arrow* $f : X \to Y$*,* $\mathcal{P}(f) : \mathcal{P}(X) \to \mathcal{P}(Y)$ *is defined as* $\mathcal{P}(f)(X') = \bigcup_{x \in X'} f(x)$ *for any* $X' \in \mathcal{P}(X)$*. The unit* $\eta^{\mathcal{P}}_X : X \to \mathcal{P}(X)$ *is defined as* $\eta^{\mathcal{P}}_X(x) = \{x\}$*, and the multiplication* $\mu^{\mathcal{P}}_X : \mathcal{P}\mathcal{P}(X) \to \mathcal{P}(X)$ *is defined as* $\mu^{\mathcal{P}}_X(\{X_1, \ldots, X_n\}) = \bigcup_{i=1}^n X_i$*.*

A probability distribution on a set X is a function $\Delta : X \to [0, 1]$ such that $\sum_{x \in X} \Delta(x) = 1$. The *support* of Δ is defined as the set $supp(\Delta) = \{x \in X \mid \Delta(x) \neq 0\}$. In this paper we only consider probability distributions with finite support which we often just refer to as distributions. The Dirac distribution $\delta(x)$ is defined as $\delta(x)(x') = 1$ if $x' = x$ and $\delta(x)(x') = 0$ otherwise. We often denote a distribution having $supp(\Delta) = \{x_1, x_2\}$ using the expression $p_1 x_1 + p_2 x_2$, with $p_i = \Delta(x_i)$. Analogously, we let $\sum_{i=1}^n p_i x_i$ denote a distribution Δ with support $\{x_1, \ldots, x_n\}$ and with $p_i = \Delta(x_i)$.

▶ **Definition 3.** *The* finitely supported probability distribution *monad* $(\mathcal{D}, \eta^{\mathcal{D}}, \mu^{\mathcal{D}})$ *on* **Set** *is defined as follows. For objects* X *in* **Set**, $\mathcal{D}(X) = \{\Delta \mid \Delta \text{ is a finitely supported probability distribution on } X\}$*. For arrows* $f : X \to Y$ *in* **Set**, $\mathcal{D}(f) : \mathcal{D}(X) \to \mathcal{D}(Y)$ *is defined as*

[2] The proof structure of our Theorem 36 can be adapted (and in fact much simplified due to the simpler nature of $\mathtt{QTh}_{SL}$ and $\mathtt{QTh}_{CA}$ compared to $\mathtt{QTh}_{CS}$) to obtain these isomorphisms of categories. The recent result [7, Thm 4.2] might also provide an alternative proof method.

$\mathcal{D}(f)(\Delta) = \big(y \mapsto \sum_{x \in f^{-1}(y)} \Delta(x)\big)$. *The unit* $\eta^{\mathcal{D}}_X : X \to \mathcal{D}(X)$ *is defined as* $\eta_X(x) = \delta(x)$. *The multiplication* $\mu^{\mathcal{D}}_X : \mathcal{DD}(X) \to \mathcal{D}(X)$ *is defined, for* $\sum_{i=1}^n p_i\Delta_i \in \mathcal{DD}(X)$, *as* $\mu^{\mathcal{D}}_X(\sum_{i=1}^n p_i\Delta_i) = \big(x \mapsto \sum_{i=1}^n p_i \cdot \Delta_i(x)\big)$.

▶ Remark 4. Given elements $\Delta_1, \ldots, \Delta_n \in \mathcal{D}(X)$, the expression $\sum_{i=1}^n p_i\Delta_i$ denotes an element in $\mathcal{DD}(X)$. The set $\mathcal{D}(X)$ can be seen as a convex subset of the real vector space $\mathbb{R}^X$, so in order to avoid confusion with the notation $\sum_{i=1}^n p_i\Delta_i$ we will use the following dot–notation $\sum_{i=1}^n p_i \cdot \Delta_i$ to denote convex combinations of distributions: $\sum_{i=1}^n p_i \cdot \Delta_i = \mu^{\mathcal{D}}_X(\sum_{i=1}^n p_i\Delta_i) = \big(x \mapsto \sum_{i=1}^n p_i \cdot \Delta_i(x)\big)$. Hence, $\sum_{i=1}^n p_i\Delta_i$ denotes an element of $\mathcal{DD}(X)$ (a distribution of distributions), while $\sum_{i=1}^n p_i \cdot \Delta_i$ denotes an element of $\mathcal{D}(X)$.

Given a collection $S \subseteq \mathcal{D}(X)$ of distributions, we can construct its *convex closure* $cc(S) = \{\sum_{i=1}^n p_i \cdot \Delta_i \mid n \geq 1, \Delta_i \in S \text{ for all } i, \text{ and } \sum_{i=1}^n p_i = 1\}$. Note that $cc(cc(S)) = cc(S)$. A subset $S \subseteq \mathcal{D}(X)$ is *convex* if $S = cc(S)$. We say that a convex set $S \subseteq \mathcal{D}(X)$ is *finitely generated* if there exists a finite set $S' \subseteq \mathcal{D}(X)$ (i.e., $S' \in \mathcal{PD}(X)$) such that $S = cc(S')$. Given a finitely generated convex set $S \subseteq \mathcal{D}(X)$, there exists one minimal (with respect to the inclusion order) finite set $\mathtt{UB}(S) \in \mathcal{PD}(X)$ such that $S = cc(\mathtt{UB}(S))$. The finite set $\mathtt{UB}(S)$ is referred to as the *unique base* of S (see, e.g., [14]). The distributions in $\mathtt{UB}(S)$ are convex–linear independent, i.e., if $\mathtt{UB}(S) = \{\Delta_1, \ldots, \Delta_n\}$, then for all i, $\Delta_i \notin cc(\{\Delta_j \mid j \neq i\})$.

▶ **Definition 5.** *The* finitely generated non-empty convex powerset of distributions *monad* $(\mathcal{C}, \eta^{\mathcal{C}}, \mu^{\mathcal{C}})$ *on* **Set** *is defined as follows. Given an object* X *in* **Set**, $\mathcal{C}(X)$ *is the collection of non-empty finitely generated convex sets of finitely supported probability distributions on* X, *i.e.,* $\mathcal{C}(X) = \{cc(S) \mid S \in \mathcal{PD}(X)\}$. *Given an arrow* $f : X \to Y$ *in* **Set**, *the arrow* $\mathcal{C}(f) : \mathcal{C}(X) \to \mathcal{C}(Y)$ *is defined as* $\mathcal{C}(f)(S) = \{\mathcal{D}(f)(\Delta) \mid \Delta \in S\}$. *The unit* $\eta^{\mathcal{C}}_X : X \to \mathcal{C}(X)$ *is defined as* $\eta^{\mathcal{C}}_X(x) = \{\delta(x)\}$, *the singleton (convex) set consisting of the Dirac distribution. The mutiplication* $\mu^{\mathcal{C}}_X : \mathcal{CC}(X) \to \mathcal{C}(X)$ *is defined, for any* $S \in \mathcal{CC}(X)$, *as* $\mu^{\mathcal{C}}_X(S) = \bigcup_{\Delta \in S} \mathtt{WMS}(\Delta)$, *where, for any* $\Delta \in \mathcal{DC}(X)$ *of the form* $\sum_{i=1}^n p_i S_i$, *with* $S_i \in \mathcal{C}(X)$, *the* weighted Minkowski sum *operation* $\mathtt{WMS} : \mathcal{DC}(X) \to \mathcal{C}(X)$ *is defined as* $\mathtt{WMS}(\Delta) = \{\sum_{i=1}^n p_i \cdot \Delta_i \mid \text{for each } 1 \leq i \leq n, \Delta_i \in S_i\}$.

2.1 Equational Theories and Monad Presentations

An important concept regarding monads is that of algebras for a monad.

▶ **Definition 6.** *Let* $(\mathcal{M} : \mathbf{C} \to \mathbf{C}, \eta, \mu)$ *be a monad. An algebra for* $\mathcal{M}$ *is a pair* (A, h) *where* $A \in \mathbf{C}$ *is an object and* $h : \mathcal{M}(A) \to A$ *is a morphism such that:* $h \circ \eta_A = id_A$ *and* $h \circ \mathcal{M}h = h \circ \mu_A$. *Given two* $\mathcal{M}$*–algebras* (A, h) *and* (A', h'), *a* $\mathcal{M}$–algebra morphism *is an arrow* $f : A \to A'$ *in* $\mathbf{C}$ *such that* $f \circ h = h' \circ \mathcal{M}(f)$. *The category of Eilenberg-Moore algebras for* $\mathcal{M}$, *denoted by* $\mathbf{EM}(\mathcal{M})$, *has* $\mathcal{M}$*–algebras as objects and* $\mathcal{M}$*–morphisms as arrows.*

The definitions above are purely categorical and, as a consequence, the category $\mathbf{EM}(\mathcal{M})$ is sometimes hard to work with as an abstract entity. It is therefore very useful when $\mathbf{EM}(\mathcal{M})$ can be proven isomorphic to a category whose objects and morphisms are well–known and understood. This leads to the concept of *presentation of a monad*. Before introducing it, we recall some basic definitions of universal algebra (see [17] for a standard introduction).

▶ **Definition 7.** *A signature* Σ *is a set of function symbols each having its own finite arity. We denote with* $\mathcal{T}(X, \Sigma)$ *the set of terms built from a set of generators* X *with the function symbols of* Σ. *An equational theory* $\mathtt{Th}$ *of type* Σ *is a set* $\mathtt{Th} \subseteq \mathcal{T}(X, \Sigma) \times \mathcal{T}(X, \Sigma)$ *of equations between terms* $\mathcal{T}(X, \Sigma)$ *closed under deducibility in the logical apparatus of equational logic. Given a set* $E \subseteq \mathcal{T}(X, \Sigma) \times \mathcal{T}(X, \Sigma)$ *of equations, the theory induced by* E *is the smallest*

equational theory containing E. The models of a theory $\mathtt{Th}$ *are* Σ–algebras of the theory $\mathtt{Th}$, *i.e., structures $(A, \{f^A\}_{f\in\Sigma})$ consisting of a set A and operations $f^A : A^{ar(f)} \to A$, for each operation symbol $f \in \Sigma$ having arity $ar(f)$, satisfying all (universally quantified) equations in* $\mathtt{Th}$. *A homomorphism from $(A, \{f^A\}_{f\in\Sigma})$ to $(B, \{f^B\}_{f\in\Sigma})$ is a function $g : A \to B$ such that $g(f^A(a_1, \ldots, a_n)) = f^B(g(a_1), \ldots, g(a_n))$, for all $f \in \Sigma$. We denote with $\mathbf{A}(\mathtt{Th})$ the category whose objects are models of the theory* $\mathtt{Th}$ *and morphisms are homomorphisms.*

▶ **Definition 8** (Presentation of **Set** monads)**.** *Let $\mathcal{M}$ be a monad on* **Set**. *A presentation of $\mathcal{M}$ is an equational theory* $\mathtt{Th}$ *such that the categories $\mathbf{EM}(\mathcal{M})$ and $\mathbf{A}(\mathtt{Th})$ are isomorphic.*

In what follows we introduce equational theories that are presentations of the three **Set** monads $\mathcal{P}$, $\mathcal{D}$ and $\mathcal{C}$ introduced earlier.

▶ **Definition 9.** *The theory $\mathtt{Th}_{SL}$ of semilattices is the theory having as signature $\Sigma_{SL} = \{\oplus\}$ and equations stating that $\oplus$ is associative, commutative, and idempotent:*

(A) $(x \oplus y) \oplus z = x \oplus (y \oplus z)$ *(C)* $x \oplus y = y \oplus x$ *(I)* $x \oplus x = x$.

▶ **Definition 10.** *The theory $\mathtt{Th}_{CA}$ of convex algebras has signature $\Sigma_{CA} = \{+_p\}_{p\in(0,1)}$ and, for all $p, q \in (0, 1)$, the equations for probabilistic associativity, commutativity, and idempotency:*

(A_p) $(x +_q y) +_p z = x +_{pq} (y +_{\frac{p(1-q)}{1-pq}} z)$ *(C_p)* $x +_p y = y +_{1-p} x$ *(I_p)* $x +_p x = x$.

▶ **Definition 11.** *The theory $\mathtt{Th}_{CS}$ of convex semilattices is the theory with signature $\Sigma_{CS} = (\{\oplus\} \cup \{+_p\}_{p\in(0,1)})$ where $\oplus$ satisfies the equations of semilattices, $+_p$ satisfies the equations of convex algebras for every $p \in (0, 1)$, and, furthermore, for every $p \in (0, 1)$ the following distributivity equation (D) is satisfied: $x +_p (y \oplus z) = (x +_p y) \oplus (x +_p z)$.*

The following proposition collects known results in the literature (see [48, 24, 32, 13]).

▶ **Proposition 12.**

1. *The theory $\mathtt{Th}_{SL}$ of semilattices is a presentation of $\mathcal{P}$, i.e., $\mathbf{A}(\mathtt{Th}_{SL}) \cong \mathbf{EM}(\mathcal{P})$.*
2. *The theory $\mathtt{Th}_{CA}$ of convex algebras is a presentation of $\mathcal{D}$, i.e., $\mathbf{A}(\mathtt{Th}_{CA}) \cong \mathbf{EM}(\mathcal{D})$.*
3. *The theory $\mathtt{Th}_{CS}$ of convex semilattices is a presentation of $\mathcal{C}$, i.e., $\mathbf{A}(\mathtt{Th}_{CS}) \cong \mathbf{EM}(\mathcal{C})$.*

2.1.1 One Application: Representation of Term Algebras

Having presentations of **Set** monads as categories of algebras of equational theories is mathematically convenient for several reasons. One useful application, especially in the field of program semantics, are representation theorems for free algebras, which are, up to isomorphism, term algebras.

In this section we assume the reader to be familiar with the concept of free object in a category (see, e.g., [3, §10.3]). The free object generated by X in the category $\mathbf{EM}(\mathcal{M})$ is the $\mathcal{M}$–algebra $(\mathcal{M}(X), \mu^{\mathcal{M}}_X)$. The free object generated by X in the category $\mathbf{A}(\mathtt{Th})$ is the term algebra, i.e., the algebra whose carrier is $\mathcal{T}(X, \Sigma)_{/\mathtt{Th}}$, the set of Σ–terms constructed from the set of generators X taken modulo the equations of the theory $\mathtt{Th}$, and with operations defined on equivalences classes, that is, $f([t_1]_{/\mathtt{Th}}, \ldots, [t_n]_{/\mathtt{Th}}) = [f(t_1, \ldots, t_n)]_{/\mathtt{Th}}$ for each $f \in \Sigma$. These characterisations, together with the fact that free objects are unique up to isomorphism, can be used to derive the following result.

▶ **Proposition 13.** *Let $\mathcal{M}$ be a monad on* **Set** *and let $F : \mathbf{A}(\mathtt{Th}) \cong \mathbf{EM}(\mathcal{M})$ be a presentation of $\mathcal{M}$ in terms of the equational theory* $\mathtt{Th}$ *of type Σ. Then the term algebra $\mathcal{T}(X, \Sigma)_{/\mathtt{Th}}$ and the free Eilenberg-Moore algebra $(\mathcal{M}(X), \mu^{\mathcal{M}}_X)$ are isomorphic (via F).*

In other words, a presentation theorem for $\mathcal{M}$ provides automatically representation results for term algebras via the known semantic behaviour of the multiplication of $\mathcal{M}$.

▶ **Example 14.** The presentation of the monad $\mathcal{C}$ in terms of the theory of convex semilattices implies that the free convex semilattice generated by X is isomorphic with the convex semilattice $(\mathcal{C}X, \oplus, +_p)$ where $S_1 \oplus S_2 = cc(S_1 \cup S_2)$ (convex union) and $S_1 +_p S_2 = \mathtt{WMS}(pS_1 + (1-p)S_2)$ (weighted Minkowski sum), for all $S_1, S_2 \in \mathcal{C}(X)$. In other words, the set $\mathcal{T}(X, \Sigma_{CS})_{/\mathtt{Th}_{CS}}$ of convex semilattice terms modulo the equational theory of convex semilattices can be identified with the set $\mathcal{C}(X)$ of finitely generated convex sets of finitely supported probability distributions on X. The isomorphism is explicitly given in [14] by the function $\kappa : \mathcal{C}(X) \to \mathcal{T}(X, \Sigma_{CS})_{/\mathtt{Th}_{CS}}$ defined as $\kappa(S) = [\bigoplus_{\Delta \in \mathtt{UB}(S)} (+_{x \in supp(\Delta)} \Delta(x)\, x)]_{/\mathtt{Th}_{CS}}$, where $\bigoplus_{i \in I} x_i$ and $+_{i \in I} p_i\, x$ are respectively notations for the binary operations $\oplus$ and $+_p$ extended to operations of arity I, for I finite (see, e.g., [47, 12]). We remark that the equation $x \oplus y = x \oplus y \oplus (x +_p y)$, which explicitly expresses closure under taking convex combinations, is derivable from the theory of convex semilattices (see, e.g., [14, Lemma 14]), and that this derivation critically uses the distributivity axiom (D).

3 Monads on Met and Quantitative Equational Theories

In Section 2 we have considered monads in the category **Set**. We now shift our focus to monads in the category **EMet** of extended metric spaces and non–expansive functions. The category **EMet** provides a natural mathematical setting for developing the semantics of programs exhibiting quantitative behaviour such as, e.g., probabilistic choice. It is indeed appropriate in this setting to replace the usual notion of program equivalence with the more informative notion of program distance (see, e.g., [45, 27, 15, 23, 16]).

▶ **Definition 15.** *An extended metric space is a pair (X, d) such that X is a set and $d : X \times X \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ is a function, called the* metric*, satisfying the following properties: $d(x, y) = 0$ if and only if $x = y$, $d(x, y) = d(y, x)$, and $d(x, y) \leq d(x, z) + d(z, y)$, for all $x, y, z \in X$. A function $f : X \to Y$ between two extended metric spaces (X, d_X) and (Y, d_Y) is called non–expansive (a.k.a. 1–Lipschitz) if $d_Y(f(x_1), f(x_2)) \leq d_X(x_1, x_2)$ for all $x_1, x_2 \in X$. We denote with* **EMet** *the category whose objects are extended metric spaces and whose morphisms are non–expansive maps.*

Since we only work with extended metric spaces, in the rest of this paper we will systematically omit the adjective "extended". Given two metrics d_1, d_2 on X, we write $d_1 \sqsubseteq d_2$ if for all $x, x' \in X$, it holds that $d_1(x, x') \leq d_2(x, x')$. Let (Y, d) be a metric space, X a set and $f : X \to Y$. We write $d\langle f, f\rangle$ for the metric on X defined as $d\langle f, f\rangle(x_1, x_2) = d(f(x_1), f(x_2))$. Let $d_{\mathbb{R}}$ be the Euclidean metric on $\mathbb{R}$ defined as $d_{\mathbb{R}}(r_1, r_2) = |r_1 - r_2|$. If (X, d) is a metric space, we simply say that $f : X \to [0, 1]$ is non–expansive to mean that $f : (X, d) \to ([0, 1], d_{\mathbb{R}})$ is non–expansive. The metric d of a metric space (X, d) induces a topology on X whose open sets are generated by the *open balls* of the form $B(x, \epsilon) = \{y \in X \mid d(x, y) < \epsilon\}$, for $x \in X$ and $\epsilon > 0$. A subset $Y \subseteq X$ is called *compact* if each of its open covers has a finite subcover. Every compact set Y is closed and bounded (i.e., the distance between elements in Y is bounded by some real number). The collection of non–empty compact subsets of a metric space (X, d) is denoted by $\mathtt{Comp}(X, d)$. Note that every finite subset of X belongs to $\mathtt{Comp}(X, d)$.

The **Set** monads $\mathcal{P}$ and $\mathcal{D}$ defined in Section 2 can be extended to monads in **EMet**. These extensions are well–known and are based on metric liftings constructions due to Hausdorff and Kantorovich (see [34] for a standard reference).

▶ **Definition 16** (Hausdorff Lifting). *Let (X,d) be a metric space. The Hausdorff lifting of d is a metric $H(d)$ on $\mathtt{Comp}(X,d)$, the collection of non–empty compact subsets of X, defined as follows for any pair $X_1, X_2 \in \mathtt{Comp}(X,d)$:*

$$H(d)\big(X_1,X_2\big) = \max\Big\{ \sup_{x_1\in X_1} \inf_{x_2\in X_2} d(x_1,x_2) \quad , \quad \sup_{x_2\in X_2} \inf_{x_1\in X_1} d(x_1,x_2)\Big\}.$$

This leads to the well–known *hyperspace* monad $\mathcal{V}$ on **EMet** ([31], see also [34]).[3]

▶ **Definition 17.** *The* hyperspace *monad $(\mathcal{V},\eta^{\mathcal{V}},\mu^{\mathcal{V}})$ on* **EMet** *is defined as follows. Given an object (X,d) in* **EMet**, *$\mathcal{V}(X,d) = \big(\mathtt{Comp}(X,d), H(d)\big)$, the metric space of non–empty compact subsets of X equipped with the Hausdorff distance. Given a non–expansive map $f : (X,d_X) \to (Y,d_Y)$, $\mathcal{V}(f)(X') = \bigcup_{x\in X'} f(x)$. The unit $\eta^{\mathcal{V}}_{(X,d)} : (X,d) \to \mathcal{V}(X,d)$ is defined as $\eta^{\mathcal{V}}_{(X,d)}(x) = \{x\}$, and the multiplication $\mu^{\mathcal{V}}_{(X,d)} : \mathcal{V}\mathcal{V}(X,d) \to \mathcal{V}(X,d)$ is defined as $\mu^{\mathcal{V}}_{(X,d)}(\{X_i\}_{i\in I}) = \bigcup_i X_i$.*

The restriction of the monad $\mathcal{V}$ to finite (hence compact) subsets leads to the non–empty finite powerset monad on **EMet**, which we denote with $\hat{\mathcal{P}}$ to distinguish it from the **Set** monad $\mathcal{P}$.

▶ **Definition 18.** *The* non–empty finite powerset *monad $(\hat{\mathcal{P}},\eta^{\hat{\mathcal{P}}},\mu^{\hat{\mathcal{P}}})$ on* **EMet** *is defined as follows. Given an object (X,d) in* **EMet**, *$\hat{\mathcal{P}}(X,d) = \big(\mathcal{P}(X), H(d)\big)$, the collection of finite non–empty subsets of X equipped with the Hausdorff distance. The action of $\hat{\mathcal{P}}$ on morphisms, the unit $\eta^{\hat{\mathcal{P}}}$ and the multiplication $\mu^{\hat{\mathcal{P}}}$ are defined as for the* **Set** *monad $\mathcal{P}$ (or, equivalently, as for the $\mathcal{V}$ monad on* **EMet** *restricted to finite sets).*

Next, we introduce the Kantorovich lifting on finitely supported distributions [34].

▶ **Definition 19** (Kantorovich Lifting). *Let (X,d) be a metric space. The Kantorovich lifting of d is a metric $K(d)$ on $\mathcal{D}(X)$, the collection of finitely supported probability distributions on X, defined as follows for any pair $\Delta_1, \Delta_2 \in \mathcal{D}(X)$:*

$$K(d)(\Delta_1,\Delta_2) = \inf_{\omega\in Coup(\Delta_1,\Delta_2)} \Big(\sum_{(x_1,x_2)\in X\times X} \omega(x_1,x_2)\cdot d(x_1,x_2)\Big)$$

where $Coup(\Delta_1,\Delta_2)$ is defined as the collection of couplings of Δ_1 and Δ_2, i.e., the collection of probability distributions on the product space $X\times X$ such that the marginals of ω are Δ_1 and Δ_2. Formally, $Coup(\Delta_1,\Delta_2) = \{\omega \in \mathcal{D}(X\times X) \mid \mathcal{D}(\pi_1)(\omega) = \Delta_1 \text{ and } \mathcal{D}(\pi_2)(\omega) = \Delta_2\}$ where $\pi_1 : X_1\times X_2 \to X_1$ and $\pi_2 : X_1\times X_2 \to X_2$ are the projection functions.

We can now introduce the following version of the finitely supported probability distribution monad on **EMet**, which we denote with $\hat{\mathcal{D}}$ to distinguish it from the **Set** monad $\mathcal{D}$.

▶ **Definition 20.** *The* finitely supported probability distribution *monad $(\hat{\mathcal{D}},\eta^{\hat{\mathcal{D}}},\mu^{\hat{\mathcal{D}}})$ on* **EMet** *is defined as follows. Given an object (X,d) in* **EMet**, *$\hat{\mathcal{D}}(X,d) = \big(\mathcal{D}(X), K(d)\big)$, the collection of finitely supported probability distributions on X equipped with the Kantorovich distance. The action of $\hat{\mathcal{D}}$ on morphisms, the unit $\eta^{\hat{\mathcal{D}}}$, and the multiplication $\mu^{\hat{\mathcal{D}}}$ are defined as for the* **Set** *monad $\mathcal{D}$.*

The fact that the above definitions are correct (i.e., that $\hat{\mathcal{D}}$ is a functor and that $\eta^{\hat{\mathcal{D}}}$ and $\mu^{\hat{\mathcal{D}}}$ are non–expansive and satisfy the monad laws) is well–known (see, e.g., [34, 15, 8]).

[3] This monad, defined on the category **Met** of ordinary (i.e., non–extended) metric spaces, is essentially due to Hausdorff [31]. See, e.g., [34] for a detailed exposition.

3.1 Quantitative Equational Theories and Quantitative Algebras

We provide here the essential definitions and results of the framework developed by Mardare, Panangaden, and Plotkin in [36] (see also [7, 37, 5, 38]). In what follows, a signature Σ is fixed. Recall that $\mathcal{T}(X,\Sigma)$ denotes the set of terms constructed from X using the function symbols in Σ. A substitution is a map of type $\sigma : X \to \mathcal{T}(X,\Sigma)$. As usual, to any interpretation $\iota : X \to A$ of the variables into a set corresponds, by homomorphic extension, a unique map $\iota : \mathcal{T}(X,\Sigma) \to A$.

▶ **Definition 21** (Quantitative Equational Theory). *A* quantitative equation *is an expression of the form $t =_\epsilon s$, where $t, s \in \mathcal{T}(X,\Sigma)$ and $\epsilon \in \mathbb{R}_{\geq 0}$. We denote with $E(\Sigma)$ the collection of all quantitative equations. We use the letters Γ, Θ to range over subsets of $E(\Sigma)$. A* quantitative inference *is an element of $2^{E(\Sigma)} \times E(\Sigma)$, i.e., a pair $(\Gamma, t =_\epsilon s)$ where $\Gamma \subseteq E(\Sigma)$ and $t =_\epsilon s$ is a quantitative equation. Note that Γ needs not be finite. A* deducibility relation *is a set of quantitative inferences $\vdash \subseteq 2^{E(\Sigma)} \times E(\Sigma)$ closed under the following conditions which are stated for arbitrary $s,t,u \in \mathcal{T}(X,\Sigma)$, $\epsilon, \epsilon' \in \mathbb{R}_{\geq 0}$, $\Gamma, \Theta \subseteq E(\Sigma)$, and $f \in \Sigma$:*
(Notation: we use the infix notation $\Gamma \vdash t =_\epsilon s$ to mean that $(\Gamma, t =_\epsilon s) \in \vdash$)

(Refl) $\emptyset \vdash t =_0 t$ *(Symm)* $\{t =_\epsilon s\} \vdash s =_\epsilon t$ *(Triang)* $\{t =_\epsilon u, u =_{\epsilon'} s\} \vdash t =_{\epsilon+\epsilon'} s$

(Max) $\{t =_\epsilon s\} \vdash t =_{\epsilon'} s$, *where* $\epsilon' > \epsilon$ *(Arch)* $\{t =_{\epsilon'} s\}_{\epsilon' > \epsilon} \vdash t =_\epsilon s$

(NExp) $\{t_i =_\epsilon s_i\}_{i \in 1 \dots ar(f)} \vdash f(t_1, \dots, t_n) =_\epsilon f(s_1, \dots s_n)$

(Subst) if $\Gamma \vdash t =_\epsilon s$ *then* $\{\sigma(t) =_\epsilon \sigma(s) \mid (t =_\epsilon s) \in \Gamma\} \vdash \sigma(t) =_\epsilon \sigma(s)$, *for all substitutions* σ

(Cut) if $\Gamma \vdash \Theta$ *and* $\Theta \vdash t =_\epsilon s$ *then* $\Gamma \vdash t =_\epsilon s$

(Assum) if $t =_\epsilon s \in \Gamma$ *then* $\Gamma \vdash t =_\epsilon s$, *for all* Γ, t, s, ϵ

where in (Cut) the expression $\Gamma \vdash \Theta$ means that for all $(t =_\epsilon s) \in \Theta$ it holds that $\Gamma \vdash t =_\epsilon s$. Given a set of quantitative inferences $\mathcal{U} \subseteq 2^{E(\Sigma)} \times E(\Sigma)$, the quantitative equational theory induced by $\mathcal{U}$ is the smallest deducibility relation which includes $\mathcal{U}$.

The models of quantitative theories are quantitative algebras, which we now introduce.

▶ **Definition 22** (Quantitative Algebra). *A* quantitative algebra *of type Σ is a structure $\mathbb{A} = \big(A, \{f^A\}_{f \in \Sigma}, d_A\big)$ where (A, d_A) is an extended metric space and, for each $f \in \Sigma$, the function $f^A : A^{ar(f)} \to A$ is a non–expansive map, with $A^{ar(f)}$ endowed with the* sup*–metric defined as $d_{\sup}(\{a_i\}_{i \in ar(f)}, \{b_i\}_{i \in ar(f)}) = \max_{i \in ar(f)}(d(a_i, b_i))$. A homomorphism between quantitative algebras $\mathbb{A}$ and $\mathbb{B}$ of type Σ is a non–expansive function $g : (A, d_A) \to (B, d_B)$ which preserves all operations in Σ, i.e., $g(f^A(x_1, \dots, x_n)) = f^B(g(x_1), \dots, g(x_n))$, for all $x_i \in A$. We say that $\mathbb{A}$ satisfies a quantitative inference $(\{s_i =_{\epsilon_i} t_i\}_{i \in I}, s =_\epsilon t)$, written $\{s_i =_{\epsilon_i} t_i\} \models_{\mathbb{A}} s =_\epsilon t$, if for every interpretation $\iota : X \to A$ of the variables X into elements of A the following holds: if for all $i \in I$, $d_A\big(\iota(s_i), \iota(t_i)\big) \leq \epsilon_i$, then $d_A\big(\iota(s), \iota(t)\big) \leq \epsilon$. We say that $\mathbb{A}$ is a model of a quantitative theory* `QTh` *if $\mathbb{A}$ satisfies every quantitative inference in* `QTh`*. We denote with* $\mathbf{QA}$(`QTh`) *the category having as objects the quantitative algebras that are models of* `QTh`*, and as arrows the non–expansive homomorphisms between quantitative algebras of type Σ.*

Every quantitative algebra of type Σ satisfies the quantitative inferences generating the deducibility relation $\vdash$ in Definition 21. We refer to [36] for proofs that all the above definitions are indeed well–defined. Two interesting quantitative theories studied in [36] are the following.[4]

[4] We remark that, in [36], the quantitative theory of convex algebras is referred to as the quantitative theory of interpolative barycentric algebras.

▶ **Definition 23** (Quantitative Semilattices). *The quantitative theory of* quantitative semilattices*, denoted by* $\mathtt{QTh}_{SL}$*, has type* Σ_{SL} *(see Definition 9) and is induced by the following quantitative inferences, for all* $\epsilon_1, \epsilon_2 \in \mathbb{R}_{\geq 0}$*:*

(A) $\emptyset \vdash x \oplus (y \oplus z) =_0 (x \oplus y) \oplus z$ *(C)* $\emptyset \vdash x \oplus y =_0 y \oplus x$ *(I)* $\emptyset \vdash x \oplus x =_0 x$

(H) $\{x_1 =_{\epsilon_1} y_1, x_2 =_{\epsilon_2} y_2\} \vdash x_1 \oplus x_2 =_{\max(\epsilon_1,\epsilon_2)} y_1 \oplus y_2$.

▶ **Definition 24** (Quantitative Convex Algebras). *The quantitative theory of* quantitative convex algebras*, denoted by* $\mathtt{QTh}_{CA}$*, has type* Σ_{CA} *(see Definition 10) and is induced by the following quantitative inferences, for all* $p, q \in (0,1)$ *and* $\epsilon_1, \epsilon_2 \in \mathbb{R}_{\geq 0}$*:*

(A$_p$*)* $\emptyset \vdash (x +_q y) +_p z =_0 x +_{pq} (y +_{\frac{p(1-q)}{1-pq}} z)$ *(C*$_p$*)* $\emptyset \vdash x +_p y =_0 y +_{1-p} x$

(I$_p$*)* $\emptyset \vdash x +_p x =_0 x$ *(K)* $\{x_1 =_{\epsilon_1} y_1, x_2 =_{\epsilon_2} y_2\} \vdash x_1 +_p x_2 =_{p\cdot\epsilon_1+(1-p)\cdot\epsilon_2} y_1 +_p y_2$.

In other words, the theories $\mathtt{QTh}_{SL}$ and $\mathtt{QTh}_{CA}$ are obtained by taking the equational axioms of semilattices and convex algebras respectively (Definitions 9 and 10), replacing the equality $(=)$ with $(=_0)$, and by introducing the quantitative inferences (H) and (K) respectively.

A general result from [36, §5] states that free objects always exist in $\mathbf{QA}(\mathtt{QTh})$, for any $\mathtt{QTh}$, and they are isomorphic to term quantitative algebras for $\mathtt{QTh}$. Moreover, such free objects are concretely identified for two relevant theories:

▶ **Theorem 25** ([36, Cor 9.4 and 10.6]). *The following hold:*

- *The free quantitative semilattice in* $\mathbf{QA}(\mathtt{QTh}_{SL})$ *generated by a metric space* (X,d) *is isomorphic to the metric space* $\hat{\mathcal{P}}(X,d) = \big(\mathcal{P}(X), H(d)\big)$.
- *The free quantitative convex algebra in* $\mathbf{QA}(\mathtt{QTh}_{CA})$ *generated by a metric space* (X,d) *is isomorphic to the metric space* $\hat{\mathcal{D}}(X,d) = \big(\mathcal{D}(X), K(d)\big)$.

We remark that the above theorem from [36] falls short from a full presentation result stating the isomorphisms of categories $\mathbf{QA}(\mathtt{QTh}_{SL}) \cong \mathbf{EM}(\hat{\mathcal{P}})$ and $\mathbf{QA}(\mathtt{QTh}_{CA}) \cong \mathbf{EM}(\hat{\mathcal{D}})$. This latter more general statement does indeed hold and can be obtained, with some minor extra work, from the technical machinery developed in [36] (see Footnote 2).

4 The Monad $\hat{\mathcal{C}}$ on the Category of Metric Spaces

In this section we introduce a **EMet** version of the **Set** monad $\mathcal{C}$, and we denote it with $\hat{\mathcal{C}}$. The monad $\hat{\mathcal{C}}$ is obtained by composing the Hausdorff lifting H and the Kantorovich lifting K introduced in the previous section.

▶ **Proposition 26.** *Let* (X,d) *be a metric space and let* $S \in \mathtt{Comp}(\mathcal{D}(X), K(d))$*. Then* $cc(S) \in \mathtt{Comp}(\mathcal{D}(X), K(d))$*, i.e., the convex closure of* S *is also compact.*

▶ **Corollary 27.** *Let* (X,d) *be a metric space. If* $S \in \mathcal{C}(X)$ *then* $S \in \mathtt{Comp}(\mathcal{D}(X), K(d))$.

Corollary 27 implies that, given a metric space (X,d), the collection $\mathcal{C}(X)$ of finitely generated non–empty convex sets of distributions on X can be endowed with the subspace metric of $\mathcal{V}(\hat{\mathcal{D}}(X,d))$, and therefore $(\mathcal{C}(X), HK(d))$ is a metric space, with $HK(d) = H(K(d))$. This observation leads to the following definition.

▶ **Definition 28** (Monad $\hat{\mathcal{C}}$). *The* finitely generated non–empty convex powerset of finitely supported probability distributions *monad* $(\hat{\mathcal{C}}, \eta^{\hat{\mathcal{C}}}, \mu^{\hat{\mathcal{C}}})$ *on* **EMet** *is defined as follows. Given an object* (X,d) *in* **EMet**, $\hat{\mathcal{C}}(X,d) = \big(\mathcal{C}(X), HK(d)\big)$*. The action of* $\hat{\mathcal{C}}$ *on morphisms, the monad unit* $\eta^{\hat{\mathcal{C}}}$*, and the monad multiplication* $\mu^{\hat{\mathcal{C}}}$ *are defined as for the* **Set** *monad* $\mathcal{C}$ *(Definition 5).*

The rest of this section is devoted to the proof that the above definition is well–specified, i.e., that $\hat{\mathcal{C}}$ is indeed a monad on **EMet**. First, one needs to verify that $\hat{\mathcal{C}}$ is a functor on **EMet**. This follows immediately from the definition, Corollary 27, and $\mathcal{C}$ being a functor on **Set**. It then remains to verify that the unit $\eta^{\hat{\mathcal{C}}}$ and the multiplication $\mu^{\hat{\mathcal{C}}}$ of $\hat{\mathcal{C}}$ are indeed morphisms in **EMet** (i.e., they are non-expansive functions) and that they satisfy the monad laws of Definition 1. The fact that the laws are satisfied follows directly from the definitions $\mu^{\hat{\mathcal{C}}} = \mu^{\mathcal{C}}$ and $\eta^{\hat{\mathcal{C}}} = \eta^{\mathcal{C}}$ and the fact that $\mathcal{C}$ is a monad on **Set** (hence $\mu^{\mathcal{C}}$ and $\eta^{\mathcal{C}}$ satisfy the monad laws). Then it only remains to verify that $\eta^{\hat{\mathcal{C}}}$ and $\mu^{\hat{\mathcal{C}}}$ are non–expansive. It is straightforward to verify that $\eta^{\hat{\mathcal{C}}}$ is an isometric (hence non–expansive) embedding of (X, d) into $\big(\mathcal{C}(X), HK(d)\big)$. Proving that $\mu^{\hat{\mathcal{C}}}$ is non–expansive, instead, does not seem straightforward and requires some detailed calculations. We state this result as a theorem.

▶ **Theorem 29.** *Let (X, d) be a metric space in* **EMet**. *Then $\mu^{\hat{\mathcal{C}}}_{(X,d)} : \hat{\mathcal{C}}\hat{\mathcal{C}}(X, d) \to \hat{\mathcal{C}}(X, d)$ is a non–expansive function, i,e., using functional notation, $HK(d)\langle\mu^{\hat{\mathcal{C}}}, \mu^{\hat{\mathcal{C}}}\rangle \sqsubseteq HKHK(d)$.*

4.1 Sketch of the Proof of Theorem 29

The key result to prove is Lemma 32, stating that the weighted Minkowski sum function $\mathtt{WMS}$ is non–expansive. This is obtained by exploiting a key property of the HK metric (see Lemma 31) called convexity. It might well be that both these results have already appeared in the literature in some form or another or are known as folklore by specialists. We present here a direct proof.

▶ **Definition 30** (Convex metric). *Let $(X, \{+_p\}_{p\in(0,1)})$ be a convex algebra, i.e., a set X equipped with operations $+_p : X \times X \to X$ satisfying the axioms of Definition 10. Let $d : X \times X \to \mathbb{R}_{\geq 0} \cup \{\infty\}$ be a metric on X. We say that d is convex if $d(x_1 +_p x_2, y_1 +_p y_2) \leq d(x_1, y_1) +_p d(x_2, y_2)$ holds for all $x_1, x_2, y_1, y_2 \in X$ and $p \in (0, 1)$, where $d(x_1, y_1) +_p d(x_2, y_2) = p \cdot d(x_1, y_1) + (1 - p) \cdot d(x_2, y_2)$ with the convention that $\infty +_p x = x +_q \infty = \infty +_r \infty = \infty$ for all $p, q, r \in (0, 1)$ and $x \in X$.*

It is well known that the Kantorovich metric $K(d)$ is convex. The following lemma states that also the Hausdorff–Kantorovich metric $HK(d)$, defined on the collection $\mathcal{C}(X)$ of non–empty finitely generated convex sets of distributions, which carries the structure of a convex semilattice (see Example 14) and thus also of a convex algebra, is convex.

▶ **Lemma 31.** *Let (X, d) be a metric space. The metric $HK(d)$ on the convex algebra $(\mathcal{C}(X), \{+_p\}_{p\in(0,1)})$, with $S_1 +_p S_2 = \mathtt{WMS}(pS_1 + (1 - p)S_2)$, is convex.*

Using the convexity of HK it is possible to prove that the $\mathtt{WMS}$ function is non–expansive.

▶ **Lemma 32.** *Let (X, d) be a metric space. The function $\mathtt{WMS} : \hat{\mathcal{D}}(\hat{\mathcal{C}}(X, d)) \to \hat{\mathcal{C}}(X, d)$ (see Definition 5) is non–expansive, i.e. $HK(d)\langle\mathtt{WMS}, \mathtt{WMS}\rangle \sqsubseteq KHK(d)$.*

Lastly, we state the following two useful properties of the Hausdorff lifting.

▶ **Proposition 33.** *Let d, d' be two metrics over X such that $d \sqsubseteq d'$. Then $H(d) \sqsubseteq H(d')$.*

▶ **Proposition 34.** *Let (X, d_X) and (Y, d_Y) be metric spaces, let $f : X \to Y$ with $d_X = d_Y\langle f, f\rangle$ (i.e., $d_X(x_1, x_2) = d_Y(f(x_1), f(x_2))$). Then $H(d_X) = H(d_Y)\langle\mathcal{V}(f), \mathcal{V}(f)\rangle$.*

Proof of Theorem 29. We need to show that $HK(d)\langle\mu^{\hat{\mathcal{C}}}, \mu^{\hat{\mathcal{C}}}\rangle \sqsubseteq HKHK(d)$.

Since $\mathcal{V}$ is a monad on **EMet** (Definition 17), $\mu^{\mathcal{V}}$ is non-expansive, i.e., $H(d)\langle\mu^{\mathcal{V}}, \mu^{\mathcal{V}}\rangle \sqsubseteq HH(d)$. By applying this to the metric $K(d)$, we derive

$$HK(d)\langle\mu^{\mathcal{V}}, \mu^{\mathcal{V}}\rangle \sqsubseteq HHK(d). \tag{1}$$

By definition $\mu^{\hat{\mathcal{C}}} = \mu^{\mathcal{V}} \circ \mathcal{V}(\mathtt{WMS})$ (i.e., $S \mapsto \bigcup \{\mathtt{WMS}(\Delta) \mid \Delta \in S\}$) and therefore:

$$\begin{aligned} HK(d)\langle \mu^{\hat{\mathcal{C}}}, \mu^{\hat{\mathcal{C}}} \rangle &= HK(d)\langle \mu^{\mathcal{V}} \circ \mathcal{V}(\mathtt{WMS}), \mu^{\mathcal{V}} \circ \mathcal{V}(\mathtt{WMS}) \rangle \\ &= HK(d)\langle \mu^{\mathcal{V}}, \mu^{\mathcal{V}} \rangle \langle \mathcal{V}(\mathtt{WMS}), \mathcal{V}(\mathtt{WMS}) \rangle \end{aligned}$$

Thus, by (1) we can derive

$$HK(d)\langle \mu^{\hat{\mathcal{C}}}, \mu^{\hat{\mathcal{C}}} \rangle \sqsubseteq HHK(d)\langle \mathcal{V}(\mathtt{WMS}), \mathcal{V}(\mathtt{WMS}) \rangle. \tag{2}$$

Moreover, by the non-expansiveness of $\mathtt{WMS}$ (Lemma 32), we know that

$$HK(d)\langle \mathtt{WMS}, \mathtt{WMS} \rangle \sqsubseteq KHK(d)$$

which implies by the monotonicity of H (Proposition 33) that

$$H(HK(d)\langle \mathtt{WMS}, \mathtt{WMS} \rangle) \sqsubseteq HKHK(d). \tag{3}$$

By Proposition 34, we can rewrite the left-hand term of (3) as follows

$$H(HK(d)\langle \mathtt{WMS}, \mathtt{WMS} \rangle) = HHK(d)\langle \mathcal{V}(\mathtt{WMS}), \mathcal{V}(\mathtt{WMS}) \rangle$$

and thus we derive from (3):

$$HHK(d)\langle \mathcal{V}(\mathtt{WMS}), \mathcal{V}(\mathtt{WMS}) \rangle \sqsubseteq HKHK(d). \tag{4}$$

Lastly, by (2) and (4): $HK(d)\langle \mu^{\hat{\mathcal{C}}}, \mu^{\hat{\mathcal{C}}} \rangle \sqsubseteq HHK(d)\langle \mathcal{V}(\mathtt{WMS}), \mathcal{V}(\mathtt{WMS}) \rangle \sqsubseteq HKHK(d)$. ◀

5 Presentation of the Monad $\hat{\mathcal{C}}$

In this section we present the main result of this work and show that the monad $\hat{\mathcal{C}}$ on **EMet**, introduced in Section 4, is presented by quantitative convex semilattices.

▶ **Definition 35.** *The quantitative equational theory of* quantitative convex semilattices*, denoted by* $\mathtt{QTh}_{CS}$*, is the quantitative theory over the signature* $\Sigma_{CS} = (\{\oplus\} \cup \{+_p\}_{p\in(0,1)})$ *of convex semilattices induced by the following set of quantitative inferences:*

- *the quantitative inferences (A), (C), (I) and (H) inducing the quantitative theory of semilattices (see Definition 23),*
- *the quantitative inferences* (A_p)*,* (C_p)*,* (I_p)*, and (K) inducing the quantitative theory of convex algebras (see Definition 24),*
- *for every* $p \in (0,1)$*, the quantitative inference (D)* $\emptyset \vdash x +_p (y \oplus z) =_0 (x +_p y) \oplus (x +_p z)$*.*

The following is the main result of this work.

▶ **Theorem 36.** *The quantitative equational theory* $\mathtt{QTh}_{CS}$ *of quantitative convex semilattices is a presentation of the monad* $\hat{\mathcal{C}}$*, that is,* $\mathbf{QA}(\mathtt{QTh}_{CS}) \cong \mathbf{EM}(\hat{\mathcal{C}})$*.*

As one direct corollary of this general statement we automatically get the following result (cf. with Theorem 25) characterising free quantitative convex semilattices, which, by [36, §5], are in turn isomorphic to term quantitative algebras for $\mathtt{QTh}_{CS}$.

▶ **Corollary 37.** *The free quantitative algebra in* $\mathbf{QA}(\mathtt{QTh}_{CS})$ *generated by a metric space* (X, d) *is isomorphic to* $\hat{\mathcal{C}}(X, d)$*, the metric space of finitely generated convex sets of probability distributions metrized by the Hausdorff–Kantorovich metric* $HK(d)$*.*

We prove Theorem 36 by explicitly defining a pair of functors $\mathcal{F} : \mathbf{EM}(\hat{\mathcal{C}}) \to \mathbf{QA}(\mathtt{QTh}_{CS})$ and $\mathcal{G} : \mathbf{QA}(\mathtt{QTh}_{CS}) \to \mathbf{EM}(\hat{\mathcal{C}})$ and proving that they are isomorphisms of categories, i.e., that $\mathcal{G} \circ \mathcal{F} = id_{\mathbf{EM}(\hat{\mathcal{C}})}$ and $\mathcal{F} \circ \mathcal{G} = id_{\mathbf{QA}(\mathtt{QTh}_{CS})}$. In the following sections, we exhibit such functors and show that they are well-defined isomorphisms.

▶ Remark 38. A recent result (Theorem 4.2 of [7]), showing that, for any quantitative equational theory, the category of Eilenberg-Moore algebras of the term monad and the category $\mathbf{QA}(\mathtt{QTh}_{CS})$ are isomorphic, might provide an alternative route to obtain the result of Theorem 36. Our proof technique has the virtue of concretely exhibiting the functors witnessing the isomorphism.

5.1 The functor $\mathcal{F} : \mathbf{EM}(\hat{\mathcal{C}}) \to \mathbf{QA}(\mathtt{QTh}_{CS})$

Recall from Definition 6 that an object in $\mathbf{EM}(\hat{\mathcal{C}})$ is a structure $((X,d),\alpha)$ where (X,d) is a metric space and $\alpha : (\mathcal{C}(X), HK(d)) \to (X,d)$ is a non-expansive function satisfying $\alpha \circ \eta^{\hat{\mathcal{C}}}_X = id_X$ and $\alpha \circ \hat{\mathcal{C}}\alpha = \alpha \circ \mu^{\hat{\mathcal{C}}}_X$. A morphism $f : ((X,d_X),\alpha_X) \to ((Y,d_Y),\alpha_Y)$ in $\mathbf{EM}(\hat{\mathcal{C}})$ is a non–expansive function $f : X \to Y$ such that $f \circ \alpha_X = \alpha_Y \circ \hat{\mathcal{C}}(f)$.

▶ **Definition 39** (Functor $\mathcal{F}$). *We define $\mathcal{F} : \mathbf{EM}(\hat{\mathcal{C}}) \to \mathbf{QA}(\mathtt{QTh}_{CS})$ as follows:*

- *on objects: $\mathcal{F}((X,d),\alpha) = (X, \Sigma^\alpha_{CS}, d)$*
  with $\Sigma^\alpha_{CS} = (\{\oplus^\alpha\} \cup \{+^\alpha_p\}_{p\in(0,1)})$ the interpretation of the convex semilattice operations $\oplus$ and $+_p$ as $x_1 \oplus^\alpha x_2 = \alpha(cc\{\delta(x_1), \delta(x_2)\})$ and $x_1 +^\alpha_p x_2 = \alpha(\{px_1 + (1-p)x_2\})$,
- *on morphisms: $\mathcal{F}(f) = f$, with $f : X \to Y$ seen as a non–expansive map from X to Y.*

We now prove that the functor $\mathcal{F}$ is well-defined. First, on objects, we need to show that $\mathcal{F}((X,d),\alpha)$ is indeed a quantitative algebra satisfying the quantitative inferences of the theory $\mathtt{QTh}_{CS}$. To show that $(X, \Sigma^\alpha_{CS}, d)$ is a quantitative algebra (Definition 22), since (X,d) is a metric space, we only need to verify that the operations $\oplus^\alpha$ and $+^\alpha_p$ are non–expansive.

▶ **Lemma 40.** *The operations $\oplus^\alpha$ and $+^\alpha_p$, for all $p \in (0,1)$, are non–expansive.*

Proof. Using functional notation we have $\oplus^\alpha = \alpha \circ cc \circ \mathcal{P}\eta^{\mathcal{D}}_X \circ (\lambda x_1, x_2.\{x_1, x_2\})$. The function α is non–expansive by assumption. $\mathcal{P}\eta^{\mathcal{D}}_X$ is non-expansive by $\hat{\mathcal{P}}$ and $\hat{\mathcal{D}}$ being monads on **EMet**. The functions $\lambda x_1, x_2.\{x_1, x_2\} : (X,d) \times (X,d) \to \hat{\mathcal{P}}(X,d)$ and $cc : \hat{\mathcal{P}}\hat{\mathcal{D}}(X,d) \to \hat{\mathcal{C}}(X,d)$ are non–expansive as well. Hence $\oplus^\alpha$ is non–expansive as composition of non–expansive maps. Similarly, we have $+^\alpha_p = \alpha \circ \eta^{\mathcal{P}}_{\mathcal{D}(X)} \circ \big(\lambda x_1, x_2.(px_1 + (1-p)x_2)\big)$ and all operations involved are non–expansive. ◀

As $\mathcal{F}((X,d),\alpha)$ is a quantitative algebra, it satisfies all the quantitative inferences of Definition 21. It only remains to show that the quantitative inferences of the theory $\mathtt{QTh}_{CS}$ (Definition 35) are also satisfied. For each of the quantitative inferences (A, C, I, A_p, C_p, I_p, D), which are of the form $\emptyset \vdash s =_0 t$, we need to show that the equality $s = t$ holds (universally quantified) in $(X, \Sigma^\alpha_{CS}, d)$. This amounts to showing that the algebra (X, Σ^α_{CS}) (with the metric d forgotten) is a model of the equational theory of convex semilattices (Definition 11). This proof has no specific metric–theoretic content and is omitted here. Thus, it only remains to show that the quantitative inferences (H) and (K) are satisfied.

▶ **Lemma 41** (H). $\{x_1 =_{\epsilon_1} y_1, x_2 =_{\epsilon_2} y_2\} \models_{\mathcal{F}((X,d),\alpha)} x_1 \oplus x_2 =_{\max(\epsilon_1,\epsilon_2)} y_1 \oplus y_2$.

Proof. The quantitative inference (H) is equivalent (i.e., mutually derivable in presence of the others deductive rules of Definition 21) with the (NExp) deductive rule. This means that (H) holds in $\mathcal{F}((X,d),\alpha)$ because the operation $\oplus^\alpha$ is non–expansive (Lemma 40). ◀

▶ **Lemma 42** (K). $x_1 =_{\epsilon_1} y_1, x_2 =_{\epsilon_2} y_2 \models_{\mathcal{F}((X,d),\alpha)} x_1 +_p x_2 =_{p\cdot\epsilon_1+(1-p)\cdot\epsilon_2} y_1 +_p y_2$.

Proof. For arbitrary $x_1, x_2, y_1, y_2 \in X$, assume $d(x_1, y_1) \leq \epsilon_1$ and $d(x_2, y_2) \leq \epsilon_2$. Then

$$\begin{aligned} d(x_1 +_p^\alpha x_2, y_1 +_p^\alpha y_2) &= d(\alpha(\{px_1 + (1-p)x_2\}), \alpha(\{py_1 + (1-p)y_2\}) \\ &\leq HK(d)(\{px_1 + (1-p)x_2\}, \{py_1 + (1-p)y_2\}) \qquad (\alpha \text{ non-exp.}) \\ &= K(d)(px_1 + (1-p)x_2, py_1 + (1-p)y_2) \\ &\leq p \cdot d(x_1, y_1) + (1-p) \cdot d(x_2, y_2) \qquad (\text{the metric } K(d) \text{ is convex}) \\ &\leq p \cdot \epsilon_1 + (1-p) \cdot \epsilon_2 \end{aligned}$$

◀

Hence $\mathcal{F}$ is well–defined on objects. It remains to verify that $\mathcal{F}$ is well defined on morphisms. Let $f : ((X,d),\alpha) \to ((Y,d'),\beta)$ be a morphism in $\mathbf{EM}(\hat{\mathcal{C}})$. We need to verify that $\mathcal{F}(f)$ is a morphisms in $\mathbf{QA}(\mathtt{QTh}_{CS})$, i.e., a non–expansive homomorphism of convex semilattices (see Definition 22). Since by definition $\mathcal{F}(f) = f$, the function $\mathcal{F}(f)$ is non–expansive. It remains to verify that it is a homomorphism. This proof has no specific metric–theoretic content and we omit it here.

5.2 The functor $\mathcal{G} : \mathbf{QA}(\mathtt{QTh}_{CS}) \to \mathbf{EM}(\hat{\mathcal{C}})$

Recall that an object in $\mathbf{QA}(\mathtt{QTh}_{CS})$ is a quantitative convex semilattice $\mathbb{A} = (X, \Sigma_{CS}^{\mathbb{A}}, d)$, with $\Sigma_{CS}^{\mathbb{A}} = (\{\oplus^{\mathbb{A}}\} \cup \{+_p^{\mathbb{A}}\}_{p\in(0,1)})$. Also, recall from Example 14 that there is an isomorphism κ mapping elements of $\mathcal{C}(X)$ to equivalence classes of convex semilattice terms in $\mathcal{T}(X, \Sigma_{CS})_{/\mathtt{Th}_{CS}}$. Let us define $\nu : \mathcal{C}(X) \to \mathcal{T}(X, \Sigma_{CS})$ as a choice function, mapping each $S \in \mathcal{C}(X)$ to one representative of the equivalence class $\kappa(S)$. This allows us to uniquely write down each $S \in \mathcal{C}(X)$ as a convex semilattice term:

$$\nu(S) = \bigoplus_{\Delta\in\mathtt{UB}(S)} \left(\mathop{+}_{x\in supp(\Delta)} \Delta(x)\, x \right).$$

With abuse of notation, we have used the letter X to range both over a set of variables and the carrier of $\mathbb{A}$. By interpreting each variable x with the corresponding element $x \in X$ of $\mathbb{A}$, and by homomorphic extension, we get that each term $t \in \mathcal{T}(X, \Sigma_{CS})$ can be interpreted as an element $t^{\mathbb{A}}$ of $\mathbb{A}$, and in particular $(\nu(S))^{\mathbb{A}}$ denotes an element of $\mathbb{A}$ for each $S \in \mathcal{C}(X)$.

▶ **Definition 43** (Functor $\mathcal{G}$). *We specify $\mathcal{G} : \mathbf{QA}(\mathtt{QTh}_{CS}) \to \mathbf{EM}(\hat{\mathcal{C}})$ as follows:*

- *on objects $\mathbb{A} = (X, \Sigma_{CS}^{\mathbb{A}}, d)$, we define $\mathcal{G}(\mathbb{A}) = ((X,d),\alpha)$, with $\alpha : (\mathcal{C}(X), HK(d)) \to (X,d)$ defined as: $\alpha(S) = (\nu(S))^{\mathbb{A}}$,*
- *on morphisms (i.e., non-expansive homomorphisms) we define $\mathcal{G}(f) = f$.*

In order to prove that $\mathcal{G}$ is well-defined on objects, we have to show that indeed $((X,d),\alpha)$ is an Eilenberg-Moore algebra for $\hat{\mathcal{C}}$, which amounts to proving the following lemma.

▶ **Lemma 44.** *Let $\mathcal{G}(\mathbb{A}) = ((X,d),\alpha)$, for $\mathbb{A} = (X, \Sigma_{CS}^{\mathbb{A}}, d) \in \mathbf{QA}(\mathtt{QTh}_{CS})$.*

1. *(X,α) is an Eilenberg-Moore algebra for $\mathcal{C}$ in $\mathbf{Set}$, i.e., $\alpha \circ \eta^{\mathcal{C}} = id$ and $\alpha \circ \mathcal{C}\alpha = \alpha \circ \mu^{\mathcal{C}}$.*
2. *α is a morphism in $\mathbf{EMet}$, i.e., α is a non-expansive map: $d\langle\alpha,\alpha\rangle \sqsubseteq HK(d)$.*

Proof. The proof of the first point does not have any specific metric–theoretic content and is omitted here. For the second point, let $S, T \in \mathcal{C}(X)$. By the definition of α, we have $d(\alpha(S), \alpha(T)) = d((\nu(S))^{\mathbb{A}}, (\nu(T))^{\mathbb{A}})$. As stated in Lemma 45 below, it is possible to derive in $\mathtt{QTh}_{CS}$ the quantitative inference

$$\bigcup_{(\Delta,\Theta)\in\mathtt{UB}(S)\times\mathtt{UB}(T)} \left(\bigcup_{(x,y)\in supp(\Delta)\times supp(\Theta)} \{x =_{d(x,y)} y\} \right) \vdash \nu(S) =_{HK(d)(S,T)} \nu(T)$$

which, since $\mathbb{A}$ is a model of $\mathtt{QTh}_{CS}$, is thereby satisfied by $\mathbb{A}$. Since all the premises of the inference hold in $\mathbb{A}$, we conclude that $d((\nu(S))^{\mathbb{A}}, (\nu(T))^{\mathbb{A}}) \leq HK(d)(S,T)$ and, therefore, $d\langle\alpha,\alpha\rangle \sqsubseteq HK(d)$ holds, as desired. ◀

The following technical lemma is critically used in the proof of Lemma 44(2) above. Note that its statement is purely syntactic as it deals with derivability in the deductive apparatus of quantitative equational theories (Definition 21).

▶ **Lemma 45.** *Let (X,d) be a metric space and let $S,T \in \mathcal{C}(X)$. Then we have in $\mathtt{QTh}_{CS}$:*

$$\bigcup_{(\Delta,\Theta)\in \mathtt{UB}(S)\times\mathtt{UB}(T)} \left(\bigcup_{(x,y)\in supp(\Delta)\times supp(\Theta)} \{x =_{d(x,y)} y\} \right) \vdash \nu(S) =_{HK(d)(S,T)} \nu(T)$$

Proof Sketch. First, we derive the following useful quantitative inference dealing with the case of $S = \{\Delta\}$ and $T = \{\Theta\}$ being singletons, so that $HK(d)(S,T) = K(d)(\Delta,\Theta)$. Let (X,d) be a metric space and let $\Delta,\Theta \in \mathcal{D}(X)$. Then the following is derivable in $\mathtt{QTh}_{CS}$:

$$\bigcup_{(x,y)\in supp(\Delta)\times supp(\Theta)} \{x =_{d(x,y)} y\} \vdash \nu(\{\Delta\}) =_{K(d)(\Delta,\Theta)} \nu(\{\Theta\}).$$

To construct this derivation we take an optimal coupling ω of Δ and Θ (see Definition 19) witnessing the Kantorovich distance $K(d)(\Delta,\Theta)$ and then use the information provided by ω to construct a syntactic derivation where only the quantitative inferences (A_p, C_p, I_p and K) of the quantitative theory of convex algebras are used. The construction of this derivation follows analogously to the completeness result for quantitative convex algebras from [36].

Secondly, we calculate the $HK(d)(S,T)$ distance between S and T.

$$HK(d)(S,T) = \max\Big\{ \sup_{\Delta\in S}\inf_{\Theta\in T} K(d)(\Delta,\Theta) \;\;,\;\; \sup_{\Theta\in T}\inf_{\Delta\in S} K(d)(\Delta,\Theta) \Big\}.$$

By compactness arguments, the inf and sup are always attained. Hence this calculation involves distances $K(d)(\Delta_i,\Theta_j)$ between a finite number of elements $\Delta_i \in S$ and $\Theta_j \in T$, for $0 \leq i \leq n$ and $0 \leq j \leq m$. Since the equation $x \oplus y = x \oplus y \oplus (x +_p y)$ holds in all convex semilattices, we can derive in the theory of convex semilattices the equalities: $\nu(S) = \nu(S) \oplus \nu(\{\Delta_1\}) \oplus \cdots \oplus \nu(\{\Delta_n\})$ and $\nu(T) = \nu(T) \oplus \nu(\{\Theta_1\}) \oplus \cdots \oplus \nu(\{\Theta_m\})$. For each of the pairs (Δ_i,Θ_j) appearing in the expressions above we can derive, as described above, the quantitative equation $\nu(\{\Delta_i\}) =_{K(d)(\Delta_i,\Theta_j)} \nu(\{\Theta_j\})$. The calculation of $HK(d)(S,T)$ can then be mimicked syntactically to derive the quantitative equation $\nu(S) =_{HK(d)(S,T)} \nu(T)$ by only using the quantitative inferences (A, C, I and H) of quantitative semilattices. This follows analogously to the completeness result for quantitative semilattices from [36]. ◀

It remains to verify that the functor $\mathcal{G}$ is well-defined on morphisms. To see this, take $f : X \to Y$ a non-expansive homomorphism of quantitative algebras $\mathbb{A} = (X, \Sigma^{\mathbb{A}}_{CS}, d)$ and $\mathbb{B} = (Y, \Sigma^{\mathbb{B}}_{CS}, d')$ in $\mathbf{QA}(\mathtt{QTh}_{CS})$. Then f is an arrow in **EMet**, being non-expansive. We therefore only need to show that f is also a morphism of Eilenberg-Moore algebras (see Definition 6) i.e., that $f \circ \alpha = \beta \circ \hat{\mathcal{C}}(f)$. The verification of this equality involves no specific metric–theoretic considerations, and is therefore omitted.

5.3 The isomorphism

It remains to prove that the functors $\mathcal{F} : \mathbf{EM}(\hat{\mathcal{C}}) \to \mathbf{QA}(\mathtt{QTh}_{CS})$ and $\mathcal{G} : \mathbf{EM}(\hat{\mathcal{C}}) \to \mathbf{QA}(\mathtt{QTh}_{CS})$ define an isomorphism between the categories $\mathbf{EM}(\hat{\mathcal{C}})$ and $\mathbf{QA}(\mathtt{QTh}_{CS})$. This means proving that $\mathcal{G} \circ \mathcal{F} = id_{\mathbf{EM}(\hat{\mathcal{C}})}$ and $\mathcal{F} \circ \mathcal{G} = id_{\mathbf{QA}(\mathtt{QTh}_{CS})}$. On morphisms, by definition

we have $\mathcal{G} \circ \mathcal{F}(f) = f = \mathcal{F} \circ \mathcal{G}(f)$. Hence the identities trivially hold true. The proofs regarding the identities on objects require only routine verifications, unfolding definitions, not involving any specific metric–theoretic content and therefore we omit them here.

6 Conclusions

We have introduced the **EMet** monad $\hat{\mathcal{C}}$ of finitely generated non–empty convex sets of distributions equipped with the Hausdorff-Kantorovich distance, and we have proved that $\hat{\mathcal{C}}$ is presented by the quantitative equational theory $\mathtt{QTh}_{CS}$ of quantitative convex semilattices. This result provides the basis for a foundational understanding of equational reasoning about program distances in processes combining nondeterminism and probabilities, as in bisimulation and trace metrics [22, 25, 26, 49, 6, 18]. This opens several directions for future research.

For instance, one interesting line of research is to examine the axiomatizations of bisimulation equivalences and metrics for nondeterministic and probabilistic programs (or process algebras) that have been proposed in the literature [40, 9, 21, 1, 2, 20]. The quantitative equational framework of quantitative convex semilattices provides a novel tool for comparing and further developing the existing works.

It is also important to explore variants of the **EMet** monad $\hat{\mathcal{C}}$ such as, for instance, the one that also includes the empty set. These are needed to model program observations such as termination. Following the ideas presented in [13], these variants can be explored via the lift monad $(\cdot + 1)$ and its quotients described by equational theories over the signature of convex semilattices extended with a new constant symbol. A systematic study of these quotients is a promising direction for future work. Applications to up-to techniques for bisimulation metrics [19, 10] could then be pursued as well.

Lastly, it is natural to ask if the monad $\hat{\mathcal{C}}$, and its presentation, can be obtained as a general categorical composition of the hyperspace monad $\mathcal{V}$ and the distribution monad $\hat{\mathcal{D}}$. Recently, Goy and Petrisan [30] have used the notion of weak distributive law to provide a positive answer for the corresponding monads in the category **Set**. Investigating whether this machinery is also applicable to the category **EMet** is an interesting topic for future work.

References

1 Suzana Andova. Process algebra with probabilistic choice. In *Formal Methods for Real-Time and Probabilistic Systems, 5th International AMAST Workshop, ARTS'99, Bamberg, Germany, May 26-28, 1999. Proceedings*, pages 111–129, 1999. doi:10.1007/3-540-48778-6_7.

2 Suzana Andova and Sonja Georgievska. On compositionality, efficiency, and applicability of abstraction in probabilistic systems. In Mogens Nielsen, Antonín Kucera, Peter Bro Miltersen, Catuscia Palamidessi, Petr Tuma, and Frank D. Valencia, editors, *SOFSEM 2009: Theory and Practice of Computer Science, 35th Conference on Current Trends in Theory and Practice of Computer Science, Spindleruv Mlýn, Czech Republic, January 24-30, 2009. Proceedings*, volume 5404 of *Lecture Notes in Computer Science*, pages 67–78. Springer, 2009. doi:10.1007/978-3-540-95891-8_10.

3 Steve Awodey. *Category Theory*. Oxord University Press, 2010.

4 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. Complete axiomatization for the total variation distance of Markov chains. In Sam Staton, editor, *Proceedings of the Thirty-Fourth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2018, Dalhousie University, Halifax, Canada, June 6-9, 2018*, volume 341 of *Electronic Notes in Theoretical Computer Science*, pages 27–39. Elsevier, 2018. doi:10.1016/j.entcs.2018.03.014.

5 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. A complete quantitative deduction system for the bisimilarity distance on Markov chains. *Logical Methods in Computer Science*, 14(4), 2018. doi:10.23638/LMCS-14(4:15)2018.

6 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, Radu Mardare, Qiyi Tang, and Franck van Breugel. Computing probabilistic bisimilarity distances for probabilistic automata. In *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, pages 9:1–9:17, 2019. doi:10.4230/LIPIcs.CONCUR.2019.9.

7 Giorgio Bacci, Radu Mardare, Prakash Panangaden, and Gordon D. Plotkin. An algebraic theory of Markov Processes. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 679–688. ACM, 2018. doi:10.1145/3209108.3209177.

8 Paolo Baldan, Filippo Bonchi, Henning Kerstan, and Barbara König. Coalgebraic behavioral metrics. *Logical Methods in Computer Science*, 14(3), 2018. doi:10.23638/LMCS-14(3:20)2018.

9 E. Bandini and R. Segala. Axiomatizations for probabilistic bisimulation. In *Proc. of the 28th Int. Coll. on Automata, Languages and Programming (ICALP 2001)*, volume 2076 of *LNCS*, pages 370–381. Springer, 2001.

10 Filippo Bonchi, Barbara König, and Daniela Petrisan. Up-to techniques for behavioural metrics via fibrations. In *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, pages 17:1–17:17, 2018. doi:10.4230/LIPIcs.CONCUR.2018.17.

11 Filippo Bonchi, Daniela Petrisan, Damien Pous, and Jurriaan Rot. A general account of coinduction up-to. *Acta Inf.*, 54(2):127–190, 2017. doi:10.1007/s00236-016-0271-4.

12 Filippo Bonchi, Alexandra Silva, and Ana Sokolova. The Power of Convex Algebras. In *CONCUR 2017*, volume 85, pages 23:1–23:18. LIPIcs, 2017. doi:10.4230/LIPIcs.CONCUR.2017.23.

13 Filippo Bonchi, Ana Sokolova, and Valeria Vignudelli. The theory of traces for systems with nondeterminism and probability. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–14, 2019. doi:10.1109/LICS.2019.8785673.

14 Filippo Bonchi, Ana Sokolova, and Valeria Vignudelli. Presenting convex sets of probability distributions by convex semilattices and unique bases, 2020. arXiv:2005.01670.

15 Franck van Breugel. The metric monad for probabilistic nondeterminism. http://www.cse.yorku.ca/~franck/research/drafts/monad.pdf, 2005.

16 Franck van Breugel and James Worrell. Towards quantitative verification of probabilistic transition systems. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 421–432. Springer, 2001. doi:10.1007/3-540-48224-5_35.

17 Stanley Burris and H. P. Sankappanavar. *A Course in Universal Algebra*. Springer-Verlag Graduate Texts in Mathematics, 1981.

18 Valentina Castiglioni. Trace and testing metrics on nondeterministic probabilistic processes. In *Proc. Express/SOS 2018.*, pages 19–36, 2018. doi:10.4204/EPTCS.276.4.

19 Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Valeria Vignudelli. Up-to techniques for generalized bisimulation metrics. In *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, pages 35:1–35:14, 2016. doi:10.4230/LIPIcs.CONCUR.2016.35.

20 Pedro R. D'Argenio, Daniel Gebler, and Matias David Lee. Axiomatizing bisimulation equivalences and metrics from probabilistic SOS rules. In *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, pages 289–303, 2014. doi:10.1007/978-3-642-54830-7_19.

21 Y. Deng and C. Palamidessi. Axiomatizations for probabilistic finite-state behaviors. *Theoretical Computer Science*, 373:92–114, 2007.

22 Yuxin Deng, Tom Chothia, Catuscia Palamidessi, and Jun Pang. Metrics for action-labelled quantitative transition systems. *Electron. Notes Theor. Comput. Sci.*, 153(2):79–96, 2006. doi:10.1016/j.entcs.2005.10.033.

23 Josee Desharnais, Radha Jagadeesan, Vineet Gupta, and Prakash Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *Proc. LICS'02*, pages 413–422. IEEE Computer Society, 2002. doi:10.1109/LICS.2002.1029849.

24 E. Doberkat. Eilenberg-moore algebras for stochastic relations. *Information and Computation*, 204(12):1756–1781, 2006. Erratum and Addendum: Eilenberg-Moore algebras for stochastic relations. *Information and Computation*, Volume 206, Issue 12, December 2008, Pages 1476-1484.

25 Daniel Gebler, Kim G. Larsen, and Simone Tini. Compositional bisimulation metric reasoning with probabilistic process calculi. *Logical Methods in Computer Science*, 12(4), 2016. doi:10.2168/LMCS-12(4:12)2016.

26 Daniel Gebler and Simone Tini. SOS specifications for uniformly continuous operators. *J. Comput. Syst. Sci.*, 92:113–151, 2018. doi:10.1016/j.jcss.2017.09.011.

27 A. Giacalone, C.-C. Jou, and S.A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proc. PROCOMET'90)*, pages 443–458. North-Holland, 1990.

28 Jean Goubault-Larrecq. Continuous previsions. In Jacques Duparc and Thomas A. Henzinger, editors, *Computer Science Logic, 21st International Workshop, CSL 2007, 16th Annual Conference of the EACSL, Lausanne, Switzerland, September 11-15, 2007, Proceedings*, volume 4646 of *Lecture Notes in Computer Science*, pages 542–557. Springer, 2007. doi:10.1007/978-3-540-74915-8_40.

29 Jean Goubault-Larrecq. Prevision domains and convex powercones. In Roberto M. Amadio, editor, *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29 - April 6, 2008. Proceedings*, volume 4962 of *Lecture Notes in Computer Science*, pages 318–333. Springer, 2008. doi:10.1007/978-3-540-78499-9_23.

30 Alexandre Goy and Daniela Petrisan. Combining probabilistic and non-deterministic choice via weak distributive laws. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 454–464. ACM, 2020. doi:10.1145/3373718.3394795.

31 Felix Hausdorff. Grundzuge der mengenlehre. *German. Veit und Co. (cit. page 5)*, 1914.

32 B. Jacobs. Convexity, duality and effects. In *Theoretical computer science*, volume 323 of *IFIP Adv. Inf. Commun. Technol.*, pages 1–19. Springer, Berlin, 2010. doi:10.1007/978-3-642-15240-5_1.

33 Bart Jacobs. Coalgebraic trace semantics for combined possibilitistic and probabilistic systems. *Electr. Notes Theor. Comput. Sci.*, 203(5):131–152, 2008.

34 Alexander Kechris. *Classical Descriptive Set Theory*. Springer-Verlag, 1995.

35 Bartek Klin. Bialgebras for structural operational semantics: An introduction. *Theor. Comput. Sci.*, 412(38):5043–5069, 2011. doi:10.1016/j.tcs.2011.03.023.

36 Radu Mardare, Prakash Panangaden, and Gordon D. Plotkin. Quantitative algebraic reasoning. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 700–709. ACM, 2016. doi:10.1145/2933575.2934518.

37 Radu Mardare, Prakash Panangaden, and Gordon D. Plotkin. On the axiomatizability of quantitative algebras. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005102.

38 Radu Mardare, Prakash Panangaden, and Gordon D. Plotkin. Free complete Wasserstein algebras. *Logical Methods in Computer Science*, 14(3), 2018. doi:10.23638/LMCS-14(3:19)2018.

39 Matteo Mio. Upper-expectation bisimilarity and łukasiewicz μ-calculus. In Anca Muscholl, editor, *Foundations of Software Science and Computation Structures - 17th International Conference, FOSSACS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8412 of *Lecture Notes in Computer Science*, pages 335–350. Springer, 2014. doi:10.1007/978-3-642-54830-7_22.

40 M. Mislove, J. Ouaknine, and J. Worrell. Axioms for probability and nondeterminism. In *Proc. of the 10th Int. Workshop on Expressiveness in Concurrency (EXPRESS 2003)*, volume 96 of *ENTCS*, pages 7–28. Elsevier, 2003.

41 Michael W. Mislove. Nondeterminism and probabilistic choice: Obeying the laws. In *CONCUR 2000*, pages 350–364. LNCS 1877, 2000. doi:10.1007/3-540-44618-4_26.

42 Michael W. Mislove. On combining probability and nondeterminism. *Electron. Notes Theor. Comput. Sci.*, 162:261–265, 2006. doi:10.1016/j.entcs.2005.12.113.

43 Eugenio Moggi. Computational lambda-calculus and monads. In *Fourth Annual IEEE Symposium on Logic in Computer Science*, pages 14–23, 1989.

44 Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

45 Prakash Panangaden. *Labelled Markov Processes*. Imperial College Press, 2009.

46 R. Segala. *Modeling and verification of randomized distributed real-time systems*. PhD thesis, MIT, 1995.

47 M.H. Stone. Postulates for the barycentric calculus. *Ann. Mat. Pura Appl. (4)*, 29:25–30, 1949. doi:10.1007/BF02413910.

48 T. Świrszcz. Monadic functors and convexity. *Bull. Acad. Polon. Sci. Sér. Sci. Math. Astronom. Phys.*, 22:39–42, 1974.

49 Qiyi Tang and Franck van Breugel. Deciding probabilistic bisimilarity distance one for probabilistic automata. In *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, pages 9:1–9:17, 2018. doi:10.4230/LIPIcs.CONCUR.2018.9.

50 Regina Tix, Klaus Keimel, and Gordon D. Plotkin. Semantic domains for combining probability and non-determinism. *Electron. Notes Theor. Comput. Sci.*, 222:3–99, 2009. doi:10.1016/j.entcs.2009.01.002.

51 Daniele Turi and Gordon D. Plotkin. Towards a mathematical operational semantics. In *Proc. LICS 1997*, pages 280–291, 1997. doi:10.1109/LICS.1997.614955.

52 D. Varacca and G. Winskel. Distributing probability over non-determinism. *Mathematical Structures in Computer Science*, 16:87–113, 2006.

Non Axiomatisability of Positive Relation Algebras with Constants, via Graph Homomorphisms

Amina Doumane
Université Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France
amina.doumane@ens-lyon.fr

Damien Pous
Université Lyon, CNRS, ENS de Lyon, UCB Lyon 1, LIP, Lyon, France
damien.pous@ens-lyon.fr

Abstract

We study the equational theories of composition and intersection on binary relations, with or without their associated neutral elements (identity and full relation). Without these constants, the equational theory coincides with that of semilattice-ordered semigroups. We show that the equational theory is no longer finitely based when adding one or the other constant, refuting a conjecture from the literature. Our proofs exploit a characterisation in terms of graphs and homomorphisms, which we show how to adapt in order to capture standard equational theories over the considered signatures.

2012 ACM Subject Classification Mathematics of computing → Discrete mathematics

Keywords and phrases Relation algebra, graph homomorphisms, (in)equational theories

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.29

Related Version Appendix available at `https://hal.archives-ouvertes.fr/hal-02870687`.

Funding This work has been funded by the European Research Council (ERC-StG CoVeCe 678157) and the French National Research Agency (ANR-10-LABX-0070 ANR-11-IDEX-0007).

1 Introduction

Several operations and constants are used frequently on binary relations: set-theoretic operations (intersection, union, complement, empty and full relations), relational composition and identity relation, and converse (transpose). Amongst others, Tarski studied those operations and analysed their expressiveness and the equational laws they satisfy [25, 26, 24, 19]. It turns out that these basic operations already make it possible to encode Peano arithmetic in a purely algebraic setting, without variables. As a consequence, the corresponding equational theory is undecidable and not finitely based [21].

The situation changes when considering *positive* fragments [13, 4, 1, 20, 2], where the complement operation is removed. Indeed, the equational theory of those fragments is decidable, even in the presence of additional operations like reflexive transitive closure [6, 22]. However, results concerning finite axiomatisations are more on the negative side. Hodkinson and Mikulás proved that one cannot obtain a finite firstorder axiomatisation whenever the operations of composition, intersection and converse are present [18]. When only two of those operations are considered, we get positive results: the problem is straightforward for converse and intersection; the case of composition and converse is more subtle and covered in [4, 11, 1]; and the equational theory of composition and intersection coincides with that of *semilattice-ordered semigroups* [3].

However, understanding the laws satisfied by the identity and the full-relation constants is difficult. They are neutral elements for composition and intersection, respectively, but they also satisfy rather unexpected laws, so that the equational theories depart from those of semilattice-ordered monoids and bounded-semilattice-ordered semigroups. The case of

composition, intersection, and identity was thought to be finitely based, with an explicit candidate [2], but there was an error in the completeness proof so that it remained as a conjecture. Our first contribution consists in refuting this conjecture: the equational theory of this fragment is not finitely based. Our second contribution is that adding the full relation to composition and intersection also yields an equational theory which is not finitely based.

The reasons for non finite axiomatisability of those two fragments are quite different. Still, our two proofs rely on a graph-theoretical characterisation the equational theory of binary relations [13, 1]. First, terms u, v built over a set of variables and the signature consisting of composition, intersection, and their neutral elements, can be used to denote graphs. The class of *expressible graphs*, those that may be denoted via a term, strongly depends on the considered fragment: they are always of treewidth at most two [8], they are also acyclic unless we have the identity constant, they are also connected unless we have the full-relation constant. The key result shared for all fragments is that a law $u \leq v$ is valid for relations if and only if there exists a homomorphism from the graph of v to the graph of u.

We prove the first negative result as follows: we first show that if we had a finite and equational axiomatisation, then we would be able to decompose every homomorphism between two expressible graphs while remaining within the class of expressible graphs – a similar idea is used in [13] for representable allegories; we formalise it in Section 3. Then we provide a counter-example: an infinite sequence of homomorphisms that cannot be decomposed accordingly, by exploiting a necessary condition for a graph to be expressible (Section 4).

We do not think a similar argument can used for the second negative result. Instead, we give directly an infinite sequence of homomorphisms and we show that for each of them, the corresponding law essentially has to be included into any sound and complete axiomatisation (Section 5).

As mentioned above, when neutral elements are taken into account, the equational theory of binary relations differs from that of natural algebraic structures extending semilattice-ordered semigroups (semilattice-ordered monoids, bounded-semilattice-ordered semigroups, and bounded-semilattice-ordered monoids). Our last contribution consists in providing graph-theoretical characterisations for those structures, yielding decidability in polynomial time of their equational theories (Section 6).

From the concurrency theory point of view, the structures considered here should not be confused with the ones studied in the litterature on pomsets [15, 14] or concurrent Kleene algebra [17]. Indeed, two forms of composition are also put forward in those lines of work: sequential and parallel composition, and they resemble the operations of relational composition and intersection we consider in the present paper (e.g., they form monoids related by the "weak exchange" law). However, the operation of intersection we use in the present paper is idempotent, and thus induces a partial order, which is not the case for parallel composition in concurrency theory. Accordingly, one should consider intersection as an operator for combining specifications rather than a program construction for concurrency, like with allegories and some of its extensions [13, 6, 23, 9].

2 Preliminaries

2.1 Terms

We fix in the rest of the paper an infinite alphabet A, and we let $a, b \dots$ range over its letters. $\mathsf{SP1}\top$ *terms* (series-parallel with one and $\top$) are generated by the following syntax

$$e, f ::= e \cdot f \mid e \cap f \mid 1 \mid \top \mid a \qquad (a \in A)$$

We denote their set by $\mathsf{SP1}\top$ and we often write ef for $e \cdot f$. We moreover assign priorities so that $ab \cap c$ reads as $(a \cdot b) \cap c$. We define $\mathsf{SP1}$, $\mathsf{SP}\top$ and SP to be respectively the set of $\mathsf{SP1}\top$ terms not containing $\top$, 1 and neither of them. We also use those symbols to denote the associated signatures.

2.2 Relational interpretation

We are primarily interested in the relational interpretation of terms, where $\cdot$ is relational composition, $\cap$ is set-theoretic intersection, 1 is the identity relation, and $\top$ is the full relation. Given an interpretation $\sigma : A \to \mathcal{P}(S \times S)$ of letters into some space of binary relations, we write $\widehat{\sigma} : \mathsf{SP1}\top \to \mathcal{P}(S \times S)$ for the corresponding extension to terms.

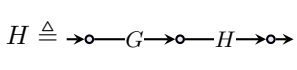
An inequation between two terms u and v is *valid*, written $\mathcal{R}el \models u \leq v$, if for every such interpretation σ we have $\widehat{\sigma}(u) \subseteq \widehat{\sigma}(v)$. We call *(in)equational theory of binary relations* the set $\mathcal{R}el$ of valid inequations. (We focus on inequations in the present work; note however that those are equivalent to equations: we have $\mathcal{R}el \models u \leq v$ iff $\mathcal{R}el \models u \cap v = u$, where the latter symbol is defined as expected.)

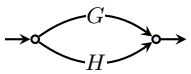
2.3 Graphs

As explained in the introduction, terms also make it possible to denote graphs – more precisely, directed multigraphs with edges labelled in A and two designated vertices. Formally, those are tuples $\langle V, E, s, t, l, \iota, o\rangle$ with V (resp. E) a finite set of vertices (resp. edges), $s, t : E \to V$ the *source* and *target* functions, $l : E \to A$ the *labelling* function, and $\iota, o \in V$ two distinguished vertices, respectively called *input* and *output*. We simply call them *graphs* in the sequel; we depict them as expected, with unlabelled ingoing and outgoing arrows to denote the input and the output, respectively.

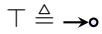
Vertices distinct from input and output are called *inner* vertices. A vertex without incident edges is *isolated*.

Graphs can be composed in series or in parallel, as depicted below:

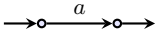
$G \cdot H \triangleq$ 

$G \cap H \triangleq$ 

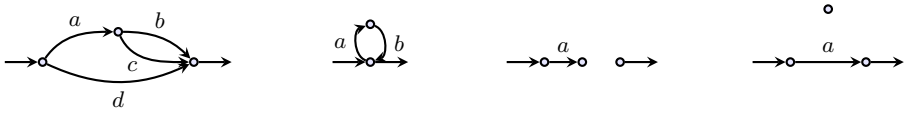
Those operations do have neutral elements, which are edge-less graphs:

$1 \triangleq$ $\top \triangleq$ 

We can thus recursively associate to every term u a graph $\mathcal{G}(u)$ called the *graph of* u, where the graph of a letter $a \in A$ is

$\mathcal{G}(a) \triangleq$ 

Here are, from left to right, the graphs of $a(b \cap c) \cap d$, $ab \cap 1$, $a\top$ and $a \cap \top\top$:



We say that a graph is SP (resp. $\mathsf{SP1}$, $\mathsf{SP}\top$, $\mathsf{SP1}\top$) if it is the graph of some SP (resp. $\mathsf{SP1}$, $\mathsf{SP}\top$, $\mathsf{SP1}\top$) term. The SP graphs are the acyclic and series-parallel graphs with all edges directed from the input towards the output. In a $\mathsf{SP1}$ graph, every vertex belongs to a directed path from the input towards the output; unlike SP graphs, they may contain cycles. In contrast, $\mathsf{SP}\top$ graphs remain acyclic but they are not necessarily connected. $\mathsf{SP1}\top$ have treewidth at most two, i.e., they are K_4-free [8].

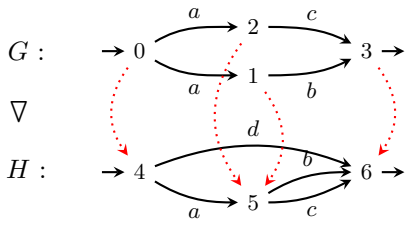
2.4 Homomorphisms

Graph homomorphisms play a central role in the paper; they are defined as follows:

▶ **Definition 1** (Graph homomorphism). *Given two graphs* $G = \langle V, E, s, t, l, \iota, o\rangle$ *and* $G' = \langle V', E', s', t', l', \iota', o'\rangle$*, a* (graph) homomorphism $h : G \to H$ *is a pair* $\langle h_v, h_e\rangle$ *of functions* $h_v : V \to V'$ *and* $h_e : E \to E'$ *that respect the various components:* $s' \circ h_e = h_v \circ s$*,* $t' \circ h_e = h_v \circ t$*,* $l = l' \circ h_e$*,* $\iota' = h_v(\iota)$*, and* $o' = h_v(o)$*.*

We write $H \lhd G$ *if there exists a graph homomorphism from* G *to* H*. Like other relations on graphs, we sometimes use this relation directly on terms, writing* $u \lhd v$ *for* $\mathcal{G}(u) \lhd \mathcal{G}(v)$*.*

The vertex component of such a homomorphism is depicted below



A pleasant way to think about graph homomorphisms is the following: we have $H \lhd G$ if H is obtained from G by merging (or identifying) some vertices and some edges, and by adding some extra vertices and edges. For instance, the graph H in the example above is obtained from G by merging vertices 1 and 2 and the two a-labelled edges, and by adding a d-labelled edge from the input to the output.

We write $h \circ g$ for the pointwise composition of the two components of two homomorphisms, which yields a homomorphism as expected.

A homomorphism is *injective* (resp. *surjective*, *bijective*) when its two components are so. We write $G \hookrightarrow H$ if there exists an injective homomorphism from G to H; in such a case, we say that G is a *subgraph* of H. We write $G \simeq H$ when there exists a bijective homomorphism from G to H (an *isomorphism*). The aforementioned intuition about homomorphisms is reflected by the epi-mono factorisation property: every homomorphism $h : G \to H$ factors uniquely into a surjective homomorphism followed by an injective homomorphism:

$$G \twoheadrightarrow h(G) \hookrightarrow H$$

The intermediate graph $h(G)$ is the *image of* h; it is a subgraph of H by definition.

The key result we exploit in the present paper is the following characterisation:

▶ **Theorem 2** ([1, Thm. 1], [13, p. 208]). *For all terms* u, v*,* $\mathcal{R}el \models u \leq v$ *iff* $u \lhd v$*.*

Thus, analysing the inequational theory of binary relations amounts to analysing homomorphism between graphs denoted by terms.

2.5 Closure under taking subgraphs

We show below that, SP$\top$ (resp. SP1$\top$), seen as a class of graphs, is the closure of SP (resp. SP1) under taking subgraphs. This property is convenient in the sequel.

▶ **Proposition 3.** *G is* SP$\top$ *(resp.* SP1$\top$*) iff G is a subgraph of a* SP *(resp.* SP1*) graph.*

Proof. We can reason mostly on terms. The forward implication is easy: given a $\mathsf{SP}\top$ (resp. $\mathsf{SP1}\top$) term u, replacing all occurrences of $\top$ with an arbitrary letter yields a SP (resp. $\mathsf{SP1}$) term u' such that $\mathcal{G}(u) \hookrightarrow \mathcal{G}(u')$.

For the converse implication, we proceed in two steps. Assume $h : G \hookrightarrow \mathcal{G}(u')$ for some SP (resp. $\mathsf{SP1}$) term u'. First observe that the edges in $\mathcal{G}(u')$ are in one-to-one correspondence with the occurrences of letters in u'. By replacing with $\top$ all occurrences of letters in u' that are not in the image of h through this correspondence, we obtain a $\mathsf{SP}\top$ (resp. $\mathsf{SP1}\top$) term u_0 such that h corestricts to $h_0 : G \hookrightarrow \mathcal{G}(u_0)$, which is actually bijective on edges. It remains to get rid of the vertices $\mathcal{G}(u_0)$ which are not in the image of h_0. Those are necessarily isolated inner vertices in $\mathcal{G}(u_0)$ since h_0 is bijective on edges. Roughly speaking, those arise via subterms of the shape $\top\top$ in u_0, which we can replace with $\top$ to obtain a $\mathsf{SP}\top$ (resp. $\mathsf{SP1}\top$) term u whose graph is G. The formal argument is slightly more involved; we give it in [10, Appendix A] ◀

▶ **Corollary 4.** *The classes of graphs* $\mathsf{SP}\top$ *and* $\mathsf{SP1}\top$ *are closed under taking subgraphs.*

2.6 Inequational reasoning

Let us define what we mean by axiomatisation in the present context, where we focus on inequations rather than equations.

Assume a signature Σ, and consider in this subsection terms u, v built over this signature and variables in the alphabet A. We let σ, θ range over *substitutions* assigning a terms to letters in A, and we write $u\sigma$ for the result of applying such a substitution σ to a term u. A *renaming* is a possibly non-injective substitution whose range consists only of letters. We let C range over *contexts*, i.e., terms with exactly one occurrence of a special letter $\bullet$ called the *hole*. We write $C[u]$ for the term obtained by replacing the hole of a context C by a term u.

An *inequation* is a pair of terms, which we denote by $u \leq v$. An *inequational theory* is a set of inequations which forms a pre-order and which is stable under contexts and substitutions. For instance, the set of inequations such that $\mathcal{R}el \models u \leq v$ is an inequational theory.

Given a set $\mathcal{H}$ of inequations, the *axioms*, the *inequational theory of* $\mathcal{H}$ is the least inequational theory containing $\mathcal{H}$. We write $\mathcal{H} \vdash u \leq v$ when the inequation $u \leq v$ belongs to the inequational theory of $\mathcal{H}$, or, equivalently, if it can be derived using the following rules:

$$\frac{}{u \leq v}\ \mathcal{H} \qquad \frac{}{u \leq u} \qquad \frac{u \leq v \qquad v \leq w}{u \leq w} \qquad \frac{u \leq v}{u\sigma \leq v\sigma} \qquad \frac{u \leq v}{C[u] \leq C[v]}$$

An inequational theory is *finitely based* if it can be generated by a finite set of axioms.

Standard algebraic structures

Inequational theories as defined above can be presented as equational theories as soon as the signature Σ contains a binary symbol $\cap$ and $\mathcal{H}$ contains the following finite set of inequations:

$$\mathcal{P} \triangleq \{a \leq a \cap a,\ a \cap b \leq a,\ a \cap b \leq b\}$$

Indeed, in such a case, $\cap$ turns the partial order $\leq$ into an *inf-semilattice*, and inf-semilattices can be defined algebraically as commutative idempotent semigroups: the partial order can be defined as $u \leq v \triangleq (u \cap v = u)$. The other operations in the signature must all be monotone, which can be expressed algebraically by adding equations of the form $f(a \cap b) \cap f(a) = f(a)$, say, for a unary symbol f. Conversely, any equational theory with a commutative idempotent

semigroup symbol and where all operations are monotone w.r.t. the associated partial order can be represented as an inequational theory in the previous sense. (Those conversions preserve the finiteness of the considered set of axioms, so that an inequational theory is finitely based iff its associated equational theory is finitely based, and vice versa.)

In particular, we capture other standard algebraic structures as follows. Define the following (finite) sets of inequations, where an equation is a shorthand for the corresponding two inequations:

$$\mathcal{SP} \triangleq \mathcal{P} \cup \{a \cdot (b \cdot c) = (a \cdot b) \cdot c\} \qquad \mathcal{SP}^{\top} \triangleq \mathcal{SP} \cup \{a \leq \top\}$$
$$\mathcal{SP}^{1} \triangleq \mathcal{SP} \cup \{a \cdot 1 = a, 1 \cdot a = a\} \qquad \mathcal{SP}^{1\top} \triangleq \mathcal{SP}^{1} \cup \mathcal{SP}^{\top}$$

(Note that these sets are implicitly associated to the four signatures we consider in the present paper: for instance, when writing $\mathcal{SP}^{1} \vdash u \leq v$, we mean that u and v are $\mathsf{SP1}$ terms and that the derivation mentions only $\mathsf{SP1}$ terms, contexts, and substitutions.)

We have that

- $\mathcal{SP}$ axiomatises *semilattice-ordered semigroups (sl-semigroups)*;
- $\mathcal{SP}^{1}$ axiomatises *semilattice-ordered monoids (sl-monoids)*;
- $\mathcal{SP}^{\top}$ axiomatises *bounded-semilattice-ordered semigroups (bsl-semigroups)*;
- $\mathcal{SP}^{1\top}$ axiomatises *bounded-semilattice-ordered monoids (bsl-monoids)*.

Axiomatisability of relations

Given a subsignature X of $\mathsf{SP1}\top$, we say that a set $\mathcal{H}$ of inequations on X *axiomatises relations on* X if

$$\text{for all terms } u, v \text{ on } X, \qquad \mathcal{H} \vdash u \leq v \quad \text{iff} \quad \mathcal{R}el \models u \leq v \ .$$

Bredihin and Schein proved that the equational theory of relations on SP coincides with that of sl-semigroups [3], which means in the above terminology that $\mathcal{SP}$ axiomatises relations on SP.

In contrast, $\mathcal{SP}^{1}$ and $\mathcal{SP}^{\top}$ do not suffice to axiomatise relations on $\mathsf{SP1}$ or $\mathsf{SP}\top$. For instance, relations satisfy the laws below (by Theorem 2, this can be proved by providing appropriate homomorphisms – most of them actually are isomorphisms here) but there are bsl-monoids violating those laws[1].

$$\mathcal{R}el \vdash a\top \cap bc = (a\top \cap b)c \qquad \mathcal{R}el \vdash \top a \top b \top = \top b \top a \top$$
$$\mathcal{R}el \vdash a \cap b \cap 1 = (a \cap 1)(b \cap 1) \qquad \mathcal{R}el \vdash (a \cap 1)b \cap c = (a \cap 1)(b \cap c) \qquad \mathcal{R}el \vdash a \cap 1 \leq aa$$

In fact, as shown in the sequel, $\mathcal{R}el$ is not finitely based on $\mathsf{SP1}$, $\mathsf{SP}\top$, and $\mathsf{SP1}\top$ (so that the corresponding equational theories are not finitely based either).

3 Decomposability

We fix a signature $X \in \{\mathsf{SP}, \mathsf{SP1}, \mathsf{SP}\top, \mathsf{SP1}\top\}$ in this section, and we provide a necessary condition for finite axiomatisability of relations on X (and thus existence of graph homomorphisms between X graphs). This is essentially the same condition as the one used by Freyd and Scedrov for representable allegories [13, pp 208–210]. We generalise it here so that it fits our needs, providing a different proof and more explicit treatment.

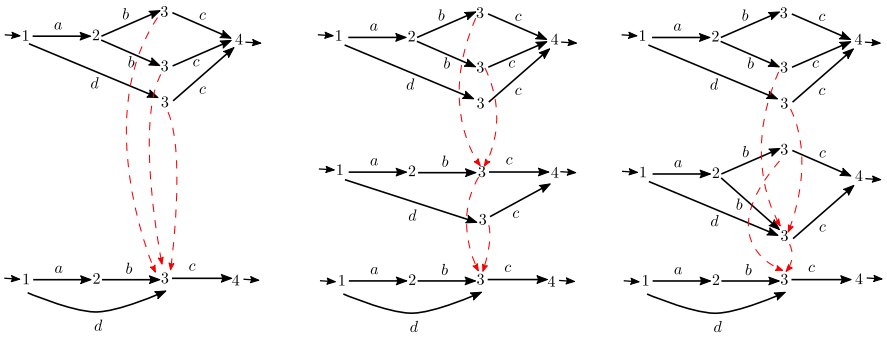
[1] even finite ones, that can easily be found with tools such as Mace4.

Recall that a homomorphism can be seen as the action of merging several vertices and edges of the source graph and adding some vertices and edges. The *degree* of a homomorphism is the number of vertices it merges:

▶ **Definition 5** (Degree). *Let $h = (h_e, h_v)$ be a graph homomorphism. The* degree of h, *denoted* $\deg(h)$ *is the number* $\#\{u \mid \exists w, w \neq u \text{ and } h_v(u) = h_v(w)\}$. *We write $H \lhd_n G$ when there exists $h : G \to H$ with $\deg(h) \leq n$.*

Since the vertices of a graph can always be merged two at a time, every homomorphism can be decomposed into a sequence of homomorphisms of degree at most two. However, intermediate graphs in this decomposition are not necessarily in X, even if the endpoints are.

For instance, we depict on the left below a homomorphism of degree three between two SP graphs (top-down). This homomorphism can be decomposed into two sequences of homomorphisms of degree two, given in the middle and on the right. The intermediate graph in the middle is SP, while the intermediate graph on the right is not SP.



We write $u \lhd_n^X v$ when u, v are terms of X and $\mathcal{G}(u) \lhd_n \mathcal{G}(v)$; we write $\lhd_n^{X*}$ for the reflexive transitive closure of this relation.

In the above example, the valid law corresponding to $(ab \cap d)c \lhd a(bc \cap bc) \cap dc$ can be decomposed into two valid laws $(ab \cap d)c \lhd_2^{\mathsf{SP}} abc \cap dc$ and $abc \cap dc \lhd_2^{\mathsf{SP}} a(bc \cap bc) \cap dc$. These latter laws are intuitively simpler: they can be justified by homomorphisms with a smaller degree.

We need the following assumption about X to obtain Proposition 8 below.

▶ **Assumption 6.** *There is an integer k such that for every term u of X, we have $u \lhd_k^{X*} u \cap u$.*

This is a rather mild assumption, which is satisfied in the context of the present paper:

▶ **Fact 7.** *Assumption 6 is satisfied with $k = 2$ for the four values of X considered here.*

Proof. By a straightforward induction on u in each case. ◀

▶ **Proposition 8.** *Under Assumption 6, if $\mathcal{R}el$ is finitely based on X, then there exists an integer n such that for all terms u, v in X, $u \lhd v$ entails $u \lhd_n^{X*} v$.*

Proof. Suppose that we have a finite axiomatisation $\mathcal{H}$. By soundness (and Theorem 2), each axiom of $\mathcal{H}$ gives rise to a graph homomorphism. Let n' be the maximal degree of these homomorphisms, and let $n = \max\{k, n'\}$. Now suppose $u \lhd v$ for some terms u, v of X. By completeness (and Theorem 2), we get a derivation $\mathcal{H} \vdash u \leq v$. We prove $u \lhd_n^{X*} v$ by induction on this derivation. The only interesting case is that of the substitution rule. In this case, we have $u \lhd_n^{X*} v$ by induction, and we must prove $u\sigma \lhd_n^{X*} v\sigma$ for a given substitution σ. W.l.o.g., we can assume $u \lhd_n^X v$, and we consider the underlying homomorphism $h : v \to u$,

of degree at most n. Observe that h can be extended into a homomorphism $h\sigma : v\sigma \to u\sigma$. If h is injective on edges, then $\deg(h\sigma) = \deg(h) \leq n$ and we are done. If instead h merges two a-labelled edges, then the degree of $h\sigma$ is at least the number of inner vertices of $\sigma(a)$, which is not bounded by n, *a priori*. In this latter case, we use the homomorphism $h' : v \to u'$ obtained from h by duplicating all edges in the image graph as many times as necessary to get injectivity on edges. The corresponding term u' is obtained from u be replacing some occurrences of letters, say a, with intersections of the same letter (e.g., $a \cap a \cap a$). The homomorphism $h'\sigma : v\sigma \to u'\sigma$ satisfies $\deg(h'\sigma) = \deg(h') = \deg(h) \leq n$, so that $u'\sigma \lhd_n^{X^*} v\sigma$ We finally obtain $u\sigma \lhd_k^{X^*} u'\sigma$ by repeatedly using Assumption 6. ◀

By contraposition, to prove non-finite-axiomatisability, it suffices to find a sequence of homomorphisms $(e_n \lhd f_n)_{n\in\omega}$ between terms of X such that for all n, $e_n \lhd_n^{X^*} f_n$ does not hold. We define such a sequence in the following section, for SP1 and SP1$\top$.

4 The fragments SP1 and SP1T

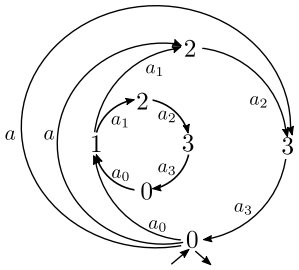
We show in this section that $\mathcal{R}el$ is not finitely based on SP1 and SP1$\top$.

▶ **Definition 9.** *Let $a, a_0, a_1, \ldots$ be a fixed sequence of pairwise disjoint letters, and let n be a strictly positive integer. Let c_n be the term $(a_1a_2 \ldots a_na_0) \cap 1$. We define the terms e_n and f_n with the help of the families $(g_i^n)_{i\in[1,n-1]}$ and $(h_i^n)_{i\in[1,n-1]}$ respectively as follows:*

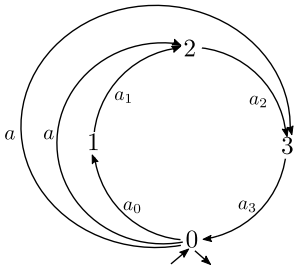
$$g_0^n = a_0c_n, \qquad g_{i+1}^n = (g_i^n a_{i+1}) \cap a, \qquad e_n = (g_{n-1}^n a_n) \cap 1.$$
$$h_0^n = a_0, \qquad h_{i+1}^n = (h_i^n a_{i+1}) \cap a, \qquad f_n = (h_{n-1}^n a_n) \cap 1.$$

For instance, the graphs of e_3 and f_3 are:

$\mathcal{G}(e_3)$:



$\mathcal{G}(f_3)$:



There is a homomorphism from $\mathcal{G}(e_3)$ to $\mathcal{G}(f_3)$, which maps the nodes of $\mathcal{G}(e_3)$ tagged by $i \in [0, 3]$ to the node of $\mathcal{G}(f_3)$ tagged i. More generally, it is not hard to see that, for every $n \in \omega$, there is a (unique) homomorphism from $\mathcal{G}(e_n)$ to $\mathcal{G}(f_n)$, that is $f_n \lhd e_n$. Let us state the main proposition of this section:

▶ **Proposition 10.** *For every $n \in \omega$, $f_n \not\lhd_n^{\mathsf{SP1}\top *} e_n$.*

Together with Proposition 8, this entails that $\mathcal{R}el$ is not finitely based on SP1$\top$. Since e_n and f_n actually are SP1 terms, this also entails that it is not finitely based on SP1.

We prove Proposition 10 below, by contradiction. In order to ease this proof, we first establish a property verified by SP1$\top$ graphs; this will allow us to reach a contradiction in the two main cases of the proof.

▶ **Definition 11** (Back pattern)**.** *A* back pattern *in a graph is a pair of distinct nodes m, n together with three directed paths: π from the input to m, κ from n to m, and ρ from n to the output, such that π and κ intersect exactly on m and κ and ρ intersect exactly on n.*

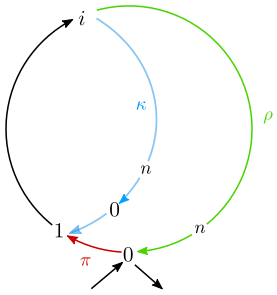


Figure 1 First case in proof of Prop. 10.

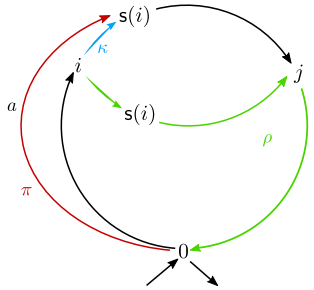


Figure 2 Second case in proof of Prop. 10.

Such a back pattern can be depicted as follows: $\iota \xrightarrow{\pi} m \xleftarrow{\kappa} n \xrightarrow{\rho} o$. They cannot arise in $\mathsf{SP1}\top$ graphs. Intuitively, although $\mathsf{SP1}$ graphs are not acyclic (unlike SP graphs), they are nevertheless oriented from the input towards the output.

▶ **Proposition 12.** $\mathsf{SP1}\top$ *graphs do not contain back patterns.*

A proof is given found in [10, Appendix B]; it is an easy induction on the structure of $\mathsf{SP1}\top$ terms, after adding some other forbidden patterns for the induction to go through. Proposition 12 can be extended to characterise $\mathsf{SP1}$ graphs (and thus $\mathsf{SP1}\top$ graphs via Corollary 4) using a slight relaxation of the definition of back patterns. Such a characterisation is not needed here, however, and we can now prove Proposition 10.

Proof of Proposition 10. Let h be the (unique) homomorphism from $\mathcal{G}(e_n)$ to $\mathcal{G}(f_n)$. To simplify the presentation, we label the nodes of e_n and f_n by integers from $[0, n]$, in the same way as in the above example for $n = 3$: the input is labelled 0, and for every $i \in [0, n-1]$, if a node is labelled i, then its a_i successors are labelled $i + 1$. Note that, for every $i \in [1, n]$, there is exactly one node labelled i in f_n; its pre-image by h consists of those nodes labelled by i in $\mathcal{G}(e_n)$.

Suppose by contradiction that $(f_n, e_n) \in \lhd_n^*$. Thus, we can find $\mathsf{SP1}\top$ terms $(g_i)_{i\in[0,m+1]}$ and homomorphisms $(h_i : g_i \to g_{i+1})_{i\in[0,m]}$ of degree at most n, and such that $g_0 = e_n$ and $g_{m+1} = f_n$. The composition of the homomorphisms $(h_i)_{i\in[0,m]}$ yields h.

Let k be an index such that $e_n \hookrightarrow g_k$ and $e_n \not\hookrightarrow g_{k+1}$. This index exists since $e_n \hookrightarrow g_0$ and $e_n \not\hookrightarrow g_{m+1}$. As $\mathcal{G}(e_n)$ is a sub-graph of $\mathcal{G}(g_k)$, we label the nodes of $\mathcal{G}(g_k)$ accordingly: keep the same labels for nodes in $\mathcal{G}(e_n)$, and do not label the other nodes.

The fact that $e_n \not\hookrightarrow g_{k+1}$ means that the homomorphism h_k merged some labelled nodes. Note that h_k cannot merge nodes tagged by different integers. Otherwise, the composition of the homomorphisms $(h_i)_{i\in[0,m]}$ would also merge nodes tagged by different labels, which is not possible. We label the nodes of $\mathcal{G}(g_{k+1})$ according to h_k: a node is labelled i if it is the image of a node labelled i by h_k.

Let us show that g_{k+1} cannot be an $\mathsf{SP1}\top$ term. Let i be the largest integer in $[1, n]$ such that for every $j \in [1, i]$ the nodes labelled by j in $\mathcal{G}(g_k)$ have been merged by h_k.

We define the function $\mathsf{s} : [1, n] \to [0, n]$ as follows: $\mathsf{s}(x) = x + 1$ if $x \in [1, n-1]$ and $\mathsf{s}(n) = 0$. Note that the nodes labelled by $\mathsf{s}(i)$ in $\mathcal{G}(g_k)$ are not merged by h_k. If this was the case, then either $i \in [1, n-1]$, which would contradict maximality of i, or $i = n$, meaning that the degree of the homomorphism is at least $n + 1$, contradicting our hypothesis.

We distinguish two cases:

- For every $j \in [i + 1, n] \cup \{0\}$, the nodes labelled j in $\mathcal{G}(g_k)$ are not merged by h_k. In this case, $i \neq 1$, otherwise no labelled nodes of $\mathcal{G}(g_k)$ would be merged by h_k. There are two distinct directed paths from i to 1 in $\mathcal{G}(g_{k+1})$: one that visits the input and one that

does not. We name the first one κ. The second can be decomposed into a path from i to the input, we name it ρ, and a path from the input to 1, we name it π.
Then the nodes labelled i and 1, together with the paths π, κ, ρ form a back-pattern in $\mathcal{G}(g_{k+1})$, as illustrated in Figure 1.

- There is $j \in [i+1, n] \cup \{0\}$ such that the nodes labelled j are merged. Recall that the nodes labelled $\mathsf{s}(i)$ are not merged by h_k. We call m the node of $\mathcal{G}(g_{k+1})$, which is labelled $\mathsf{s}(i)$ and which is the a successor of the input. Note that since the nodes labelled j are merged, there is a path in $\mathcal{G}(g_{k+1})$ connecting the node labelled i to the input (which in this case coincide with the output), which does not go through m. We call this path ρ. We call π the path labelled a from the input to m and κ the path labelled a_i from i to $\mathsf{s}(i)$. The nodes $\mathsf{s}(i)$ and i, together with the paths π, κ and ρ form a back-pattern in $\mathcal{G}(g_{k+1})$, as illustrated in Figure 2. ◀

5 The fragment SPT

We show in this section that $\mathcal{R}el$ is not finitely based on $\mathsf{SP}\top$. We do not adopt the same strategy as for $\mathsf{SP1}$[2]. Instead, our proof in this case is in two steps. First, we show that every axiomatisation can be turned into one with a very constrained shape, called *simple axiomatisation*. In a second step, we exhibit an infinite collection of inequations $(f_n \leq e_n)_{n\in\omega}$ which any simple axiomatisation should contain in a certain sense.

5.1 Dealing with idempotency

Simple axiomatisations, which we introduce in the following, are axiomatisations where idempotency of intersection is used in a very controlled way, following Freyd and Scedrov' idea of *separatedness* [13, page 208].

We call *idempotency axiom* an inequation of the form $a \leq a \cap a$ for some letter a.

▶ **Definition 13.** *A term v is* simple *if every letter appears at most once in v. An inequation $u \leq v$ is* simple *if v is simple. An axiomatisation $\mathcal{H}$ is* simple *if contains only simple axioms and idempotency axioms.*

The key intuition about simple axioms is that the corresponding homomorphisms cannot merge edges; in a sense, they are idempotency-free.

For every term v, there is a simple term v' and a renaming θ_v such that $v = v'\theta_v$ (e.g., for $v = aa$, take $v' = a_1a_2$ and $\theta_v = \{a_1, a_2 \mapsto a\}$). In such a case, write θ_v^{-1} for the following substitution: $\theta_v^{-1}(a) = \bigcap_{\theta_v(a')=a} a'$. For every term u such that $u \lhd v$, we have $u\theta_v^{-1} \lhd v'$.

We can thus turn any sound axiomatisation $\mathcal{H}$ into a simple axiomatisation $\mathcal{H}'$ as follows: adjoin an idempotency axiom, and replace each non-simple axiom $u \leq v$ of $\mathcal{H}$ with $u\theta_v^{-1} \leq v'$. Every axiom of $\mathcal{H}$ is derivable in $\mathcal{H}'$ using its simple counterpart and the idempotency axiom; therefore if $\mathcal{H}$ axiomatises $\mathcal{R}el$ then so does $\mathcal{H}'$. Note that $\mathcal{H}'$ is finite whenever $\mathcal{H}$ is finite.

Idempotency can thus be isolated from the other axioms. We go further and show that its use may actually be pushed towards the leaves. Let $\mathcal{I}$ be the following set of inequations

$$a \leq a \cap a \quad \top \leq \top \cap \top \quad (a \cap c)(b \cap d) \leq ab \cap cd \quad (a \cap c) \cap (b \cap d) \leq (a \cap b) \cap (c \cap d)$$

These axioms are sound w.r.t. $\mathcal{R}el$; we need these axioms to obtain Proposition 15 below: every derivation can be seen as a sequence of rewriting steps where idempotency axioms are used only on letters (i.e., to merge parallel edges).

[2] We actually conjecture that for all terms u, v in $\mathsf{SP}\top$, $u \lhd v$ entails $u \lhd_2^{\mathsf{SP}\top *} v$.

▶ **Definition 14.** *Given two terms e, f, if $e = C[u\sigma]$ and $f = C[v\sigma]$ for some context C, substitution σ, and terms u, v, we say that (C, σ) is a* unifying context-substitution for e and f with inner terms u and v.

▶ **Proposition 15.** *If $\mathcal{I} \subseteq \mathcal{H}$ and $\mathcal{H} \vdash f \leq e$, then there is a sequence $g_0, \dots, g_m$ of terms such that $e = g_0$, $g_m = f$, and for all $i < m$, there is a unifying context-substitution for g_i and g_{i+1} with inner terms u and v such that $(u \leq v) \in \mathcal{H}$, and the substitution is a renaming if the latter axiom is an idempotency axiom.*

Proof. A simple induction on the derivation yields a sequence as in the statement, but without the constraint idempotency axioms. We refine this sequence by using the following property:

> For every $\mathsf{SP}\top$ term e there is a sequence $g_0, \dots, g_m$ of terms such that $e \cap e = g_0$, $g_n = e$, and for all $i < m$, there is a unifying context-substitution for g_i and g_{i+1} with inner terms u and v such that $(u \leq v) \in \mathcal{I}$, and the substitution is a renaming if the latter axiom is the idempotency axiom of $\mathcal{I}$.

This property is proved by an easy induction on e, using in each case the corresponding axiom of $\mathcal{I}$, e.g., in the product case,

$$ef \leq (e \cap e)f \leq (e \cap e)(f \cap f) \leq ef \cap ef$$

where the first two (sequences of) steps are obtained by induction hypothesis and the third step is an instance of the third axiom of $\mathcal{I}$. ◀

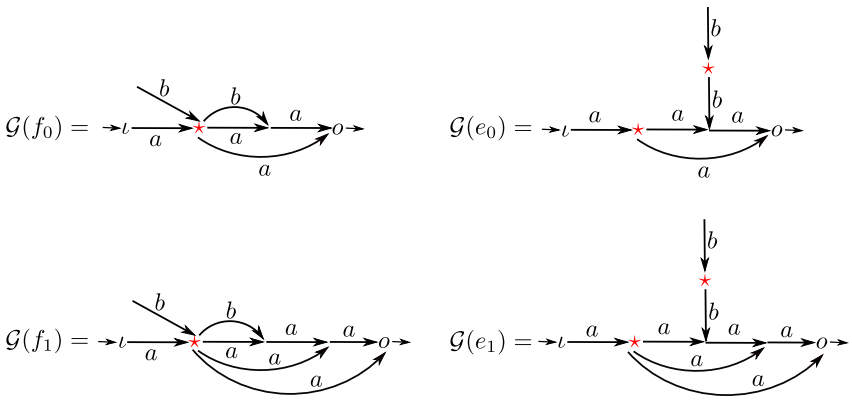
5.2 The counter-example

We can finally give the counter-example.

▶ **Definition 16.** *Let a, b be fixed letters. Given $n \in \omega$, the $\mathsf{SP}\top$ terms e_n and f_n are defined as follows, with the help of two sequences $(u_i^n)_{i \leq n}$ $(v_i^n)_{i \leq n}$ parameterised by n:*

$$u_0^n = (\top bb \cap a)a \cap a \qquad u_{i+1}^n = (u_i^n a) \cap a \qquad e_n = a u_n^n$$
$$v_0^n = (a \cap b)a \cap a \qquad v_{i+1}^n = (v_i^n a) \cap a \qquad f_n = (\top b \cap a) v_n^n$$

The graphs of f_0, e_0 and f_1, e_1 are depicted below.



There is a unique homomorphism $!_n$ from e_n to f_n. This homomorphism is surjective, it just merges two vertices of the graph of e_n, which are tagged by red stars in the picture above.

The key property of the sequences $(e_n)_{n \in \omega}$ and $(f_n)_{n \in \omega}$ is stated in Proposition 18 below: every inequation $e_n \leq f_n$ can be injected up-to renaming into an inequation of $\mathcal{H}$.

▶ **Definition 17.** *We say that* v injects up-to renaming into u*, written* $v \hookrightarrow_\alpha u$*, if* $v \hookrightarrow u\theta$ *for some renaming* θ*.*

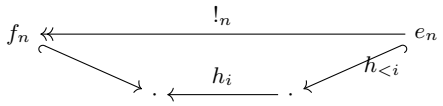
▶ **Proposition 18.** *If* $\mathcal{H}$ *is a simple axiomatisation of relations on* $\mathsf{SP}\top$*, then for every* $n \in \omega$*, there are* $\mathsf{SP}\top$ *terms* e'_n *and* f'_n *such that* $f_n \hookrightarrow_\alpha f'_n$*,* $e_n \hookrightarrow_\alpha e'_n$ *and* $(f'_n \leq e'_n) \in \mathcal{H}$*.*

As the size of the graphs of e_n and f_n grows infinitely, the set of inequations $(f'_n \leq e'_n)_{n\in\omega}$ is infinite. Thus $\mathcal{R}el$ is not finitely based on $\mathsf{SP}\top$. (Note that the above proposition is stronger than necessary: we could focus on one side of the equations.)

The proof of Proposition 18 relies on the three following lemmas, whose proofs can be found in [10, Appendix C]. The first one says that if $!_n : e_n \twoheadrightarrow f_n$ decomposes into a sequence of homomorphisms, then of one them must essentially act like $!_n$ under some irrelevant context.

▶ **Lemma 19.** *Fix* $n \in \omega$ *and assume* $!_n : e_n \twoheadrightarrow f_n$ *decomposes into a sequence* $h_m \circ \cdots \circ h_0$*. For* $i < m$*, write* $h_{<i}$ *for the partial composition* $h_{i-1} \circ \cdots \circ h_0$*. There exists an index* $i < m$ *such that* $h_{<i}$ *is injective and the image of* $h_{<i+1}$ *is the graph of* f_n*.*

In other words, in the above statement, $h_{<i+1}$ factors through f_n as in the diagram below.



The second lemma essentially says that if there is a unifying context-substitution (Definition 14) for e_n and f_n whose inner terms are related by a homomorphism, then the context is necessarily trivial and e_n and f_n can be injected up-to renaming into the inner terms. We need to be slightly more flexible and we use the following notation: we write $u \dot{\hookrightarrow} v$ if there is a homomorphism from u to v which is bijective on edges and injective on vertices. In other words, $u \dot{\hookrightarrow} v$ when v is u with some additional isolated vertices.

▶ **Lemma 20.** *Fix* $n \in \omega$ *and suppose that there is a context* C*, a substitution* σ *and two terms* u *and* v *such that* $f_n \dot{\hookrightarrow} C[v\sigma]$*,* $e_n \dot{\hookrightarrow} C[u\sigma]$*, and* $v \lhd u$*. Then* $f_n \hookrightarrow_\alpha v$ *and* $e_n \hookrightarrow_\alpha u$*.*

The last ingredient says that when a term e can be injected in a term of the form $C[u\sigma]$, for a simple term u, then we can restrict C, u and σ to match e up to some isolated vertices.

▶ **Lemma 21.** *If* $\iota : e \hookrightarrow C[u\sigma]$ *with* u *simple, then there is a context* $C' \hookrightarrow C$*, a substitution* $\sigma' \hookrightarrow \sigma$ *and a term* $u' \hookrightarrow u$ *such that* ι *decomposes into* $e \dot{\hookrightarrow} C'[u'\sigma'] \hookrightarrow C[u\sigma]$*. (Where we extend* $\hookrightarrow$ *to substitutions componentwise:* $\sigma \hookrightarrow \gamma$ *if* $dom(\sigma) \subseteq dom(\gamma)$ *and* $\forall a \in dom(\sigma), \sigma(a) \hookrightarrow \gamma(a)$*.)*

Note that the requirement that u must be simple cannot be dropped (because, e.g., $bc \hookrightarrow (b \cap c)(b \cap c) = (aa)\{a \mapsto b \cap c\}$).

Proof of Proposition 18. Let $\mathcal{H}$ be a simple axiomatisation of relations on $\mathsf{SP}\top$. Since $\mathcal{I}$ is simple and none of the $f_n \leq e_n$ injects up-to renaming into $\mathcal{I}$, we can assume w.l.o.g that $\mathcal{H}$ contains $\mathcal{I}$.

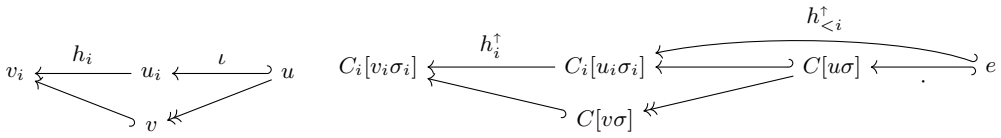
Let $n \in \omega$. In the rest of this proof we write e, f, and $!$ for e_n, f_n, and $!_n$, respectively.

Since $f \lhd e$, we have $\mathcal{H} \vdash f \leq e$ by completeness (and Theorem 2). By Proposition 15, we obtain a sequence $g_0, \ldots, g_m$ of terms such that $e = g_0$, $g_m = f$, and for all $i < m$, there is a unifying-context substitution (C_i, σ_i) for g_i and g_{i+1} with inner terms u_i and v_i (i.e., $g_i = C_i[u_i\sigma_i]$ and $g_{i+1} = C_i[v_i\sigma_i]$) such that either $(v_i \leq u_i) \in \mathcal{H}$ is simple or this is an idempotency axiom and σ_i is a renaming.

By soundness (and Theorem 2), there are homomorphisms $h_i : u_i \to v_i$ for all $i < m$. Each of these homomorphisms can be extended, through the context C_i and the substitution σ_i, into a homomorphism $h_i^{\uparrow} : g_i \to g_{i+1}$. The composition of these homomorphisms must be the only homomorphism $! : e \twoheadrightarrow f$. Hence, by Lemma 19, there is an index i such that $h_{<i}^{\uparrow}$ is injective and the image of $h_{<i+1}^{\uparrow}$ is f. Since h_i merges two vertices, it cannot be the case that this step is an idempotency step: since those are restricted to letters, they only merge edges. Therefore u_i must be simple.

Since $h_{<i}^{\uparrow} : e \hookrightarrow g_i = C_i[u_i\sigma_i]$, Lemma 21 gives us a context $C \hookrightarrow C_i$, a substitution $\sigma \hookrightarrow \sigma_i$ and a term $u \hookrightarrow u_i$ such that $\iota(e) \dot{\hookrightarrow} C[u\sigma]$. Call ι the injective homomorphism $u \hookrightarrow u_i$.

The image of $h_i \circ \iota : u \to v_i$, as a subgraph of v_i, must be a $\mathsf{SP}\top$ term v by Proposition 3. This decomposition corresponds to the commuting diagram on the left below, and assembling the various ingredients collected so far, we obtain the commuting diagram on the right.



Since f is the image of $h_{<i+1}^{\uparrow}$, we deduce $f \dot{\hookrightarrow} C[v\sigma]$.

We can finally use Lemma 20 to deduce $e \hookrightarrow_\alpha u$ and $f \hookrightarrow_\alpha v$. As $u \hookrightarrow u_i$ and $v \hookrightarrow v_i$, we also have that $e \hookrightarrow_\alpha u_i$ and $f \hookrightarrow_\alpha v_i$. Since $(v_i \leq u_i) \in \mathcal{H}$ this concludes the proof. ◀

6 Graph theoretical characterisation for natural structures

The characterisation of $\mathcal{R}el$ in terms of graph homomorphisms (Theorem 2) works for all $\mathsf{SP1}\top$ terms. This inequational theory coincides with that of sl-semigroups ($\mathcal{SP}$) for SP terms, but we have seen that it departs from related algebraic structures (sl-monoids, bsl-semigroups and bsl-monoids) for $\mathsf{SP1}$, $\mathsf{SP}\top$, and $\mathsf{SP1}\top$ terms. We show in this section that we can nevertheless obtain simple graph theoretical characterisations of the (in)equational theory of those three algebraic structures.

Let us focus on the SP case first, for which the notions we have used so far just work:

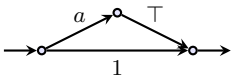
▶ **Proposition 22.** *For all $u, v \in \mathsf{SP}$, $\mathcal{SP} \vdash u \leq v$ iff $\mathcal{G}(u) \lhd \mathcal{G}(v)$.*

This result is a consequence of Theorem 2 and [3], but we give below a direct proof due to Brunet [5]. We say that a graph *goes forward* if every vertex belongs to a simple directed path from the input to the output, and so does every edge. All SP graphs go forward, which make it possible to obtain the following property:

▶ **Lemma 23.** *For all SP terms u, v_1, v_2 such that $\mathcal{G}(u) \lhd \mathcal{G}(v_1 v_2)$, there are SP terms u_1, u_2 such that $\mathcal{G}(u_1) \lhd \mathcal{G}(v_1)$, $\mathcal{G}(u_2) \lhd \mathcal{G}(v_2)$, and $\mathcal{SP} \vdash u \leq u_1 u_2$.*

Proposition 22 follows easily (see [10, Appendix D]).

The above lemma cannot be adapted directly in the presence of 1 and $\top$, because the graphs no longer go forward, Instead, we start from a simpler interpretation of terms into graphs, ensuring that we keep forward graphs: we simply interpret 1 and $\top$ as letters, so that graphs are now labelled in $A \uplus \{1\} \uplus \{\top\}$. We write $\mathcal{G}'(\cdot)$ for the corresponding interpretation function. For instance, $\mathcal{G}'(a\top \cap 1)$ is the following graph:



The previous notion of homomorphism is of course too strict, as it leaves the constants 1 and $\top$ uninterpreted. We thus adjust the notion of homomorphism below. We first define some notations: given two vertices x, y in a graph as above, we write

- $x \Rightarrow y$ if there is a directed path from x to y along edges labelled with 1;
- $x \overset{a}{\Rightarrow} y$ if $x \Rightarrow x' \overset{a}{\to} y' \Rightarrow y$ for some vertices x' and y';
- $x \to^* y$ if there is a directed path from x to y along arbitrary edges;
- $x \to^+ y$ if there is a non-empty directed path from x to y along arbitrary edges.

▶ **Definition 24.** *Fix X in $\{1, \top, 1\top\}$ and two graphs G, H as above. An X-homomorphism from G to H is a function h from the vertices of G to those of H preserving input and output, and such that:*

- (a) *if $x \overset{a}{\to} y$ in G then $h(x) \overset{a}{\Rightarrow} h(y)$ in H;*
- (1) *for $X \in \{1, 1\top\}$, if $x \overset{1}{\to} y$ in G then $h(x) \Rightarrow h(y)$ in H;*
- $(\top)$ *for $X = \top$, if $x \overset{\top}{\to} y$ in G then $h(x) \to^+ h(y)$ in H;*
- $(\top')$ *for $X = 1\top$, if $x \overset{\top}{\to} y$ in G then $h(x) \to^* h(y)$ in H;*

We write $H \lhd^X G$ when there exists such a X-homomorphism.

We finally state our three characterisations, for sl-monoids, bsl-semigroups, and bsl-monoids.

▶ **Theorem 25.** *For all X in $\{1, \top, 1\top\}$, for all $u, v \in \mathsf{SP}X$, $\mathcal{SP}^X \vdash u \leq v$ iff $\mathcal{G}'(u) \lhd^X \mathcal{G}'(v)$.*

The distinction in the clause for $\top$-labelled edges in the definitions of $\top$- and $1\top$-homomorphisms is required because bsl-monoids validate $\top \leq \top\top$ (for instance, because $\top = 1\top \leq \top\top$) while bsl-semigroups do not: allowing to use empty-paths with $1\top$-homomorphisms makes it possible to absorb one of the two edges of $\mathcal{G}'(\top\top)$, while this not possible with $\top$-homomorphisms.

Lemma 23 extends as follows:

▶ **Lemma 26.** *For all X in $\{1, \top, 1\top\}$, for all $\mathsf{SP}X$ terms u, v_1, v_2 s.t. $\mathcal{G}'(u) \lhd^X \mathcal{G}'(v_1 v_2)$, there are $\mathsf{SP}X$ terms u_1, u_2 such that $\mathcal{G}'(u_1) \lhd^X \mathcal{G}'(v_1)$, $\mathcal{G}'(u_2) \lhd \mathcal{G}'(v_2)$, and $\mathcal{SP} \vdash u \leq u_1 u_2$.*

Like in the SP case, this lemma is proved by induction on the size of u, producing terms u_1, u_2 such that $\mathcal{G}'(u_1 u_2)$ is a subgraph of $\mathcal{G}'(u)$ (in the strict sense), this is why we can get a derivation in $\mathcal{SP}$ rather than in $\mathcal{SP}^X$.

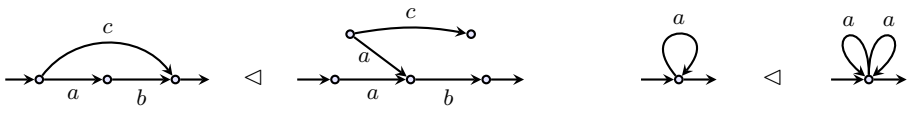
Theorem 25 follows like in the SP case, by two inductions. The base case for letters is slightly more involved in the presence of 1; we give all details in [10, Appendix D].

▶ **Corollary 27.** *The (in)equational theories of sl-monoids, bsl-semigroups, and bsl-monoids are decidable in polynomial time.*

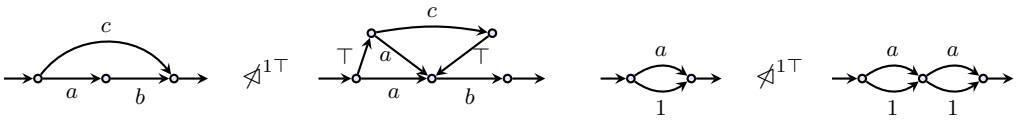
Proof. The existence of an X-homomorphism from G to H is equivalent to the existence of a homomorphism from G to an appropriate closure of H. The graph of a term can obviously be computed in polynomial time, as well as its closure. The graph-homomorphism problem can be solved in polynomial time when the source graph has bounded treewidth [12, 7, 16], which is the case here (series-parallel graphs have treewidth at most two). ◀

As an example, consider the following homomorphisms, establishing two valid laws of $\mathcal{R}el$:

$$\mathcal{R}el \models ab \cap c \leq (a \cap \top(a \cap c\top))b \qquad \mathcal{R}el \models 1 \cap a \leq (1 \cap a)(1 \cap a)$$



Those are not laws of bsl-monoids: under the simpler interpretation $\mathcal{G}'$, we obtain the following pairs of graphs, which are not related by $\lhd^{1\top}$.



7 Conclusion

We have shown that on the signatures SP1, SP⊤ and SP1⊤, $\mathcal{R}el$ is not finitely axiomatisable with a set of *(in)equations*. Does this change if we are more flexible on the shape of axioms? For example if Horn sentences or first-order formulas are allowed as axioms?

We have given a necessary condition on signatures for $\mathcal{R}el$ to be finitely based. This is the decomposability condition given in Proposition 8. Can we obtain a full characterisation?

References

1 H. Andréka and D.A. Bredikhin. The equational theory of union-free algebras of relations. *Algebra Universalis*, 33(4):516–532, 1995. doi:10.1007/BF01225472.

2 Hajnal Andréka and Szabolcs Mikulás. Axiomatizability of positive algebras of binary relations. *Algebra universalis*, 66(7), 2011. An erratum appears at http://www.dcs.bbk.ac.uk/~szabolcs/AM-AU-err6.pdf. doi:10.1007/s00012-011-0142-3.

3 D. A. Bredihin and B. M. Schein. Representations of ordered semigroups and lattices by binary relations. *Colloquium Mathematicae*, 39(1):1–12, 1978. URL: http://eudml.org/doc/266458.

4 D. A. Bredikhin. The equational theory of relation algebras with positive operations. *Izv. Vyssh. Uchebn. Zaved. Mat.*, 37(3):23–30, 1993. In Russian. URL: http://mi.mathnet.ru/ivm4374.

5 Paul Brunet. The equational theory of algebras of languages. Talk at RAMiCS, Lyon, May 2017 (Special session on mechanised reasoning), 2017.

6 Paul Brunet and Damien Pous. Petri automata for Kleene allegories. In *LICS*, pages 68–79. ACM, 2015. doi:10.1109/LICS.2015.17.

7 Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theoretical Computer Science*, 239(2):211–229, 2000. doi:10.1016/S0304-3975(99)00220-0.

8 R. Diestel. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2005.

9 Amina Doumane and Damien Pous. Completeness for identity-free Kleene lattices. In *CONCUR*, volume 118 of *LIPIcs*, pages 18:1–18:17. Schloss Dagstuhl, 2018. doi:10.4230/LIPIcs.CONCUR.2018.18.

10 Amina Doumane and Damien Pous. Full version of this paper, with appendices, 2020. URL: https://hal.archives-ouvertes.fr/hal-02870687.

11 Z. Ésik and L. Bernátsky. Equational properties of Kleene algebras of relations with conversion. *Theoretical Computer Science*, 137(2):237–251, 1995. doi:10.1016/0304-3975(94)00041-G.

12 Eugene C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *NCAI*, pages 4–9. AAAI Press / The MIT Press, 1990. URL: http://www.aaai.org/Library/AAAI/1990/aaai90-001.php.

13 P. Freyd and A. Scedrov. *Categories, Allegories*. North Holland, 1990.

14 Jay L. Gischer. The equational theory of pomsets. *Theoretical Computer Science*, 61(2):199–224, 1988. doi:10.1016/0304-3975(88)90124-7.

15 J. Grabowski. On partial languages. *Fundamenta Informaticae*, 4:427–498, 1981.

16 Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *Journal of the ACM*, 54(1):1:1–1:24, 2007. doi:10.1145/1206035.1206036.

17 T. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene algebra and its foundations. *Journal of Logic and Algebraic Programming*, 80(6):266–296, 2011.

18 Ian Hodkinson and Szabolcs Mikulás. Axiomatizability of reducts of algebras of relations. *Algebra Universalis*, 43:127–156, 2000. doi:10.1007/s000120050150.

19 Roger Maddux. *Relation Algebras*. Elsevier, 2006.

20 Szabolcs Mikulás. Axiomatizability of algebras of binary relations. In *Classical and New Paradigms of Computation and their Complexity Hierarchies*, pages 187–205. Springer Netherlands, 2004. doi:10.1007/978-1-4020-2776-5_11.

21 Donald Monk. On representable relation algebras. *The Michigan mathematical journal*, 31(3):207–210, 1964. doi:10.1307/mmj/1028999131.

22 Yoshiki Nakamura. Partial derivatives on graphs for kleene allegories. In *Lics*, pages 1–12. IEEE, 2017. doi:10.1109/LICS.2017.8005132.

23 Damien Pous and Valeria Vignudelli. Allegories: decidability and graph homomorphisms. In *LiCS*, pages 829–838. ACM, 2018. doi:10.1145/3209108.3209172.

24 V. Pratt. Origins of the calculus of binary relations. In *LICS*, pages 248–254. IEEE Computer Society Press, 1992.

25 A. Tarski. On the calculus of relations. *J. of Symbolic Logic*, 6(3):73–89, 1941.

26 A. Tarski and S. Givant. *A Formalization of Set Theory without Variables*, volume 41 of *Colloquium Publications*. American Mathematical Society, Providence, Rhode Island, 1987.

Decidability and Synthesis of Abstract Inductive Invariants

Francesco Ranzato
Dipartimento di Matematica, University of Padova, Italy
https://www.math.unipd.it/~ranzato
francesco.ranzato@unipd.it

Abstract

Decidability and synthesis of inductive invariants ranging in a given domain play an important role in software verification. We consider here inductive invariants belonging to an abstract domain A as defined in abstract interpretation, namely, ensuring the existence of the best approximation in A of any system property. In this setting, we study the decidability of the existence of abstract inductive invariants in A of transition systems and their corresponding algorithmic synthesis. Our model relies on some general results which relate the existence of abstract inductive invariants with least fixed points of best correct approximations in A of the transfer functions of transition systems and their completeness properties. This approach allows us to derive decidability and synthesis results for abstract inductive invariants which are applied to the well-known Karr's numerical abstract domain of affine equalities. Moreover, we show that a recent general algorithm for synthesizing inductive invariants in domains of logical formulae can be systematically derived from our results and generalized to a range of algorithms for computing abstract inductive invariants.

2012 ACM Subject Classification Theory of computation → Invariants; Theory of computation → Abstraction

Keywords and phrases Inductive invariant, program verification, abstract interpretation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.30

Funding *Francesco Ranzato*: Partially funded by *University of Padova*, under the SID2018 project "Analysis of STatic Analyses (ASTA)"; *Italian Ministry of University and Research*, under the PRIN2017 project no. 201784YSZ5 "AnalysiS of PRogram Analyses (ASPRA)"; *Facebook Research*, under a "Probability and Programming Research Award".

Acknowledgements I thank Patrick Cousot for comments and suggestions on an earlier version of this paper.

1 Introduction

Proof and inference methods based on inductive invariants are widespread in automatic (or semi-automatic) program and system verification (see, *e.g.*, [2,5,8,9,10,11,18,19,22,25,28,32]). The inductive invariant proof method roots at the works of Floyd [13], Park [29,30], Naur [27] and Manna et al. [20]. Given a transition system $\mathcal{T} = \langle \Sigma, \tau, \Sigma_0 \rangle$, where τ is a transition relation on states ranging in Σ and $\Sigma_0 \subseteq \Sigma$ is a subset of initial states, together with a safety property $P \subseteq \Sigma$ to check, let us recall that a property $I \in \wp(\Sigma)$ is an *inductive invariant* for $\langle \mathcal{T}, P \rangle$ when: $\Sigma_0 \subseteq I$, *i.e.* the initial states satisfy I; $I \subseteq P$, *i.e.* I entails P; $\tau(I) \subseteq I$, *i.e.* I is inductive. The *inductive invariant principle* states that P holds for all the reachable states of $\mathcal{T}$ iff there exists an inductive invariant I for $\langle \mathcal{T}, P \rangle$. In such an explicit form this principle has been probably first formulated in 1982 by Cousot and Cousot [5, Section 5] and called "induction principle for invariance proofs". In most cases, verification and inference methods rely on inductive invariants I that range in some restricted domain $\mathcal{A} \subseteq \wp(\Sigma)$, such as a domain of logical formulae (*e.g.*, some separation logic or a fragment of first-order logic [28]) or a domain of abstract interpretation [3, 4] (*e.g.*, numerical abstract domains of affine relations or convex polyhedra). In this context, if an inductive invariant I belongs to $\mathcal{A}$ then I is called an *abstract inductive invariant* (inductive $\mathcal{A}$-invariant in [32, Section 1]).

Main Contributions. Our primary goal was to investigate whether and how the inductive invariant principle can be adapted when inductive invariants are restricted to range in an abstract domain $\mathcal{A}$. We make the following working assumption: $\mathcal{A} \subseteq \wp(\Sigma)$ is an abstract domain as defined in abstract interpretation [3, 4]. This means that each state property $X \in \wp(\Sigma)$ has a *best* over-approximation (w.r.t. $\subseteq$) $\alpha_{\mathcal{A}}(X)$ in $\mathcal{A}$ and each state transition relation τ has a *best* correct approximation $\tau^{\mathcal{A}}$ on the abstract domain $\mathcal{A}$. Under these hypotheses, we prove an *abstract inductive invariant principle* stating that there exists an abstract inductive invariant in $\mathcal{A}$ proving a property P of a transition system $\mathcal{T}$ iff the *best abstraction* $\mathcal{T}^{\mathcal{A}}$ in $\mathcal{A}$ of the system $\mathcal{T}$ allows us to prove P. The decidability/undecidability question of the existence of abstract inductive invariants in some abstract domain $\mathcal{A}$ for some class of transition systems has been recently investigated in a few significant cases [12, 16, 24, 31, 32]. We show how the abstract inductive invariant principle allows us to derive a general decidability result on the existence of inductive invariants in some abstract domain $\mathcal{A}$ and to design a general algorithm for synthesizing the least (w.r.t. the order of $\mathcal{A}$) abstract inductive invariant in $\mathcal{A}$, when this exists, by a least fixpoint computation in $\mathcal{A}$.

We also show a related result which is of independent interest in abstract interpretation: the (concrete) inductive invariant principle for a system $\mathcal{T}$ is equivalent to the abstract inductive invariant principle for $\mathcal{T}$ on an abstract domain $\mathcal{A}$ iff *fixpoint completeness* of $\mathcal{T}$ on $\mathcal{A}$ holds, *i.e.*, the best abstraction in $\mathcal{A}$ of the reachable states of $\mathcal{T}$ coincides with the reachable states of the best abstraction $\mathcal{T}^{\mathcal{A}}$ of $\mathcal{T}$ on $\mathcal{A}$.

The decidability/synthesis of abstract inductive invariants in a domain $\mathcal{A}$ for some class $\mathcal{C}$ of systems essentially boils down to prove that the best correct approximation $\tau^{\mathcal{A}}$ in $\mathcal{A}$ of the transition relation τ of systems in $\mathcal{C}$ is algorithmically computable. As case study, we provide one such result for Karr's affine relationships [17], which is a well-known and widely used abstract domain in numerical program analysis [21]. As a second application, we design an inductive invariant synthesis algorithm which, by generalizing an algorithm by Padon et al. [28] tailored for logical invariants, outputs the most abstract (*i.e.*, weakest/greatest) inductive invariant in a domain $\mathcal{A}$ which satisfies some suitable hypotheses. In particular, we show that this synthesis algorithm is obtained by instantiating a concrete co-inductive greatest fixpoint checking algorithm by Cousot [1] to a domain $\mathcal{A}$ of abstract invariants which is *disjunctive*, *i.e.*, abstract least upper bounds of $\mathcal{A}$ do not lose precision. This generalization allows us to design further related co-inductive algorithms for synthesizing abstract inductive invariants.

Due to lack of space in the main body of the paper, the proofs are moved to Section A.2 in Appendix A.

2 Background

2.1 Order Theory

If X is a subset of some universe set U then $\neg X$ denotes the complement of X with respect to U when U is implicitly given by the context. If $f : X \to Y$ is a function between sets and $S \in \wp(X)$ then $f(S) \triangleq \{f(x) \in Y \mid x \in S\}$ denotes the image of f on S. If $\vec{x} \in X^n$ is a vector in a product domain, $j \in [1, n]$ and $y \in X$ then $\vec{x}[x_j/y]$ denotes the vector obtained from $\vec{x}$ by replacing its j-th component x_j with y. To keep the notation simple and compact, we use the same symbol for a function/relation and its componentwise (*i.e.* pointwise) extension on product domains, *e.g.*, if $\vec{S}, \vec{T} \in \wp(X)^n$ then $\vec{S} \subseteq \vec{T}$ denotes that for all $i \in [1, n]$, $\vec{S}_i \subseteq \vec{T}_i$. Sometimes, to emphasize a pointwise definition, a dotted notation can be used such as in $f \dot{\leq} g$ for the pointwise ordering between functions.

A quasiordered set (or poset) $D_{\leq}$ satisfies the ascending (resp. descending) chain condition (ACC, resp. DCC) if D contains no countably infinite sequence of distinct elements $\{x_i\}_{i\in\mathbb{N}}$ such that, for all $i \in \mathbb{N}$, $x_i \leq x_{i+1}$ (resp. $x_{i+1} \leq x_i$). A poset is a directed-complete partial order (CPO) if it has the least upper bound (lub) of all its directed subsets. A complete lattice is a poset having the lub of all its arbitrary (possibly empty) subsets (and therefore having arbitrary glbs). In a complete lattice (or CPO), $\vee$ (or $\sqcup$) and $\wedge$ (or $\sqcap$) denote, resp., lub and glb, and $\bot$ and $\top$ denote, resp., least and greatest element.

Let $P_{\leq}$ be a poset and $f : P \to P$. Then, $\mathrm{Fix}(f) \triangleq \{x \in P \mid f(x) = x\}$, $\mathrm{Fix}^{\leq}(f) \triangleq \{x \in P \mid f(x) \leq x\}$, $\mathrm{Fix}^{\geq}(f) \triangleq \{x \in P \mid f(x) \geq x\}$, and $\mathsf{lfp}(f)$, $\mathsf{gfp}(f)$ denote, resp., the least and greatest fixpoint in $\mathrm{Fix}(f)$, when they exist. Let us recall Knaster-Tarski fixpoint theorem: if $\langle C, \leq, \vee, \wedge\rangle$ is a complete lattice and $f : C \to C$ is monotonic then $\langle \mathrm{Fix}(f), \leq\rangle$ is a complete lattice, $\mathsf{lfp}(f) = \wedge \mathrm{Fix}^{\leq}(f)$ and $\mathsf{gfp}(f) = \vee \mathrm{Fix}^{\geq}(f)$. Also, Knaster-Tarski-Kleene fixpoint theorem states that if $\langle C, \leq, \vee, \bot\rangle$ is a CPO with least element and $f : C \to C$ is Scott-continuous (*i.e.*, f preserves lubs of directed subsets) then $\mathsf{lfp}(f) = \vee_{i\in\mathbb{N}} f^i(\bot)$, where, for all $x \in C$ and $i \in \mathbb{N}$, $f^0(x) \triangleq x$ and $f^{i+1}(x) \triangleq f(f^i(x))$; dually, if $\langle C, \leq, \wedge, \top\rangle$ is a dual-CPO with greatest element and $f : C \to C$ is Scott-co-continuous then $\mathsf{gfp}(f) = \wedge_{i\in\mathbb{N}} f^i(\top)$. A function $f : C \to C$ on a complete lattice is additive when f preserves arbitrary lubs.

2.2 Abstract Domains

Let us recall some basic notions on closures and Galois connections which are commonly used in abstract interpretation [3, 4] to define abstract domains (see, *e.g.*, [21]). Closure operators and Galois connections are equivalent notions and are both used for defining the notion of approximation in abstract interpretation, where closure operators bring the advantage of defining abstract domains independently of a specific representation for abstract objects which is required by Galois connections.

An upper closure operator (uco), or simply upper closure, on a poset $C_{\leq}$ is a function $\mu : C \to C$ which is monotonic, idempotent and extensive (*i.e.*, $x \leq \mu(x)$ for all $x \in C$). Dually, a lower closure operator (lco) $\eta : C \to C$ is monotonic, idempotent and reductive (*i.e.*, $\eta(x) \leq x$ for all $x \in C$). The set of all upper/lower closures on $C_{\leq}$ is denoted by $\mathrm{uco}(C_{\leq})/\mathrm{lco}(C_{\leq})$. We write $c \in \mu(C)$, or simply $c \in \mu$, to denote that there exists $c' \in C$ such that $c = \mu(c')$, and we recall that this happens iff $\mu(c) = c$. In what follows, assume that $C_{\leq}$ is a complete lattice. Let us recall that $\langle \mu(C), \leq\rangle$ is closed under glb of arbitrary subsets and, conversely, $X \subseteq C$ is the image of some $\mu \in \mathrm{uco}(C)$ iff X is closed under glb of all its subsets, and in this case $\mu(c) = \wedge\{c' \in X \mid c \leq c'\}$ holds. Dually, $X \subseteq C$ is closed under arbitrary lub of its subsets iff X is the image a lower closure $\eta \in \mathrm{lco}(C)$, and in this case $\eta(c) = \vee\{c' \in X \mid c' \leq c\}$. In abstract interpretation, a closure $\mu \in \mathrm{uco}(C)$ on a concrete domain $C_{\leq}$ plays the role of an abstract domain having best approximations: $c \in C$ is (upper-)approximated by any $\mu(c')$ such that $c \leq \mu(c')$ and $\mu(c)$ is the best approximation of c in μ because $\mu(c) = \wedge\{\mu(c') \mid c' \in C,\, c \leq \mu(c')\}$.

A Galois Connection (GC, also called adjunction) between two posets $\langle C, \leq_C\rangle$, called concrete domain, and $\langle A, \leq_A\rangle$, called abstract domain, consists of two maps $\alpha : C \to A$ and $\gamma : A \to C$ such that $\alpha(c) \leq_A a \Leftrightarrow c \leq_C \gamma(a)$ holds. A GC is called Galois insertion (GI) when α is surjective or, equivalently, γ is injective. Any GC can be transformed into a GI simply by removing useless elements in $A \setminus \alpha(C)$ from the abstract domain A. A GC/GI is denoted by $(C_{\leq_C}, \alpha, \gamma, A_{\leq_A})$. GCs and ucos are equivalent notions because any GC $\mathcal{G} = (C, \alpha, \gamma, A)$ induces a closure $\mu_{\mathcal{G}} \triangleq \gamma \circ \alpha \in \mathrm{uco}(C)$, any $\mu \in \mathrm{uco}(C)$ induces a GI $\mathcal{G}_\mu \triangleq (C, \mu, \lambda x.x, \mu(C))$, and these two transforms are inverse of each other.

2.3 Transition Systems

Let $\mathcal{T} = \langle \Sigma, \tau \rangle$ be a transition system where Σ is a set of states and $\tau \subseteq \Sigma \times \Sigma$ is a transition relation inducing the following transformers of type $\wp(\Sigma) \to \wp(\Sigma)$:

$$\mathsf{pre}(X) \triangleq \{s \in \Sigma \mid \exists s' \in X.(s, s') \in \tau\} \qquad \widetilde{\mathsf{pre}}(X) \triangleq \{s \in \Sigma \mid \forall s'.(s, s') \in \tau \Rightarrow s' \in X\}$$
$$\mathsf{post}(X) \triangleq \{s' \in \Sigma \mid \exists s \in X.(s, s') \in \tau\} \qquad \widetilde{\mathsf{post}}(X) \triangleq \{s' \in \Sigma \mid \forall s.(s, s') \in \tau \Rightarrow s \in X\}$$

We will equivalently specify a transition system by one of the above transformers (typically post) in place of the transition relation τ. Let us also recall (see *e.g.* [6]) that $(\wp(\Sigma)_\subseteq, \mathsf{pre}, \widetilde{\mathsf{post}}, \wp(\Sigma)_\subseteq)$ and $(\wp(\Sigma)_\subseteq, \mathsf{post}, \widetilde{\mathsf{pre}}, \wp(\Sigma)_\subseteq)$ are GCs. The set of reachable states of $\mathcal{T}$ from a set of initial states $\Sigma_0 \subseteq \Sigma$ is $\mathrm{Reach}[\mathcal{T}, \Sigma_0] \triangleq \mathsf{lfp}(\lambda X \in \wp(\Sigma).\, \Sigma_0 \cup \mathsf{post}(X))$, and $\mathcal{T}$ satisfies a safety property $P \subseteq \Sigma$ when $\mathrm{Reach}[\mathcal{T}, \Sigma_0] \subseteq P$ holds.

2.4 Inductive Invariant Principle

Given a transition system $\mathcal{T} = \langle \Sigma, \tau \rangle$, a set of states $I \in \wp(\Sigma)$ is an *inductive invariant* for $\mathcal{T}$ w.r.t. $\langle \Sigma_0, P \rangle \in \wp(\Sigma)^2$ when: (i) $\Sigma_0 \subseteq I$; (ii) $\mathsf{post}(I) \subseteq I$; (iii) $I \subseteq P$. An inductive invariant I allows us to prove that $\mathcal{T}$ is safe, *i.e.* $\mathrm{Reach}[\mathcal{T}, \Sigma_0] \subseteq P$, by the *inductive invariant principle* (a.k.a. fixpoint induction principle), a consequence of Knaster-Tarski fixpoint theorem: If $C_\leq$ is a complete lattice, $c' \in C$ and $f : C \to C$ is monotonic then

$$\mathsf{lfp}(f) \leq c' \Leftrightarrow \exists i \in C.\, f(i) \leq i \wedge i \leq c' \tag{1}$$

In particular, given $c, c' \in C$, since $c \vee_C f(i) \leq i$ iff $c \leq i \wedge f(i) \leq i$, it turns out that:

$$\mathsf{lfp}(\lambda x.c \vee_C f(x)) \leq c' \Leftrightarrow \exists i \in C.\, c \leq i \wedge f(i) \leq i \wedge i \leq c' \tag{2}$$

One such $i \in C$ such that $c \leq i \wedge f(i) \leq i \wedge i \leq c$ is called an inductive invariant of f for $\langle c, c' \rangle$. Hence, (2) is applied to the function $\lambda X.\Sigma_0 \cup \mathsf{post}(X) : \wp(\Sigma) \to \wp(\Sigma)$, which is monotonic on $\wp(\Sigma)_\subseteq$, so that $\mathsf{lfp}(\lambda X.\Sigma_0 \cup \mathsf{post}(X)) \subseteq P$ holds iff there exists an inductive invariant I for $\mathcal{T}$ w.r.t. $\langle \Sigma_0, P \rangle$. In most contexts for defining transition systems, the decision problem of the existence of a (concrete) inductive invariant for a class of transition systems w.r.t. a set of initial states and some safety property turns out to be undecidable.

3 Abstract Inductive Invariants

An array of recent works, [12, 16, 24, 28, 31, 32] among the others, consider a notion of abstract inductive invariant and study the corresponding decidability/undecidability and synthesis problems. The common approach of these works consists in restricting the range of inductive invariants from a concrete domain C to some abstraction A_C of C, which, in a general setting, is simply a subset of C. Let us formalize abstract inductive invariants in order-theoretic terms. Given a class $\mathcal{C}$ of complete lattices and, for all $C \in \mathcal{C}$, a class of functions $\mathcal{F}_C \subseteq C \to C$, a set of initial properties $\mathrm{Init}_C \subseteq C$, a set of safety properties $\mathrm{Safe}_C \subseteq C$, and an abstract domain $A_C \subseteq C$, a first problem is the decidability of the following decision question:

$$\forall C \in \mathcal{C}.\forall f \in \mathcal{F}_C.\forall c \in \mathrm{Init}_C\, .\forall c' \in \mathrm{Safe}_C\, .\exists i \in^? A_C.\, c \leq i \wedge f(i) \leq i \wedge i \leq c' \tag{3}$$

where one such $i \in A_C$ is called an *abstract inductive invariant* for f and $\langle c, c' \rangle \in C^2$. Thakur et al. [32, Section 1] use the terminology "inductive A_C-invariant" when for some transition system $\langle \Sigma, \tau \rangle$, $f = \mathsf{post}_\tau$, $A_C \subseteq \wp(\Sigma)$ and $c' = \Sigma$.

The corresponding synthesis problem consists in designing algorithms which output abstract inductive invariants in A_C or notify that no inductive invariant in A_C exists.

Given $\mathcal{T} = \langle \Sigma, \tau \rangle$ whose successor transformer is post, the problem (3) is instantiated to $C_{\leq} = \wp(\Sigma)_{\subseteq}$, $f = \mathsf{post}(X)$, $c = \Sigma_0 \in \wp(\Sigma)$ set of initial states and $c' = P \in \wp(\Sigma)$ safety property. When $\mathcal{T}$ is the control flow graph generated by some program, Σ_0 are the states of some initial control node and P is a safety property given by the states which are not in some bad control node, abstract inductive invariants are called *separating invariants* and the decision problem (3) is called *Monniaux problem* by Fijalkow et al. [12], because this problem was first formulated by Monniaux [23, 24].

3.1 Abstract Inductive Invariant Principle

Our working assumption is that in problem (3) the invariants i range in an abstract domain A as dictated by abstract interpretation [3, 4].

▶ **Assumption 3.1.** $\langle A, \leq_A \rangle$ is an abstract domain of the complete lattice $\langle C, \leq_C \rangle$ which has best approximations, *i.e.*, one of these two equivalent assumptions is satisfied:

(i) $(C_{\leq_C}, \alpha, \gamma, A_{\leq_A})$ is a Galois insertion;

(ii) $\langle A, \leq_A \rangle = \langle \mu(C), \leq_C \rangle$ for some upper closure $\mu \in \mathrm{uco}(C_{\leq_C})$. ⌟

Under Assumption 3.1, let us recall that if $f : C \to C$ is a concrete monotonic function then the mappings $\alpha f \gamma : A \to A$, for the case of GIs, and $\mu f : \mu(C) \to \mu(C)$, for the case of ucos, are called *best correct approximation* (bca) in A of f. This is justified by the observation that an abstract function $f^\sharp : A \to A$ (or $f^\sharp : \mu(C) \to \mu(C)$ for ucos) is a correct (or sound) approximation of f when $\alpha f \gamma \mathrel{\dot{\leq}_A} f^\sharp$ (or $\mu f \mathrel{\dot{\leq}_C} f^\sharp$ for ucos) holds. Our first result is an *abstract* inductive invariant principle which restricts the invariants of f in (1) to those ranging in an abstract domain A: when the abstract domain A is specified by a GI, this means that $a \in A$ is an abstract invariant of f when $f\gamma(a) \leq_C \gamma(a)$ holds; when the abstract domain is a closure $\mu \in \mathrm{uco}(C)$, this means that $a \in \mu \subseteq C$ is an abstract invariant of f when $fa \leq_C a$ holds.

▶ **Lemma 3.2 (Abstract Inductive Invariant Principle).** *Let $(C_{\leq_C}, \alpha, \gamma, A_{\leq_A})$ be a GI. For all $c' \in C$ and $a' \in A$:*

(a) $\gamma(\mathsf{lfp}(\alpha f \gamma)) \leq_C c' \Leftrightarrow \exists a \in A.\ f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C c'$;

(b) $\mathsf{lfp}(\alpha f \gamma) \leq_A a' \Leftrightarrow \exists a \in A.\ f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C \gamma(a')$.

It is worth stating Lemma 3.2 (a) in an equivalent form for an abstract domain represented by a closure $\mu \in \mathrm{uco}(C)$: $\mathsf{lfp}(\mu f) \leq_C c' \Leftrightarrow \exists a \in \mu.\ fa \leq_C a \wedge a \leq_C c'$.

Let us observe that point (b) is an easy consequence of point (a), because, by surjectivity of α in GIs, for all $a' \in A$, there exists some $c' \in C$ such that $a' = \alpha(c')$, and $\gamma(\mathsf{lfp}(\alpha f \gamma)) \leq_C \gamma(\alpha(c')) \Leftrightarrow \mathsf{lfp}(\alpha f \gamma) \leq_A \alpha(c')$ holds. Moreover, point (b) easily follows from the inductive invariant principle (1) for the bca $\alpha f \gamma : A \to A$. On the other hand, it is worth remarking that point (a) cannot be obtained from (b), *i.e.* (a) is *strictly stronger* than (b), because (a) allows us to prove concrete properties $c' \in C$ which are not exactly represented by A (*i.e.*, $c' \notin \gamma(A)$) by abstract inductive invariants in A, as shown by the following tiny example.

▶ **Example 3.3.** Consider a 4-points chain $C = \{1 < 2 < 3 < 4\}$, the function $f : C \to C$ defined by $\{1 \mapsto 1;\ 2 \mapsto 2;\ 3 \mapsto 4;\ 4 \mapsto 4\}$, and the abstraction $A = \{2, 4\}$ with $\gamma = \mathrm{id}$ and $\alpha = \{1 \mapsto 2;\ 2 \mapsto 2;\ 3 \mapsto 4;\ 4 \mapsto 4\}$. Here, we have that $\alpha f \gamma = \{2 \mapsto 2;\ 4 \mapsto 4\}$ and $\mathsf{lfp}(\alpha f \gamma) = 2$. In this case, Lemma 3.2 (b) allows us to prove all the abstract properties $a' \in A$ by abstract inductive invariants, while Lemma 3.2 (a) allows us to prove an additional

concrete property $3 \in C \smallsetminus \gamma(A)$, which is not exactly represented by A, by an abstract inductive invariant, and this would not be possible by resorting to Lemma 3.2 (b). Also, $\gamma(\mathsf{lfp}(\alpha f \gamma)) \not\leq 1$ holds, thus, by Lemma 3.2 (a), the concrete property $1 \in C \smallsetminus \gamma(A)$ cannot be proved by an abstract inductive invariant in A, whereas Lemma 3.2 (b) does not allow us to infer this. ⌟

Lemma 3.2 (b) tells us that the existence of an abstract inductive invariant of f proving an abstract property $a' \in A$ is equivalent to the fact that the least fixpoint of the bca $\alpha f \gamma$ entails a'. This formalizes for an abstract domain satisfying Assumption 3.1 an observation in [12, Section 1] stating (in our terminology) that "*the existence of some abstract inductive invariant for $\alpha f \gamma$ proving a' is equivalent to whether the strongest abstract invariant $\mathsf{lfp}(\alpha f \gamma)$ entails a'*", *i.e.* is inductive, and generalizes [32, Observation 1] stating (in our terminology) that "$\mathsf{lfp}(\alpha f \gamma)$ *is the strongest abstract inductive invariant*". If, instead, we aim at proving *any* concrete property $c' \in C$, possibly not in $\gamma(A)$, by an abstract inductive invariant then Lemma 3.2 (a) states that this is equivalent to the strictly stronger condition $\gamma(\mathsf{lfp}(\alpha f \gamma)) \leq_C c'$.

As a consequence of Lemma 3.2 (a) we derive the following characterization of the problem (3).

▶ **Corollary 3.4.** *Let $\mathcal{F} \subseteq C \to C$ and $\mathrm{Init}, \mathrm{Safe} \subseteq C$. The Monniaux decision problem $\forall f \in \mathcal{F}. \forall c \in \mathrm{Init}. \forall c' \in \mathrm{Safe}. \exists^? a \in A.\ c \leq_C \gamma(a) \wedge f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C c'$ is decidable iff the decision problem $\forall f \in \mathcal{F}. \forall c \in \mathrm{Init}. \forall c' \in \mathrm{Safe}. \gamma(\mathsf{lfp}(\lambda x \in A. \alpha(c) \vee_A \alpha f \gamma(x))) \leq^?_C c'$ is decidable.*

Moreover, as a consequence of Lemma 3.2 (b) we obtain the following abstract invariant synthesis algorithm.

▶ **Corollary 3.5.** *Assume that the lub $\vee_A : A \times A \to A$ and the bca $\alpha f \gamma : A \to A$ are finitely computable, the partial order $\leq_A$ is decidable and A is an ACC CPO with least element. For all $c \in C$ such that $\alpha(c)$ is finitely computable and $a' \in A$, the following procedure:*

$$\begin{array}{l} \textsc{AInv}(f, A, c, a') \triangleq\ i := \alpha(c); \\ \qquad\qquad \textbf{while } i \leq_A a' \textbf{ do } \{\textbf{if } \alpha f \gamma(i) \leq_A i \textbf{ return } i;\ \textbf{else } i := \alpha f \gamma(i);\} \\ \qquad\qquad \textbf{return } \textit{no abstract inductive invariant for } f \textit{ and } \langle c, \gamma(a')\rangle; \end{array}$$

is a terminating algorithm which outputs the least abstract inductive invariant for f and $\langle c, \gamma(a')\rangle$, when one such abstract inductive invariant exists, otherwise outputs "no abstract inductive invariant".

Under the same hypotheses for the abstract domain A, Thakur et al. [32, Observation 2] state (in our terminology) that the problem of computing the least abstract inductive invariant in A for some successor transformer post_τ reduces to the problem of computing the best correct approximation $\alpha\, \mathsf{post}_\tau\, \gamma$.

4 Fixpoint Completeness and Abstract Inductive Invariants

4.1 Completeness in Abstract Interpretation

Soundness in abstract interpretation (or, more in general, in static analysis) is a mandatory requirement stating that no false negative can occur: if $f : C \to C$ and $f^\sharp : A \to A$ are the concrete and abstract monotonic transformers then *fixpoint soundness* means that $\alpha(\mathsf{lfp}(f)) \leq_A \mathsf{lfp}(f^\sharp)$ holds, so that a positive abstract proof $\mathsf{lfp}(f^\sharp) \leq_A a'$ entails that $\gamma(a')$ concretely holds, *i.e.*, $\mathsf{lfp}(f) \leq_C \gamma(a')$. Fixpoint soundness is usually proved as a consequence

of *pointwise soundness*: if $f^\sharp$ is a pointwise correct approximation of f, *i.e.* $\alpha f \dot{\leq}_A f^\sharp\alpha$, then $\alpha(\mathsf{lfp}(f)) \leq_A \mathsf{lfp}(f^\sharp)$ holds. While soundness is indispensable, completeness in abstract interpretation encodes an ideal situation where no false positives (also called false alarms) arise: *fixpoint completeness* means that $\alpha(\mathsf{lfp}(f)) = \mathsf{lfp}(f^\sharp)$ holds, so that $\mathsf{lfp}(f^\sharp) \not\leq_A a'$ entails $\mathsf{lfp}(f) \not\leq_C \gamma(a')$. One can also consider a *strong fixpoint completeness* requiring that $\mathsf{lfp}(f) = \gamma(\mathsf{lfp}(f^\sharp))$, so that $\mathsf{lfp}(f^\sharp) \not\leq_A \alpha(c')$ entails $\mathsf{lfp}(f) \not\leq_C c'$. However, it should be remarked that $\mathsf{lfp}(f) = \gamma(\mathsf{lfp}(f^\sharp))$ is much stronger than $\alpha(\mathsf{lfp}(f)) = \mathsf{lfp}(f^\sharp)$ since it means that the concrete lfp is precisely represented by the abstract lfp.

It is important to remark that if $f^\sharp$ is a pointwise correct approximation of f and fixpoint completeness for $f^\sharp$ holds then since $\alpha(\mathsf{lfp}(f)) \leq_A \mathsf{lfp}(\alpha f\gamma) \leq_A \mathsf{lfp}(f^\sharp)$ always holds, one obtains that $\alpha(\mathsf{lfp}(f)) = \mathsf{lfp}(\alpha f\gamma) = \mathsf{lfp}(f^\sharp)$ holds, namely, the bca $\alpha f\gamma$ is fixpoint complete as well. This means that the possibility (and therefore impossibility) of defining an approximate transformer $f^\sharp : A \to A$ on A which is fixpoint complete does not depend on the specific definition of $f^\sharp$ but is instead an intrisic *property of the abstract domain* A w.r.t. the concrete transformer f, as formalized by the equation $\alpha(\mathsf{lfp}(f)) = \mathsf{lfp}(\alpha f\gamma)$. Moreover, fixpoint completeness is typically proved as a by-product of *pointwise completeness* $\alpha f = f^\sharp\alpha$, and if $f^\sharp$ is pointwise complete then it turns out that $f^\sharp = \alpha f\gamma$, that is, $f^\sharp$ actually is the bca of f. This justifies why, without loss of generality, we can consider fixpoint and pointwise completeness of bca's $\alpha f\gamma$ only, *i.e.*, as properties of abstract domains [14, 15].

4.2 Characterizing Fixpoint Completeness by Abstract Inductive Invariants

We show that the abstract inductive invariant principle is closely related to fixpoint completeness. More precisely, we provide an answer to the following question: in the abstract inductive invariant principle as stated by Lemma 3.2, can we replace $\mathsf{lfp}(\alpha f\gamma)$ with $\alpha(\mathsf{lfp}(f))$? This question is settled by the following result.

▶ **Theorem 4.1.** *Let* $(C_{\leq_C}, \alpha, \gamma, A_{\leq_A})$ *be a GI.*

(a) $\mathsf{lfp}(f) = \gamma(\mathsf{lfp}(\alpha f\gamma))$ *iff* $\forall c' \in C.\big(\mathsf{lfp}(f) \leq_C c' \Leftrightarrow \exists a \in A.\, f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C c'\big)$;

(b) $\alpha(\mathsf{lfp}(f)) = \mathsf{lfp}(\alpha f\gamma)$ *iff* $\forall a' \in A.\big(\mathsf{lfp}(f) \leq_C \gamma(a') \Leftrightarrow \exists a \in A.\, f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C \gamma(a')\big)$.

Theorem 4.1 (b) can be stated by means of ucos as follows: if $\mu \in \mathrm{uco}(C)$ then $\mu(\mathsf{lfp}(f)) = \mathsf{lfp}(\mu f)$ iff $\forall a' \in \mu.(\mathsf{lfp}(f) \leq_C a' \Leftrightarrow \exists a \in \mu.\, fa \leq_C a \wedge a \leq_C a')$.

The above result can be read as follows. Since, by the inductive invariant principle (1), $\mathsf{lfp}(f) \leq_C c'$ iff there exists a concrete inductive invariant proving c', it turns out that Theorem 4.1 (a) states that the existence of an *abstract* inductive invariant proving c' is equivalent to the existence of *any* inductive invariant proving c' iff fixpoint completeness holds. In other terms, the (concrete) inductive invariant principle is equivalent to the abstract inductive invariant principle iff fixpoint completeness holds. This result is of independent interest in abstract interpretation, since it provides a new characterization of the key property of fixpoint completeness of abstract domains.

A further interesting characterization of fixpoint completeness is as follows.

▶ **Lemma 4.2.** $\alpha(\mathsf{lfp}(f)) = \mathsf{lfp}(\alpha f\gamma) \Leftrightarrow \exists a \in A. f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C \gamma\alpha(\mathsf{lfp}(f))$.

As a consequence, fixpoint completeness for f does not hold in A iff the abstract property $\alpha(\mathsf{lfp}(f)) \in A$ cannot be proved by an abstract inductive invariant in A

▶ **Example 4.3.** Consider a 3-points chain $C = \{1 < 2 < 3\}$ and the monotonic concrete function $f : C \to C$ defined by $f = \{1 \mapsto 1;\ 2 \mapsto 3;\ 3 \mapsto 3\}$.

Consider the uco $\mu = \{2,3\}$, *i.e.*, $\mu = \{1 \mapsto 2;\ 2 \mapsto 2;\ 3 \mapsto 3\}$, so that $\mu f = \{1 \mapsto 2;\ 2 \mapsto 3;\ 3 \mapsto 3\}$. Fixpoint completeness does not hold because $\mu(\mathsf{lfp}(f)) = \mu(1) = 2 < 3 = \mathsf{lfp}(\mu f)$. Thus, in accordance with Lemma 4.2, it turns out that $\mu(\mathsf{lfp}(f)) = 2$ cannot be inductively proved in the abstraction μ. In fact, $f(2) \not\leq 2$, while $f(3) \leq 3$ but $3 \not\leq \mu(\mathsf{lfp}(f))$.
Consider now the uco $\eta = \{1,3\}$, *i.e.*, $\eta = \{1 \mapsto 1;\ 2 \mapsto 3;\ 3 \mapsto 3\}$, so that $\eta f = \{1 \mapsto 1;\ 2 \mapsto 3;\ 3 \mapsto 3\}$. Here, $\eta(\mathsf{lfp}(f)) = \eta(1) = 1 = \mathsf{lfp}(\eta f)$, therefore fixpoint completeness holds. Thus, by the uco version of Theorem 4.1 (b), any valid abstract invariant of f can be inductively proved: in fact, $1, 3 \in \eta$ are valid abstract invariants of f and are both inductive. ⌟

Due to lack of space, we moved to Appendix A.1 an application of Theorem 4.1 which provides a model showing how the "Safety $=^?$ Abstract Invariance" problem is related to fixpoint completeness in abstract interpretation, as informally hinted by Padon et al. [28].

5 Abstract Inductive Invariants of Nondeterministic Programs

We consider transition systems as represented by a control flow graph (CFG) of a possibly nondeterministic imperative program. A program is a tuple $\mathcal{P} = \langle Q, n, \mathbb{V}, \mathsf{T}, \rightarrow \rangle$ where Q is a finite set of control nodes (or program points), $n \in \mathbb{N}$ is the number of program variables of type $\mathbb{V}$ (*e.g.*, $\mathbb{V} = \mathbb{Z}, \mathbb{Q}, \mathbb{R}$), T is a finite set of (possibly nondeterministic) transfer functions of type $\mathbb{V}^n \to \wp(\mathbb{V}^n)$, $\rightarrow \subseteq Q \times \mathsf{T} \times Q$ is a (possibly nondeterministic) control flow relation, where $q \xrightarrow{t} q'$ denotes a flow transition with transfer function $t \in \mathsf{T}$. A program $\mathcal{P}$ therefore defines a transition system $\mathcal{T}_{\mathcal{P}} = \langle \Sigma, \tau \rangle$ where $\Sigma \triangleq Q \times \mathbb{V}^n$ is the set of states and the transition relation $\tau \subseteq \Sigma \times \Sigma$ is defined by $\langle (q, \vec{v}), (q', \vec{v}') \rangle \in \tau \overset{\triangle}{\Leftrightarrow} \exists t \in \mathsf{T}.\ q \xrightarrow{t} q' \wedge \vec{v}' \in t(\vec{v})$. The transfer functions in T include assignments and Boolean guards, where if $b \in \wp(\mathbb{V}^n)$ is a deterministic Boolean predicate (such as $x_1 + 2x_2 - 1 = 0$) then the corresponding transfer function $t_b : \mathbb{V}^n \to \wp(\mathbb{V}^n)$ is $t_b(\vec{v}) \triangleq$ **if** $\vec{v} \in b$ **then** $\{\vec{v}\}$ **else** $\varnothing$. Examples of transfer functions include: affine, polynomial, nondeterministic assignments and affine equalities guards. The next value transformer $\mathsf{post}_{\langle q,q' \rangle} : \wp(\mathbb{V}^n) \to \wp(\mathbb{V}^n)$ for a pair $\langle q, q' \rangle \in Q \times Q$ of control nodes is $\mathsf{post}_{\langle q,q' \rangle}(X) \triangleq \cup\{t(X) \in \wp(\mathbb{V}^n) \mid \exists t \in \mathsf{T}. q \xrightarrow{t} q'\}$. The complete lattice $\langle \wp(\Sigma), \subseteq \rangle$ of sets of states can be equivalently represented by the Q-indexed product lattice $\langle \wp(\mathbb{V}^n)^{|Q|}, \dot{\subseteq} \rangle$. Hence, the successor transformer $\mathsf{post}_{\mathcal{P}} : \wp(\mathbb{V}^n)^{|Q|} \to \wp(\mathbb{V}^n)^{|Q|}$ and the set of reachable states from $\Sigma_0 \in \wp(\mathbb{V}^n)^{|Q|}$ are defined as follows:

$$\mathsf{post}_{\mathcal{P}}(\langle X_q \rangle_{q \in Q}) \triangleq \langle \cup_{q \in Q} \mathsf{post}_{\langle q,q' \rangle}(X_q) \rangle_{q' \in Q} \qquad \mathrm{Reach}[\mathcal{P}, \Sigma_0] \triangleq \mathsf{lfp}(\lambda \vec{X}.\Sigma_0 \cup \mathsf{post}_{\mathcal{P}}(\vec{X}))$$

For all control nodes $q \in Q$ and vectors $\vec{X} \in \wp(\mathbb{V}^n)^{|Q|}$, we will also use $\pi_q(\vec{X})$ and $\vec{X}_q$ to denote the q-indexed component of $\vec{X}$, *e.g.*, $\mathrm{Reach}[\mathcal{P}, \Sigma_0]_q \in \wp(\mathbb{V}^n)$ will be the set of reachable values at control node q.

We are interested in decidability and synthesis of abstract inductive invariants ranging in an abstract domain A as specified by a GI $(\wp(\mathbb{V}^n)_{\subseteq}, \alpha, \gamma, A_{\leq_A})$ parametric on $n \in \mathbb{N}$. By Corollary 3.4, for a given class $\mathcal{C}$ of programs, a class Init of sets of initial states and a class Safe of sets of safety properties, the Monniaux problem (3) is decidable iff for all $\mathcal{P} = \langle Q, n, \mathbb{V}, \mathsf{T}, \rightarrow \rangle \in \mathcal{C}$, $\Sigma_0 \in \mathrm{Init}$ and $P \in \mathrm{Safe}$,

$$\dot{\gamma}(\mathsf{lfp}(\lambda \vec{a} \in A^{|Q|}.\ \dot{\alpha}(\Sigma_0)\, \dot{\vee}_A\, \dot{\alpha}(\mathsf{post}_{\mathcal{P}}(\dot{\gamma}(\vec{a}))))) \dot{\subseteq}^? P \tag{4}$$

is decidable. Moreover, Corollary 3.5 provides an abstract inductive invariant synthesis algorithm AInv for safety properties represented by A (*i.e.*, $P \in \gamma(A)$) when A, $\mathcal{C}$, Init and Safe satisfy the hypotheses of Corollary 3.5.

5.1 Karr's Affine Equalities Domain

Program analysis on the domain of affine equalities has been introduced in 1976 by Karr [17] who designed some algorithms computing for each program point some correct affine equalities between numerical variables. This abstract domain, here denoted by Aff, is relatively simple and widely used in numerical program analysis (see, *e.g.*, [21]). Müller-Olm and Seidl [26] put forward simpler and more efficient algorithms for Aff based on a different representation of affine sets and proved that Aff is fixpoint complete for unguarded nondeterministic affine programs, while for linearly guarded nondeterministic affine programs it is undecidable whether a given affine equality holds at a given program point or not.

Let us briefly recall the definition of the abstract domain $\textsc{Aff}_n$ for n program variables ranging in $\mathrm{Var}_n \triangleq \{x_1, ..., x_n\}$ and assuming rational values[1], that is, $\mathbb{V} = \mathbb{Q}$. The logical abstract invariants represented by $\textsc{Aff}_n$ are finite (possibly empty) conjunctions of affine equalities between variables, namely, $\bigwedge_{j=1}^{k}(\sum_{i=1}^{n} m_{i,j}x_i + b_j = 0)$, with $m_{i,j}, b_j \in \mathbb{Q}$. Any conjunction of affine equalities defines an affine subset of $\mathbb{Q}^n$, and each subset $X \in \wp(\mathbb{Q}^n)$ is approximated by the least (w.r.t. $\subseteq$) affine subset containing X, which is:

$$\mathsf{aff}(X) \triangleq \{\textstyle\sum_{j=0}^{m}\lambda_j\vec{v}_j \in \mathbb{Q}^n \mid m \in \mathbb{N},\ \lambda_j \in \mathbb{Q},\ \vec{v}_j \in X,\ \sum_{j=0}^{m}\lambda_j = 1\}.$$

This map $\mathsf{aff} : \wp(\mathbb{Q}^n) \to \wp(\mathbb{Q}^n)$ is an upper closure on $\langle\wp(\mathbb{Q}^n), \subseteq\rangle$ whose fixpoints are precisely the affine subsets of $\mathbb{Q}^n$ and therefore may be used to define the affine equalities domain $\textsc{Aff}_n \triangleq \langle\mathsf{aff}(\wp(\mathbb{Q}^n)), \subseteq\rangle$ independently of a specific representation for its elements. Karr [17] represents affine sets by kernels of affine transformations stored as matrix-vector pairs, while Müller-Olm and Seidl [26] employ an affine basis of indepedent vectors called generators. One can switch from one representation to the other by solving equations and using Gaussian elimination. Here, we do not need to choose a specific representation of affine sets so that the upper closure aff is meant to act as abstraction map $\alpha_{\textsc{Aff}} : \wp(\mathbb{Q}^n) \to \textsc{Aff}_n$ and correspondingly the concretization $\gamma_{\textsc{Aff}} : \textsc{Aff}_n \to \wp(\mathbb{Q}^n)$ is the identity. For the sake of clarity, in our examples we will use logical affine equalities for representing affine sets. $\textsc{Aff}_n$ is a complete lattice of finite height $n+1$, because if $a, a' \in \textsc{Aff}_n$ and $a \subsetneq a'$ then $\dim(a) < \dim(a')$, where $\dim(\varnothing) = -1$ and $\dim(\mathbb{Q}^n) = n$. The domain $\textsc{Aff}_n$ is not closed under arbitrary unions, *i.e.*, aff is not an additive uco, so that the lub of $\mathcal{X} \subseteq \textsc{Aff}_n$ is given by $\sqcup_{\textsc{Aff}_n}\mathcal{X} \triangleq \mathsf{aff}(\cup_{X\in\mathcal{X}}X)$. A matrix-based algorithm for computing a binary lub $a \sqcup_{\textsc{Aff}} a'$ of two affine sets represented by affine transformations is given by Karr [17, Section 5.2] (a simpler and more efficient version is in [21, Section 5.2.2]), while a binary lub can be easily computed for the generators-based representation in [26, Section 3].

By Corollary 3.4, the existence of abstract inductive invariants in Aff for a given class $\mathcal{C}$ of programs, Init of sets of initial states and Safe of sets of safety properties, is a decidable problem iff for all $\mathcal{P} \in \mathcal{C}$ with n variables and control nodes in Q, for all $\Sigma_0 \in \mathrm{Init}_{\mathcal{P}} \subseteq \wp(\mathbb{V}^n)^{|Q|}$ and for all $P \in \mathrm{Safe}_{\mathcal{P}} \subseteq \wp(\mathbb{V}^n)^{|Q|}$,

$$\dot{\gamma}_{\textsc{Const}}(\mathsf{lfp}(\lambda\vec{a} \in \textsc{Aff}_n^{|Q|}\,.\,\dot{\alpha}_{\textsc{Aff}}(\Sigma_0)\,\dot{\sqcup}_{\textsc{Aff}_n}\,\dot{\alpha}_{\textsc{Aff}}(\mathsf{post}_{\mathcal{P}}(\dot{\gamma}_{\textsc{Aff}}(\vec{a}))))) \,\dot{\subseteq}^{?}\, P \tag{5}$$

is decidable. Therefore, when $\Sigma_0, P \in \textsc{Aff}_n$ and since $\textsc{Aff}_n^{|Q|}$ has finite height $|Q|(n+1)$, a sufficient condition for the decidability of the problem (5) is that the bca $\dot{\alpha}_{\textsc{Aff}} \circ \mathsf{post}_{\mathcal{P}} \circ \dot{\gamma}_{\textsc{Aff}} : \textsc{Aff}_n^{|Q|} \to \textsc{Aff}_n^{|Q|}$ is computable, where for all $\vec{a} \in \textsc{Aff}_n^{|Q|}$:

[1] Values range in $\mathbb{Q}$ because the representation of affine subspaces and the transfer functions rely on algorithms working on fields rather than rings such as $\mathbb{Z}$.

$$\alpha_{\dot{\text{AFF}}}(\mathsf{post}_{\mathcal{P}}(\dot{\gamma}_{\text{AFF}}(\vec{a}))) = \langle \sqcup_{\text{AFF}_n} \{\alpha_{\text{AFF}}(t(\pi_q(\dot{\gamma}_{\text{AFF}}(\vec{a})))) \mid \exists q \in Q, \exists t \in \mathsf{T}_{\mathcal{P}}.\, q \xrightarrow{t} q'\}\rangle_{q' \in Q} \quad (6)$$

Because in (6) we have a finite lub, it is enough that for all the transfer functions $t \in \mathsf{T}_{\mathcal{P}}$ of $\mathcal{P}$, the bca $\alpha_{\text{AFF}} \circ t \circ \gamma_{\text{AFF}} : \text{AFF}_n \to \text{AFF}_n$ is algorithmically computable.

We consider single affine assignments $t_a^j \triangleq x_j := \sum_{i=1}^n m_i x_i + b$ and linear Boolean guards $t_b \triangleq \sum_{i=1}^n m_i x_i + b \bowtie 0$, where $m_i, b \in \mathbb{Q}$ and $\bowtie \in \{=, \neq, <, \leq, >, \geq\}$, whose corresponding transfer functions are as follows: for all $Y \in \wp(\mathbb{Q}^n)$,

$$t_a^j(Y) \triangleq \{\vec{v}[\vec{v}_j/v'] \in \mathbb{Q}^n \mid \vec{v} \in Y, v' = \textstyle\sum_{i=1}^n m_i \vec{v}_i + b\}, \quad t_b(Y) \triangleq \{\vec{v} \in Y \mid \textstyle\sum_{i=1}^n m_i \vec{v}_i + b \bowtie 0\}.$$

These transfer functions can be extended to include parallel affine assignments $\vec{x} := M\vec{x} + \vec{b}$, where $M \in \mathbb{Q}^{n \times n}$ is a $n \times n$ matrix and $\vec{b} \in \mathbb{Q}^n$, which performs n parallel single affine assignments, and conjunctive (disjunctive) linear Boolean guards $M\vec{x} + \vec{b} \bowtie 0$, which holds iff, for all (there exists) $j \in [1, n]$, $\sum_{i=1}^n M_{ji} x_i + b_j \bowtie 0$ holds.

Karr gave already in [17, Section 4.2] an algorithm for computing the bca of an affine assignment t_a^j for affine sets represented by kernels of affine transformations. Müller-Olm and Seidl [26] put forward a more efficient algorithm for their representation based on generators. It is also worth remarking that Müller-Olm and Seidl [26, Lemma 2] observe that the bca of t_a^j turns out to be pointwise complete, namely $\mathsf{aff} \circ t_a^j \circ \mathsf{aff} = \mathsf{aff} \circ t_a^j$ holds. Hence, in turn, computability of parallel affine assignments $\vec{x} := M\vec{x} + \vec{b}$ easily follows. [26, Section 4] also shows that the bca of a nondeterministic assignment $t_{a?}^j \triangleq x_j :=?$ is computable, where the corresponding transfer function is defined by: $t_{x_j:=?}(Y) \triangleq \{\vec{v}[\vec{v}_j/v'] \in \mathbb{Q}^n \mid \vec{v} \in Y,\, v' \in \mathbb{Q}\}$. In fact, one can observe [26, Lemma 4] that $\mathsf{aff}(t_{x_j:=?}(\mathsf{aff}(Y))) = \mathsf{aff}(t_{x_j:=0}(\mathsf{aff}(Y))) \sqcup_{\text{AFF}} \mathsf{aff}(t_{x_j:=1}(\mathsf{aff}(Y)))$, so that computing the bca $\mathsf{aff}(t_{x_j:=?}(\mathsf{aff}(Y)))$ is reduced to the lub of the bca's of the transfer functions of the affine assignments $x_j := 0$ and $x_j := 1$.

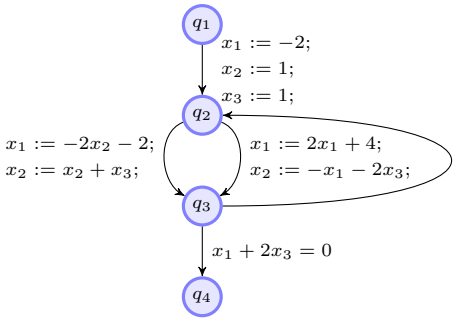
As observed by Karr [17, Section 4.1] (see also [21, Section 5.2.3] for a modern approach), bca's of affine equalities Boolean guards of the shape $t_{b=} \triangleq \sum_{i=1}^n m_i x_i + b = 0$ are algorithmically computable through the glb of AFF_n, *i.e.*, for all $a \in \text{AFF}_n$, $\alpha_{\text{AFF}}(t_{b=}(\gamma_{\text{AFF}}(a))) = a \sqcap_{\text{AFF}_n} a_{b=}$, where $a_{b=} \in \text{AFF}_n$ denotes the affine set representing the affine equality $\sum_{i=1}^n m_i x_i + b = 0$. For affine inequalities Boolean guards $t_{b\neq} \triangleq \sum_{i=1}^n m_i x_i + b \neq 0$, Karr [17, Section 4.1] defines the following abstract function: $t_{b\neq}^\sharp(a) \triangleq \textbf{if } a \subseteq a_{b=} \textbf{ then } \bot_{\text{AFF}_n} \textbf{ else } a$, and states that "we must be content with a on the "otherwise" case...a general study of how best to handle decision nodes which are not of the simple form $t_{b=}$ is *in preparation*", but this document never appeared. Nevertheless, we notice that the above definition of $t_{b\neq}^\sharp$ actually is the bca $\alpha_{\text{AFF}} \circ t_{b\neq} \circ \gamma_{\text{AFF}} = \lambda a.\, \mathsf{aff}(a \cap \neg a_{b=})$. In fact, let us observe the following fact $(*)$: if $a, a' \in \text{AFF}_n$ then $a' \subsetneq a \Rightarrow \mathsf{aff}(a \cap \neg a') = a$. In fact, we have that $\dim(a') < \dim(a)$ and, in turn, $\dim(\mathsf{aff}(a \cap \neg a')) = \dim(\mathsf{aff}(a \smallsetminus a')) = \dim(a)$ hold, therefore entailing that $\mathsf{aff}(a \cap \neg a') = a$. Thus: (a) if $a_{b=} \subsetneq a$ then, by $(*)$, $\mathsf{aff}(a \cap \neg a_{b=}) = a$; (b) if $a_{b=}$ and a are incomparable then $a \cap a_{b=} \subsetneq a$, so that, by $(*)$, $\mathsf{aff}(a \cap \neg a_{b=}) = \mathsf{aff}(a \cap \neg(a \cap a_{b=})) = a$.

Summing up, as a consequence of Corollary 3.4, the above analysis of bca's in the abstract domain Aff gives us the following result for the class $\mathcal{C}_{\text{AFF}}$ of nondeterministic programs with (possibly parallel) affine assignments, (possibly parallel) nondeterministic assignments and (conjunctive or disjunctive) affine equalities/inequalities guards.

▶ **Theorem 5.1** (Decidability and Synthesis of Inductive Invariants in Aff)**.** *The Monniaux problem* (4) *on* Aff *for programs in* $\mathcal{C}_{\text{AFF}}$*, affine sets of initial states and affine sets of state properties is decidable. Moreover, the algorithm* AInv *of Corollary 3.5 instantiated to* $\mathsf{post}_{\mathcal{P}}$ *for* $\mathcal{P} \in \mathcal{C}_{\text{AFF}}$*, synthesizes the least inductive invariant of* $\mathcal{P}$ *in* Aff*, when this exists.*

To the best of our knowledge, the literature provides no algorithm for computing the bca of further linear Boolean guards $\sum_{i=1}^{n} m_i x_i + b \bowtie 0$, with $\bowtie \in \{<, \leq, \}$. We conjecture that at least some of these bca's are algorithmically computable.

▶ **Example 5.2.** Consider the following nondeterministic program $\mathcal{R}$ with $\Sigma_0 = \{q_1\} \times \mathbb{Q}^3 \in \textsc{Aff}^{|Q|}$ and the property $P^\sharp = \langle\langle q_1, \top\rangle, \langle q_2, \top\rangle, \langle q_3, \top\rangle, \langle q_4, x_1 + x_2 + 1 = 0\rangle\rangle \in \textsc{Aff}^{|Q|}$.



The algorithm AInv of Corollary 3.5 yields the following sequence of $I^j \in \textsc{Aff}^{|Q|}$:

$$
\begin{aligned}
I^0 &= \dot{\alpha}_{\textsc{Aff}}(\Sigma_0) = \langle\langle q_1, \top\rangle, \langle q_2, \bot\rangle, \langle q_3, \bot\rangle, \langle q_4, \bot\rangle\rangle \\
I^1 &= \langle\langle q_1, \top\rangle, \langle q_2, x_1 + 2 = 0 \wedge x_2 - 1 = 0 \wedge x_3 - 1 = 0\rangle, \langle q_3, \bot\rangle, \langle q_4, \bot\rangle\rangle \\
I^2 &= \langle\langle q_1, \top\rangle, \langle q_2, x_1 + 2 = 0 \wedge x_2 - 1 = 0 \wedge x_3 - 1 = 0\rangle, \\
&\qquad \langle q_3, x_1 + 2x_2 = 0 \wedge x_3 = 1\rangle, \langle q_4, \bot\rangle\rangle \\
I^3 &= \langle\langle q_1, \top\rangle, \langle q_2, x_1 + 2x_2 = 0 \wedge x_3 = 1\rangle, \langle q_3, x_1 + 2x_2 = 0 \wedge x_3 = 1\rangle, \\
&\qquad \langle q_4, x_1 + x_2 + 1 = 0 \wedge x_3 = 1, \bot\rangle\rangle = \dot{\alpha}_{\textsc{Aff}}(\mathsf{post}_{\mathcal{R}}(\dot{\gamma}_{\textsc{Aff}}(I^3))) \dot{\leq}_{\textsc{Aff}} P^\sharp
\end{aligned}
$$

The output I^3 is the analysis of $\mathcal{R}$ with the bca's of its transfer functions in Aff, *i.e.*, it is the least inductive invariant in Aff which allows us to prove that $P^\sharp$ holds. ⌟

5.1.1 Relationship with Müller-Olm and Seidl [26]

Müller-Olm and Seidl [26] implicitly show that the transfer functions of affine assignments t_a and of nondeterministic assignments $t_{x_j:=?}$ are pointwise complete. In fact, [26, Lemma 2] shows that for all $X \in \wp(\mathbb{Q}^n)$, $t_a(\mathsf{aff}(X)) = \mathsf{aff}(t_a(X))$, from which we easily obtain: $\mathsf{aff}(t_a(X)) = \mathsf{aff}(t_a(\mathsf{aff}(X)))$ and

$$
\begin{aligned}
&\mathsf{aff}(t_{x_j:=?}(X)) = \mathsf{aff}(\cup_{z\in\mathbb{Q}} t_{x_j:=z}(X)) = \mathsf{aff}(\cup_{z\in\mathbb{Q}}\, \mathsf{aff}(t_{x_j:=z}(X))) = \\
&\qquad = \mathsf{aff}(\cup_{z\in\mathbb{Q}}\, \mathsf{aff}(t_{x_j:=z}(\mathsf{aff}(X))) = \mathsf{aff}(\cup_{z\in\mathbb{Q}} t_{x_j:=z}(\mathsf{aff}(X)) = \mathsf{aff}(t_{x_j:=?}(\mathsf{aff}(X)))
\end{aligned}
$$

Thus, since pointwise completeness entails fixpoint completeness (cf. Section 4.1), for all unguarded programs $\mathcal{P}$ with affine and nondeterministic assignments, $\dot{\alpha}_{\textsc{Aff}}(\mathsf{lfp}(\lambda\vec{X}.\Sigma_0 \cup \mathsf{post}_{\mathcal{P}}(\vec{X}))) = \mathsf{lfp}(\lambda\vec{a}.\dot{\alpha}_{\textsc{Aff}}(\Sigma_0) \dot{\sqcup}_{\textsc{Aff}}\ \dot{\alpha}_{\textsc{Aff}}(\mathsf{post}_{\mathcal{P}}(\dot{\gamma}_{\textsc{Aff}}(\vec{a}))))$ holds, which is the reason why "Karr's algorithm is precise for affine programs, *i.e.*, computes not just some but all valid affine relations" [26, Section 1]. However, fixpoint completeness is lost as soon as affine equality or inequality guards are included, although these programs still belong to $\mathcal{C}_{\textsc{Aff}}$, because these guards are not pointwise complete: for example, we have that $\mathsf{aff}(t_{x_1=0}(\{(1,0),(-1,0)\})) = \mathsf{aff}(\varnothing) = \varnothing$, whereas $\mathsf{aff}(t_{x_1=0}(\mathsf{aff}(\{(1,0),(-1,0)\}))) = \mathsf{aff}(t_{x_1=0}(x_2 = 0)) = \mathsf{aff}(\{(0,0)\}) = \{(0,0)\}$. Müller-Olm and Seidl [26, Section 7] also prove that as soon as affine equality guards are added to nondeterministic affine programs it becomes undecidable whether a

given affine equality holds in some program point or not. It is therefore worth remarking that this undecidability does not prevent the decidability result of Theorem 5.1 on the existence (and synthesis) of *inductive* affine equalities proving a given affine equality, since these two decision problems are different. In fact, Müller-Olm and Seidl [26, Section 7] prove that $\forall f \in \mathcal{C}_{\text{AFF}}.\forall a \in \text{AFF}\,.\,\alpha_{\text{AFF}}(\mathsf{lfp}(f)) \leq^{?}_{\text{AFF}} a$ is an undecidable problem, while Theorem 5.1 shows that $\forall f \in \mathcal{C}_{\text{AFF}}.\forall a \in \text{AFF}\,.\,\mathsf{lfp}(\alpha_{\text{AFF}} f \gamma_{\text{AFF}}) \leq^{?}_{\text{AFF}} a$ is decidable, and, by Theorem 4.1 (b), these are equivalent problems iff (where "if" is obvious) fixpoint completeness $\forall f \in \mathcal{C}_{\text{AFF}}.\alpha_{\text{AFF}}(\mathsf{lfp}(f)) = \mathsf{lfp}(\alpha_{\text{AFF}} f \gamma_{\text{AFF}})$ holds.

▶ **Example 5.3.** Consider the following deterministic program $\mathcal{P} \in \mathcal{C}_{\text{AFF}}$:

$$\mathcal{P} \;\equiv\; x_1 := 0;\; x_2 := 3;\; \textbf{while}\ (x_1 \neq 3)\ \textbf{do}\ \{x_1 := x_1 + 2;\; x_2 := x_2 - 2\}$$

where q_i and q_o denote, resp., its entry and exit program points and $\Sigma_0 = \{q_i\} \times \mathbb{Q}^2$ is the set of initial states. Then, we have that the affine abstraction of the reachable states at the exit point q_0 is

$$\begin{aligned}\alpha_{\text{AFF}}(\pi_{q_o}(\mathsf{lfp}(\lambda\vec{X}.\Sigma_0 \cup \mathsf{post}_{\mathcal{P}}(\vec{X})))) &= \alpha_{\text{AFF}}(\{(0+2n, 3-2n) \mid n \in \mathbb{N}\} \cap \langle x_1 = 3\rangle)\\ &= \alpha_{\text{AFF}}(\varnothing) = \bot_{\text{AFF}}\end{aligned}$$

while the algorithm AInv of Corollary 3.5 using the best correct approximations of the transfer functions of $b^{=} \equiv (x_1 = 3)$ and $b^{\neq} \equiv (x_1 \neq 3)$ gives:

$$\begin{aligned}\pi_{q_o}(\mathsf{lfp}(\lambda\vec{a} \in \text{AFF}^{|Q|}.\, \alpha_{\dot{\text{AFF}}}(\Sigma_0)\,\dot{\sqcup}_{\text{AFF}}\, \alpha_{\dot{\text{AFF}}}(\mathsf{post}_{\mathcal{P}}(\gamma_{\dot{\text{AFF}}}(\vec{a})))))) &= \langle x_1 + x_2 = 3\rangle \sqcap_{\text{AFF}} \langle x_1 = 3\rangle\\ &= \langle x_1 = 3 \wedge x_2 = 0\rangle \not\leq_{\text{AFF}} \bot_{\text{AFF}}\end{aligned}$$

The least inductive invariant $\langle x_1 = 3 \wedge x_2 = 0\rangle$ in AFF does not entail $\bot_{\text{AFF}}$, namely, it cannot prove that q_o is unreachable, therefore showing that the two aforementioned decision problems are not equivalent for programs ranging in $\mathcal{C}_{\text{AFF}}$. ⌟

6 Co-Inductive Synthesis of Abstract Inductive Invariants

In this section we design a synthesis algorithm which, by generalizing an algorithm by Padon et al. [28], outputs the *greatest* abstract inductive invariant ranging in some abstract domain, when this exists. This algorithm is obtained by dualizing the procedure AInv in Corollary 3.5 to a co-inductive greatest fixpoint computation and requires that the abstract domain is equipped with a suitable well-quasiorder relation. Let us recall that a quasiordered set $D_{\leq}$ is a well-quasiordered set (wqoset), and $\leq$ is called well-quasiorder (wqo), when for every countably infinite sequence of elements $\{x_i\}_{i\in\mathbb{N}}$ in D there exist $i, j \in \mathbb{N}$ such that $i < j$ and $x_i \leq x_j$. Equivalently, D is a wqoset iff D is DCC (also called well-founded) and D has no infinite antichain (*i.e.*, a subset whose distinct elements are pairwise incomparable).

In the following, we will leverage on the closure operator approach for defining abstract domains, which, as recalled in Section 2.2, is completely equivalent to Galois connections and particularly suitable for reasoning on abstract domains independently from their representation. Let $\mathcal{T} = \langle \Sigma, \tau\rangle$ be a transition system whose successor transformer is denoted by post. Padon et al. [28] consider abstract invariants ranging in a set (of semantics of logical formulae) $L \subseteq \wp(\Sigma)$ and assume (in [28, Theorem 4.2]) that $\langle L, \subseteq\rangle$ is closed under finite intersections (*i.e.*, logical conjunctions). Accordingly to Assumption 3.1, we ask that $\langle L, \subseteq\rangle$ satisfies the requirement of being an abstract domain of $\langle \wp(\Sigma), \subseteq\rangle$, which corresponds to ask that $\langle L, \subseteq\rangle$ is closed under arbitrary, rather than finite, intersections. Thus, L is the image of

an upper closure $\check{\mu}_L \in \mathrm{uco}(\wp(\Sigma)_{\subseteq})$ defined by: $\check{\mu}_L(X) \triangleq \cap\{\phi \in L \mid X \subseteq \phi\}$. The three key definitions and related assumptions of the synthesis algorithm defined in [28, Theorem 4.2] concern a quasiorder $\sqsubseteq_L \subseteq \Sigma \times \Sigma$ between states, a function $Av_L : \Sigma \to \wp(\Sigma)$ called Avoid, and an abstract transition relation $\tau^L \subseteq \Sigma \times \Sigma$, and are as follows:

(1) $s \sqsubseteq_L s' \triangleq \forall \phi \in L.\, s' \in \phi \Rightarrow s \in \phi$	Assumption (A_1): $\langle \Sigma, \sqsubseteq_L \rangle$ is a wqoset
(2) $Av_L(s) \triangleq \cup\{\phi \in L \mid \phi \subseteq \neg\{s\}\}$	Assumption (A_2): $\forall s \in \Sigma.\ Av_L(s) \in L$
(3) $(s, s') \in \tau^L \triangleq (s, s') \in \tau \vee s' \sqsubseteq_L s$	Abstract Transition System $\mathcal{T}^L \triangleq \langle \Sigma, \tau^L \rangle$

Correspondingly, we define the down-closure $\delta_L : \wp(\Sigma) \to \wp(\Sigma)$ of the quasiorder $\sqsubseteq_L$, we lift $Av_L : \wp(\Sigma) \to \wp(\Sigma)$ to sets of states and we define the successor transformer $\mathsf{post}^L : \wp(\Sigma) \to \wp(\Sigma)$ of $\mathcal{T}^L$ as follows:

(1) $\delta_L(X) \triangleq \{s \in \Sigma \mid \exists s' \in X.\, s \sqsubseteq_L s'\}$	Down-closure of $\sqsubseteq_L$
(2) $Av_L(X) \triangleq \cup\{\phi \in L \mid \phi \subseteq \neg X\}$	Av_L acts on any set X of states
(3) $\mathsf{post}^L(X) \triangleq \mathsf{post}(X) \cup \delta_L(X)$	Successor transformer of $\mathcal{T}^L$

▶ **Lemma 6.1.** *The following properties hold:*
(a) $s \sqsubseteq_L s'$ *iff* $\check{\mu}_L(\{s\}) \subseteq \check{\mu}_L(\{s'\})$.
(b) (A_1) *holds iff* $\langle L, \subseteq \rangle$ *is a well-quasi order.*
(c) (A_2) *holds iff* L *is closed under arbitrary unions.*
(d) *If* (A_2) *holds then* $\delta_L = \check{\mu}_L$ *and both are additive ucos (and* $\delta_L(\phi) = \phi \Leftrightarrow \phi \in L$*).*
(e) *For all* $\Sigma_0 \in \wp(\Sigma)$, $\mathsf{lfp}(\lambda X.\Sigma_0 \cup \mathsf{post}^L(X)) = \mathsf{lfp}(\lambda X.\check{\mu}_L(\Sigma_0 \cup \mathsf{post}(X)))$.

In particular, Lemma 6.1 (e) states that the reachable states in $\mathcal{T}^L$ coincide with the reachable states of the abstract transition system $\mathcal{T}^{\check{\mu}_L} \triangleq \langle \Sigma, \check{\mu}_L \circ \mathsf{post} \rangle$ obtained by considering the best correct approximation of post in the abstract domain $\check{\mu}_L$. As a direct consequence of the abstract inductive invariant principle Lemma 3.2 (a), we obtain the following characterization of the abstract inductive invariants ranging in L of the transition system $\mathcal{T}$ which relies on the reachable states of its best abstraction $\mathcal{T}^{\check{\mu}_L}$ in the domain $\check{\mu}_L$.

▶ **Corollary 6.2.** *Let* $\Sigma_0, P \in \wp(\Sigma)$. *Then,* $\exists \phi \in L.\, \Sigma_0 \subseteq \phi \wedge \mathsf{post}(\phi) \subseteq \phi \wedge \phi \subseteq P$ *iff* $\mathrm{Reach}[\mathcal{T}^{\check{\mu}_L}, \Sigma_0] \subseteq P$.

6.1 Co-Inductive Invariants

Following [28], in the following we make assumption (A_2), that is, by Lemma 6.1 (c), we assume that $L \subseteq \wp(\Sigma)$ is closed under arbitrary unions. This means that $\check{\mu}_L$ is an additive uco on $\wp(\Sigma)_{\subseteq}$, *i.e.*, in abstract interpretation terminology, $\check{\mu}_L$ is a *disjunctive* abstract domain whose abstract lub does not lose precision (see, *e.g.*, [21, Section 6.3]). Furthermore, we also have that L is the image of a co-additive (*i.e.*, preserving arbitrary intersections) lower closure $\hat{\mu}_L : \wp(\Sigma) \to \wp(\Sigma)$ defined by $\hat{\mu}_L(X) \triangleq \cup\{\phi \in L \mid \phi \subseteq X\}$. It turns out that the uco $\check{\mu}_L$ is adjoint to the lco $\hat{\mu}_L$ namely, $\check{\mu}_L(X) \subseteq Y \Leftrightarrow X \subseteq \hat{\mu}_L(Y)$ holds. In fact, if $\check{\mu}_L(X) \subseteq Y$ then, by applying $\hat{\mu}_L$, $X \subseteq \check{\mu}_L(X) = \hat{\mu}_L(\check{\mu}_L(X)) \subseteq \hat{\mu}_L(Y)$; the converse is dual.

As observed by Cousot [1, Theorem 4], the inductive invariant principle (2) can be dualized when f admits right-adjoint $\widetilde{f} : C \to C$ (this happens iff f is additive): in this case,

$$\mathsf{lfp}(\lambda x.c \vee f(x)) \leq c' \Leftrightarrow c \leq \mathsf{gfp}(\lambda x.c' \wedge \widetilde{f}(x)) \tag{7}$$

holds and one obtains a *co-inductive* invariant principle:

$$c \leq \mathsf{gfp}(\lambda x.\widetilde{f}(x) \wedge c') \Leftrightarrow \exists j \in C.\, c \leq j \wedge j \leq \widetilde{f}(j) \wedge j \leq c' \tag{8}$$

One such $j \in C$ is therefore called a *co-inductive invariant* of f for $\langle c, c' \rangle$. The co-inductive invariant proof method (8) can be applied to safety verification of any transition system $\mathcal{T}$ because post is additive and therefore it always admits right adjoint $\widetilde{\mathsf{pre}}$ (cf. Section 2.3). Hence, we obtain that $\mathsf{lfp}(\lambda X.\Sigma_0 \cup \mathsf{post}(X)) \subseteq P$ iff $\Sigma_0 \subseteq \mathsf{gfp}(\lambda X.\, \widetilde{\mathsf{pre}}(X) \cap P)$ iff there exists a co-inductive invariant for $\widetilde{\mathsf{pre}}$ for $\langle \Sigma_0, P \rangle$. By (2) and (8), it turns out that I is an inductive invariant of post for $\langle \Sigma_0, P \rangle$ iff I is a co-inductive invariant of $\widetilde{\mathsf{pre}}$ for $\langle \Sigma_0, P \rangle$. Also, while $\mathsf{lfp}(\lambda X.\Sigma_0 \cup \mathsf{post}(X))$ is the least, *i.e.* logically strongest, inductive invariant, we have that $\mathsf{gfp}(\lambda X.\, \widetilde{\mathsf{pre}}(X) \cap P)$ is the greatest, *i.e.* logically weakest, inductive invariant [1, Theorem 6].

We show how the co-inductive invariant principle (8) applied to the best abstract transition system $\mathcal{T}^{\check{\mu}_L} = (\Sigma, \check{\mu}_L \circ \mathsf{post}_{\mathcal{T}})$ provides exactly the synthesis algorithm by Padon et al. [28, Algorithm 1]. In order to do this, we first give the following alternative characterization of the reachable states of $\mathcal{T}^{\check{\mu}_L}$.

▶ **Lemma 6.3.** $\mathsf{lfp}(\lambda X.\Sigma_0 \cup \check{\mu}_L(\mathsf{post}(X))) = \mathsf{lfp}(\lambda X.\Sigma_0 \cup \mathsf{post}(\check{\mu}_L(X)) \cup \check{\mu}_L(X))$.

Consequently, $\mathsf{lfp}(\lambda X.\Sigma_0 \cup \check{\mu}_L(\mathsf{post}(X))) \subseteq P \Leftrightarrow \mathsf{lfp}(\lambda X.\Sigma_0 \cup \mathsf{post}(\check{\mu}_L(X)) \cup \check{\mu}_L(X)) \subseteq P$ holds. Since $\lambda X.\, \mathsf{post}(\check{\mu}_L(X)) \cup \check{\mu}_L(X)$ is additive, we can apply the co-inductive invariant principle (8) by considering its adjoint function, which is as follows:

$$\mathsf{post}(\check{\mu}_L(X)) \cup \check{\mu}_L(X) \subseteq Y \Leftrightarrow \mathsf{post}(\check{\mu}_L(X)) \subseteq Y \wedge \check{\mu}_L(X) \subseteq Y \Leftrightarrow$$
$$X \subseteq \hat{\mu}_L(\widetilde{\mathsf{pre}}(Y)) \wedge X \subseteq \hat{\mu}_L(Y) \Leftrightarrow X \subseteq \hat{\mu}_L(\widetilde{\mathsf{pre}}(Y)) \cap \hat{\mu}_L(Y).$$

Thus, by Lemma 6.3 and (7), we obtain: $\mathsf{lfp}(\lambda X.\check{\mu}_L(\Sigma_0) \cup \check{\mu}_L(\mathsf{post}(X))) \subseteq P$ iff $\check{\mu}_L(\Sigma_0) \subseteq \mathsf{gfp}(\lambda X.\hat{\mu}_L(\widetilde{\mathsf{pre}}(X) \cap X \cap P))$ iff $\Sigma_0 \subseteq \mathsf{gfp}(\lambda X.\hat{\mu}_L(\widetilde{\mathsf{pre}}(X) \cap X \cap P))$, and, in turn, by the abstract inductive invariant principle (Lemma 3.2 (a) for ucos) applied to $\check{\mu}_L \circ (\lambda X.\Sigma_0 \cup \mathsf{post}(X)) = \lambda X.\check{\mu}_L(\Sigma_0) \cup \check{\mu}_L(\mathsf{post}(X))$ we get:

$$\exists \phi \in L.\Sigma_0 \subseteq \phi \wedge \mathsf{post}(\phi) \subseteq \phi \wedge \phi \subseteq P \Leftrightarrow \Sigma_0 \subseteq \mathsf{gfp}(\lambda X.\hat{\mu}_L(\widetilde{\mathsf{pre}}(X) \cap X \cap P)).$$

This leads us to use the algorithm introduced by Cousot [1, Algorithm 2] which synthesizes an inductive invariant by applying Knaster-Tarski theorem to compute the iterates of the greatest fixpoint of $\lambda X.\hat{\mu}_L(\widetilde{\mathsf{pre}}(X) \cap X \cap P)$ as long as the current iterate I_1 contains Σ_0:

Algorithm 1 Co-inductive backward synthesis of abstract inductive invariants.

$I_1 := \Sigma$;
while $\Sigma_0 \subseteq I_1$ **do** // Loop invariant: $I_1 \in L$
  if $(I_1 = \hat{\mu}_L(\widetilde{\mathsf{pre}}(I_1) \cap I_1 \cap P))$ **then return** I_1 *is an inductive invariant in* L;
  $I_1 := I_1 \cap \hat{\mu}_L(\widetilde{\mathsf{pre}}(I_1) \cap I_1 \cap P)$;
return *no inductive invariant in* L;

Since, $\widetilde{\mathsf{pre}}$ is computable and, by Lemma 6.1 (b), $\langle \hat{\mu}_L, \subseteq \rangle = \langle L, \subseteq \rangle$ is a wqo, we immediately obtain that Algorithm 1 is correct and terminating. Furthermore, if Algorithm 1 outputs an inductive invariant I_1 proving the property P then I_1 is the *greatest* inductive invariant in L proving P. It turns out that Algorithm 1 exactly coincides with the synthesis algorithm by Padon et al. [28], which is replicated here as Algorithm 2.

▶ **Theorem 6.4.** *Algorithm* 1 = *Algorithm* 2.

This shows that Algorithm 2 in [28] for a disjunctive GC-based abstract domain A amounts to a backward (*i.e.*, propagating $\widetilde{\mathsf{pre}}$) static analysis using the best correct approximations in A of the transfer functions, as long as the ordering of A guarantees its termination, *e.g.*, because A is well-founded.

Algorithm 2 Inductive invariant algorithm by [28].

$I_2 := \Sigma$;
while I_2 *is not an inductive invariant* **do** // Loop invariant: $I_2 \in L$
  if $\Sigma_0 \not\subseteq I_2$ **then return** *no inductive invariant in* L;
  choose $s \in \Sigma$ *as a counterexample to inductiveness of* I_2;
  $I_2 := I_2 \cap Av_L(s)$;
return I_2 *is an inductive invariant in* L;

6.2 Backward and Forward Algorithms

Algorithm 1 is backward because it applies $\widetilde{\mathsf{pre}}$, for termination it requires that the abstract domain $\langle \check{\mu}_L, \subseteq \rangle$ is DCC and it turns out to be the dual of the forward algorithm AInv provided by Corollary 3.5 for post and requiring that $\langle \check{\mu}_L, \subseteq \rangle$ is ACC. A different gfp-based *forward* algorithm can be designed by observing (as in [6, Section 3]) that $\mathsf{lfp}(\lambda X.\Sigma_0 \cup \mathsf{post}(X)) \subseteq P$ iff $\mathsf{lfp}(\lambda X.\neg P \cup \mathsf{pre}(X)) \subseteq \neg\Sigma_0$. Here, the dualization provided by the equivalence (7) yields:

$$\mathsf{lfp}(\lambda X.\neg P \cup \check{\mu}_L(\mathsf{pre}(X))) \subseteq \neg\Sigma_0 \Leftrightarrow \neg P \subseteq \mathsf{gfp}(\lambda X.\hat{\mu}_L(\widetilde{\mathsf{post}}(X) \cap X \cap \neg\Sigma_0))$$

and induces the following *co-inductive forward algorithm* which relies on the state transformer $\widetilde{\mathsf{post}}$ and is terminating when $\langle \check{\mu}_L, \subseteq \rangle$ is assumed to be DCC:

Algorithm 3 Co-inductive forward synthesis of abstract inductive invariants.

$I := \Sigma$;
while $\neg P \subseteq I$ **do** // Loop invariant: $I \in L$
  if $(I = \hat{\mu}_L(\widetilde{\mathsf{post}}(I) \cap I \cap \neg\Sigma_0))$ **then return** I *is an inductive invariant in* L;
  $I := I \cap \hat{\mu}_L(\widetilde{\mathsf{post}}(I) \cap I \cap \neg\Sigma_0)$;
return *no inductive invariant in* L;

Furthermore, by dualizing the technique described by Cousot and Cousot [6, Section 4.3] for post and pre, one could also design a more efficient combined forward/backward synthesis algorithm which simultaneously make backward, by $\widetilde{\mathsf{pre}}$, and forward, by $\widetilde{\mathsf{post}}$, steps.

7 Future Work

As hinted by Monniaux [24], results of undecidability to the question (3) for some abstract domain A display a foundational trait since they "vindicate" (often years of intense) research on precise and efficient algorithms for *approximate* static program analysis on A. To the best of our knowledge, few undecidability results are available: an undecidability result by Monniaux [24, Theorem 1] for convex polyhedra [7] and by Fijalkow et al. [12, Theorem 1] for semilinear sets, *i.e.* finite unions of convex polyhedra. However, convex polyhedra and semilinear sets cannot be defined by a Galois connection and therefore do not satisfy our Assumption 3.1. As future work we plan to investigate whether the abstract inductive invariant principle could be exploited to provide a reduction of the undecidability of the question (3) for abstract domains which satisfy Assumption 3.1 and, in view of the characterization of fixpoint completeness in Section 4.2, for transfer functions which are not fixpoint complete.

We also plan to study whether complete abstractions can play a role in the decidability result by Hrushovski et al. [16] on the computation of the strongest polynomial invariant of an affine program. This hard result relies on the Zariski closure, which is continuous for affine functions and is pointwise complete for the transfer functions of affine programs.

Thus, fixpoint completeness for affine programs holds, and one could investigate whether the algorithm in [16] may be viewed as a least fixpoint computation of a best correct approximation on the Zariski abstraction.

References

1 Patrick Cousot. Partial completeness of abstract fixpoint checking. In *Proceedings of the 4th International Symposium on Abstraction, Reformulation, and Approximation (SARA'02)*, volume 1864 of *Lecture Notes in Computer Science*, pages 1–25. Springer-Verlag, 2000. doi:10.1007/3-540-44914-0_1.

2 Patrick Cousot. On fixpoint/iteration/variant induction principles for proving total correctness of programs with denotational semantics. In *Proc. 29th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2019)*, volume 12042 of *LNCS*, pages 3–18. Springer, 2019. doi:10.1007/978-3-030-45260-5_1.

3 Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'77)*, pages 238–252. ACM Press, 1977. doi:10.1145/512950.512973.

4 Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Proceedings of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'79)*, pages 269–282, New York, NY, USA, 1979. ACM. doi:10.1145/567752.567778.

5 Patrick Cousot and Radhia Cousot. Induction principles for proving invariance properties of programs. In D. Néel, editor, *Tools & Notions for Program Construction: an Advanced Course*, pages 75–119. Cambridge University Press, Cambridge, UK, August 1982.

6 Patrick Cousot and Radhia Cousot. Refining model checking by abstract interpretation. *Automated Software Engineering*, 6(1):69–95, 1999. doi:10.1023/A:1008649901864.

7 Patrick Cousot and Nicolas Halbwachs. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'78)*, pages 84–96. ACM, 1978. doi:10.1145/512760.512770.

8 Isil Dillig, Thomas Dillig, Boyang Li, and Kenneth L. McMillan. Inductive invariant generation via abductive inference. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, (OOPSLA'13)*, pages 443–456. ACM, 2013. doi:10.1145/2509136.2509511.

9 Yotam M. Y. Feldman, Neil Immerman, Mooly Sagiv, and Sharon Shoham. Complexity and information in invariant inference. *Proc. ACM Program. Lang.*, 4(POPL), 2019. doi:10.1145/3371073.

10 Yotam M. Y. Feldman, Oded Padon, Neil Immerman, Mooly Sagiv, and Sharon Shoham. Bounded Quantifier Instantiation for Checking Inductive Invariants. *Logical Methods in Computer Science*, Volume 15, Issue 3, 2019. doi:10.23638/LMCS-15(3:18)2019.

11 Yotam M. Y. Feldman, James R. Wilcox, Sharon Shoham, and Mooly Sagiv. Inferring inductive invariants from phase structures. In *Proc. 31st International Conference on Computer Aided Verification (CAV'19)*, volume 11562 of *Lecture Notes in Computer Science*, pages 405–425. Springer, 2019. doi:10.1007/978-3-030-25543-5_23.

12 Nathanaël Fijalkow, Engel Lefaucheux, Pierre Ohlmann, Joël Ouaknine, Amaury Pouly, and James Worrell. On the Monniaux problem in abstract interpretation. In *Proc. 26th International Static Analysis Symposium (SAS'19)*, volume 11822 of *Lecture Notes in Computer Science*, pages 162–180. Springer, 2019. doi:10.1007/978-3-030-32304-2_9.

13 Robert W. Floyd. Assigning meanings to programs. *Proceedings of Symposium on Applied Mathematics*, 19:19–32, 1967.

14 Roberto Giacobazzi, Francesco Ranzato, and Francesca Scozzari. Complete abstract interpretations made constructive. In *Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS'98)*, volume 1450 of *Lecture Notes in Computer Science*, pages 366–377. Springer, 1998. doi:10.1007/BFb0055786.

15 Roberto Giacobazzi, Francesco Ranzato, and Francesca Scozzari. Making abstract interpretations complete. *J. ACM*, 47(2):361–416, 2000. doi:10.1145/333979.333989.

16 Ehud Hrushovski, Joël Ouaknine, Amaury Pouly, and James Worrell. Polynomial invariants for affine programs. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, (LICS'18)*, pages 530–539. ACM, 2018. doi:10.1145/3209108.3209142.

17 Michael Karr. Affine relationships among variables of a program. *Acta Inf.*, 6:133–151, 1976.

18 Zachary Kincaid, John Cyphert, Jason Breck, and Thomas Reps. Non-linear reasoning for invariant synthesis. *Proc. ACM Program. Lang.*, 2(POPL), 2018. doi:10.1145/3158142.

19 K. Rustan M. Leino. Dafny: An automatic program verifier for functional correctness. In *Proc. International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-16), Revised Selected Papers*, volume 6355 of *Lecture Notes in Computer Science*, pages 348–370. Springer, 2010. doi:10.1007/978-3-642-17511-4_20.

20 Zohar Manna, Stephen Nes, and Jean Vuillemin. Inductive methods for proving properties of programs. *Commun. ACM*, 16(8):491–502, 1973. doi:10.1145/355609.362336.

21 Antoine Miné. Tutorial on static inference of numeric invariants by abstract interpretation. *Foundations and Trends in Programming Languages*, 4(3-4):120–372, 2017. doi:10.1561/2500000034.

22 Antoine Miné, Jason Breck, and Thomas W. Reps. An algorithm inspired by constraint solvers to infer inductive invariants in numeric programs. In *Proc. 25th European Symposium on Programming (ESOP'16)*, volume 9632 of *Lecture Notes in Computer Science*, pages 560–588. Springer, 2016. doi:10.1007/978-3-662-49498-1_22.

23 David Monniaux. On the decidability of the existence of polyhedral invariants in transition systems. *arXiv CoRR*, abs/1709.04382, 2017. arXiv:1709.04382.

24 David Monniaux. On the decidability of the existence of polyhedral invariants in transition systems. *Acta Inf.*, 56(4):385–389, 2019. doi:10.1007/s00236-018-0324-y.

25 Christian Müller, Helmut Seidl, and Eugen Zalinescu. Inductive invariants for noninterference in multi-agent workflows. In *Proc. 31st IEEE Computer Security Foundations Symposium (CSF'18)*, pages 247–261. IEEE Computer Society, 2018. doi:10.1109/CSF.2018.00025.

26 Markus Müller-Olm and Helmut Seidl. A note on Karr's algorithm. In *Proceedings 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, volume 3142 of *Lecture Notes in Computer Science*, pages 1016–1028. Springer, 2004. doi:10.1007/978-3-540-27836-8_85.

27 Peter Naur. Proof of algorithms by general snapshots. *BIT Numerical Mathematics*, 6(4):310–316, 1966. doi:10.1007/BF01966091.

28 Oded Padon, Neil Immerman, Sharon Shoham, Aleksandr Karbyshev, and Mooly Sagiv. Decidability of inferring inductive invariants. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'16)*, pages 217–231. ACM, 2016. doi:10.1145/2837614.2837640.

29 David Park. Fixpoint induction and proofs of program properties. *Machine Intelligence*, 5, 1969.

30 David Park. On the semantics of fair parallelism. In Dines Bjøorner, editor, *Abstract Software Specifications*, pages 504–526. Springer Berlin Heidelberg, 1980.

31 Sharon Shoham. Undecidability of inferring linear integer invariants. *arXiv CoRR*, abs/1812.01069, 2018. arXiv:1812.01069.

32 A. Thakur, A. Lal, J. Lim, and T. Reps. PostHat and all that: Automating abstract interpretation. *Electronic Notes in Theoretical Computer Science*, 311:15–32, 2015. Fourth Workshop on Tools for Automatic Program Analysis (TAPAS 2013). doi:10.1016/j.entcs.2015.02.003.

A Appendix

A.1 When Safety = Abstract Invariance?

Padon et al. [28, Section 9] in their investigation on the decidability of inferring inductive invariants state that "*Usually completeness for abstract interpretation means that the abstract domain is precise enough to prove all interesting safety properties, e.g.,* [15]. *In our terms, this means that* SAFE = INV*, that is, that all safe programs have an inductive invariant expressible in the abstract domain.*" As a by-product of the results in Section 4.2, we are able to give a formal justification and statement of this informal characterization of completeness.

Let $\mathcal{F} \subseteq C \to C$ be a class of monotonic functions, $\mathcal{S} \subseteq C$ be some set of safety properties and $\mathcal{A} \subseteq C$ be an abstract domain of program properties. Let us define:

$$\textsc{safe}[\mathcal{F},\mathcal{S}] \triangleq \{\langle f, s\rangle \in \mathcal{F}\times\mathcal{S} \mid \mathsf{lfp}(f) \leq_C s\}$$
$$\textsc{inv}[\mathcal{F},\mathcal{S},\mathcal{A}] \triangleq \{\langle f, s\rangle \in \mathcal{F}\times\mathcal{S} \mid \exists a \in \mathcal{A}.\, fa \leq_C a \wedge a \leq_C s\}$$

so that in our model $\textsc{safe}[\mathcal{F},\mathcal{S}]$ and $\textsc{inv}[\mathcal{F},\mathcal{S},\mathcal{A}]$ play the role of, resp., "safe programs" and "programs having an inductive invariant expressible in $\mathcal{A}$". As a consequence of Theorem 4.1, we derive the following characterization.

▶ **Corollary A.1.** *Assume that $\mathcal{A}$ satisfies Assumption 3.1 for some GI $\langle C, \alpha, \gamma, \mathcal{A}\rangle$.*
(a) *Assume that $\mathcal{S} \subseteq \mathcal{A}$. Then,* $\textsc{safe}[\mathcal{F},\mathcal{S}] = \textsc{inv}[\mathcal{F},\mathcal{S},\mathcal{A}]$ *iff* $\forall f \in \mathcal{F}.\alpha(\mathsf{lfp}(f)) = \mathsf{lfp}(\alpha f\gamma)$.
(b) $\textsc{safe}[\mathcal{F},\mathcal{S}] = \textsc{inv}[\mathcal{F},\mathcal{S},\mathcal{A}]$ *iff* $\forall f \in \mathcal{F}.\, \mathsf{lfp}(f) = \gamma(\mathsf{lfp}(\alpha f\gamma))$.

Proof. Point (a) follows by Theorem 4.1 (a), since $\mathcal{S} \subseteq \mathcal{A}$ is assumed to hold. Point (b) follows by Theorem 4.1 (b). ◀

Corollary A.1 therefore provides a precise equivalence of the SAFE $=^?$ INV problem, as stated by Padon et al. [28], with fixpoint completeness (strong fixpoint completeness, in case (b)) in abstract interpretation.

A.2 Proofs

Proof of Lemma 3.2. Let us first recall that in a GI, for all $a, a' \in A$, $a \leq_A a' \Leftrightarrow \gamma(a) \leq_C \gamma(a')$ holds.
(a) ($\Leftarrow$) We have that:

$$\begin{aligned}
\exists a \in A.\, f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C c' \Leftrightarrow \quad & \text{[by GC]}\\
\exists a \in A.\, \alpha f\gamma(a) \leq_A a \wedge \gamma(a) \leq_C c' \Rightarrow \quad & \text{[by } Fx \leq x \Rightarrow \mathsf{lfp}(F) \leq x\text{]}\\
\exists a \in A.\, \mathsf{lfp}(\alpha f\gamma) \leq_A a \wedge \gamma(a) \leq_C c' \Leftrightarrow \quad & \text{[by GI]}\\
\exists a \in A.\, \gamma(\mathsf{lfp}(\alpha f\gamma)) \leq_C \gamma(a) \wedge \gamma(a) \leq_C c' \Rightarrow \quad & \text{[by transitivity]}\\
\gamma(\mathsf{lfp}(\alpha f\gamma)) \leq_C c' &
\end{aligned}$$

($\Rightarrow$) Define $a \triangleq \mathsf{lfp}(\alpha f\gamma) \in A$. It turns out that $\alpha f\gamma(a) \leq_A a$ so that, by GC, $f\gamma(a) \leq_C \gamma(a)$, and, by hypothesis, $\gamma(a) \leq_C c'$.
(b) It turns out that:

$$\begin{aligned}
\exists a \in A.\, f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C \gamma(a') \Leftrightarrow \quad & \text{[By Lemma 3.2 (a)]}\\
\gamma(\mathsf{lfp}(\alpha f\gamma)) \leq_C \gamma(a') \Leftrightarrow \quad & \text{[by GI]}\\
\mathsf{lfp}(\alpha f\gamma) \leq_A a' &
\end{aligned}$$

◀

Proof of Corollary 3.4. By Lemma 3.2 (a), because $\lambda x \in A.\alpha(c) \vee_A \alpha f\gamma(x) = \lambda x \in A.\alpha(c \vee_C f\gamma(x))$ is the best correct approximation of $\lambda x \in C.c \vee_C f(x)$. ◀

Proof of Corollary 3.5. The hypotheses guarantee that the procedure AInv is a terminating algorithm, in particular because the sequence of computed iterates i is an ascending chain in A. If the algorithm AInv outputs i then $i = \mathsf{lfp}(\lambda a.\alpha(c) \vee_A \alpha f\gamma(a)) \leq_A a'$, so that $i = \wedge\{a \in A \mid \alpha(c) \leq_A i, \alpha f\gamma(i) \leq_A i, i \leq_A a'\}$, that is, i is the least inductive invariant in A for f and $\langle c, \gamma(a')\rangle$. If the algorithm AInv outputs "no abstract inductive invariant for f and $\langle c, \gamma(a')\rangle$" then there exists $j \in \mathbb{N}$ such that $(\lambda a.\alpha(c) \vee_A \alpha f\gamma(a))^j(\bot_A) \not\leq_A a'$, so that $\mathsf{lfp}(\lambda a.\alpha(c) \vee_A \alpha f\gamma(a)) \not\leq_A a'$, that is, there exists no inductive invariant in A for f and $\langle c, \gamma(a')\rangle$. ◀

Proof of Theorem 4.1.

(a) ($\Rightarrow$): By Lemma 3.2 (a).
($\Leftarrow$): Since $\mathsf{lfp}(f) \leq_C \mathsf{lfp}(f)$ holds, we have that $\exists a \in A.\ f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C \mathsf{lfp}(f)$. Thus, by Lemma 3.2 (a), $\gamma(\mathsf{lfp}(\alpha f\gamma)) \leq_C \mathsf{lfp}(f)$ follows. On the other hand, $\mathsf{lfp}(f) \leq \gamma(\mathsf{lfp}(\alpha f\gamma))$ always holds because the pointwise correctness of $\alpha f\gamma$ implies $\alpha(\mathsf{lfp}(f)) \leq_A \mathsf{lfp}(\alpha f\gamma)$, hence, by GC, $\mathsf{lfp}(f) \leq \gamma(\mathsf{lfp}(\alpha f\gamma))$ follows.

(b) ($\Rightarrow$): By Lemma 3.2 (b) because $\mathsf{lfp}(\alpha f\gamma) \leq_A a' \Leftrightarrow \alpha(\mathsf{lfp}(f)) \leq a' \Leftrightarrow \mathsf{lfp}(f) \leq_C \gamma(a')$.
($\Leftarrow$): We consider $a' \triangleq \alpha(\mathsf{lfp}(f))$, so that, $\mathsf{lfp}(f) \leq_C \gamma(a')$ holds and by the equivalence of the hypothesis, $\exists a \in A.\ f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C \gamma\alpha(\mathsf{lfp}(f))$ holds. This implies, by GI, that $\exists a \in A.\ \alpha f\gamma(a) \leq_C a \wedge a \leq_A \alpha(\mathsf{lfp}(f))$. By the inductive invariant principle (1), this implies that (actually, is equivalent to) $\mathsf{lfp}(\alpha f\gamma) \leq \alpha(\mathsf{lfp}(f))$. Furthermore, $\alpha(\mathsf{lfp}(f)) \leq_A \mathsf{lfp}(\alpha f\gamma)$ always holds, therefore proving that $\alpha(\mathsf{lfp}(f)) = \mathsf{lfp}(\alpha f\gamma)$. ◀

Proof of Lemma 4.2. We have that:

$$\begin{aligned}
\exists a \in A. f\gamma(a) \leq_C \gamma(a) \wedge \gamma(a) \leq_C \gamma\alpha(\mathsf{lfp}(f)) \Leftrightarrow & \quad \text{[by GI]} \\
\exists a \in A. \alpha f\gamma(a) \leq_A a \wedge a \leq_A \alpha(\mathsf{lfp}(f)) \Leftrightarrow & \quad \text{[by (1) for } \alpha f\gamma\text{]} \\
\mathsf{lfp}(\alpha f\gamma) \leq_A \alpha(\mathsf{lfp}(f)) \Leftrightarrow & \quad \text{[as } \alpha(\mathsf{lfp}(f)) \leq_A \mathsf{lfp}(\alpha f\gamma)\text{]} \\
\alpha(\mathsf{lfp}(f)) = \mathsf{lfp}(\alpha f\gamma) & \qquad \blacktriangleleft
\end{aligned}$$

Proof of Lemma 6.1.

(a) If $s \sqsubseteq_L s'$ and $t \in \breve{\mu}_L(\{s\})$ then $t \in \breve{\mu}_L(\{s'\}) = \cap\{\phi \in L \mid s' \in \phi\}$: if $\phi \in L$ and $s' \in \phi$ then $s \in \phi$, so that, since $t \in \breve{\mu}_L(\{s\})$, $t \in \phi$. Conversely, if $\breve{\mu}_L(\{s\}) \subseteq \breve{\mu}_L(\{s'\})$, $\phi \in L$ and $s' \in \phi$, then, since $s \in \breve{\mu}_L(\{s'\})$, $s \in \phi$.

(b) By (a), we equivalently prove that $\langle\{\breve{\mu}_L(\{s\}) \mid s \in \Sigma\}, \subseteq\rangle$ is a wqo iff $\langle L, \subseteq\rangle$ is a wqo.

($\Rightarrow$): [28, Lemma 4.6] proves $\langle L, \subseteq\rangle$ is well-founded, we additionally show that it does not contain infinite antichains. By contradiction, assume that $\{\phi_i\}_{i\in\mathbb{N}}$ is an infinite antichain in $\langle L, \subseteq\rangle$. Thus, for all $i \neq j$, $\phi_i \not\subseteq \phi_j$ and $\phi_j \not\subseteq \phi_i$, so that there exist $s_{i,j} \in \phi_i \smallsetminus \phi_j$ and $s_{j,i} \in \phi_j \smallsetminus \phi_i$. From $s_{j,i} \in \phi_j$ we obtain that $\breve{\mu}_L(\{s_{j,i}\}) \subseteq \breve{\mu}_L(\phi_j) = \phi_j$. From $s_{i,j} \notin \phi_j$, we obtain that $s_{i,j} \in \breve{\mu}_L(\{s_{i,j}\}) \not\subseteq \phi_{j,i}$. It turns out that $\breve{\mu}_L(\{s_{i,j}\}) \not\subseteq \breve{\mu}_L(\{s_{j,i}\})$, otherwise from $s_{i,j} \in \breve{\mu}_L(\{s_{i,j}\}) \subseteq \breve{\mu}_L(\{s_{j,i}\}) \subseteq \phi_j$ we would obtain the contradiction $s_{i,j} \in \phi_j$. Dually, $\breve{\mu}_L(\{s_{j,i}\}) \not\subseteq \breve{\mu}_L(\{s_{i,j}\})$ holds. Thus, for any $i \in \mathbb{N}$, $\{\breve{\mu}_L(\{s_{i,j}\}) \mid j \in \mathbb{N}, j \neq i\}$ is an infinite antichain in $\langle\{\breve{\mu}_L(\{s\}) \mid s \in \Sigma\}, \subseteq\rangle$, which is a contradiction.

($\Leftarrow$): $\langle\{\breve{\mu}_L(\{s\}) \mid s \in \Sigma\}, \subseteq\rangle$ is trivially a wqo because $\{\breve{\mu}_L(\{s\}) \mid s \in \Sigma\} \subseteq L$ and L is a wqo.

(c) Assume that for all $s \in \Sigma$, $Av_L(s) \in L$. Let us show that for all $S \in \wp(\Sigma)$, $\cap_{s\in S} Av_L(s) = Av_L(S)$.

($\supseteq$): Let $t \in \phi$ for some $\phi \in L$ such that $\phi \subseteq \neg S$. Then, for all $s \in S$, $\phi \subseteq Av_L(s)$, so that $t \in \cap_{s\in S} Av_L(s)$.

($\subseteq$): Let $t \in \cap_{s\in S} Av_L(s)$. For all $s \in S$, there exists $\phi_s \in L$ such that $\phi_s \subseteq \neg\{s\}$ and $t \in \phi_s$. Thus, $\cap_{s\in S}\phi_s \in L$ and $t \in \cap_{s\in S}\phi_s \subseteq \neg S$, meaning that $t \in Av_L(S)$.

Thus, since L is assumed to be closed under arbitrary intersections we obtain that $Av_L(S) = \cap_{s\in S} Av_L(s) \in L$. Consider now $\Phi \subseteq L$. Then, $Av_L(\neg(\cup\Phi)) = \cup\{\phi \in L \mid \phi \subseteq \neg\neg(\cup\Phi)\} = \cup\{\phi \in L \mid \phi \subseteq \cup\Phi\} = \cup\Phi$, so that $\cup\Phi \in L$.

Conversely, if L is closed under arbitrary unions then $Av_L(s) = \cup\{\phi \in L \mid \phi \subseteq \neg\{s\}\} \in L$.

(d) Since $\sqsubseteq_L$ is a quasi-order relation, its down-closure δ_L is an upper closure on $\langle \wp(\Sigma), \subseteq\rangle$. By (a), $\delta_L(X) = \{s \in \Sigma \mid \exists s' \in X.s \sqsubseteq_L s'\} = \{s \in \Sigma \mid \exists s' \in X.\breve{\mu}_L(\{s\}) \subseteq \breve{\mu}_L(\{s'\})\} = \{s \in \Sigma \mid \exists s' \in X.s \in \breve{\mu}_L(\{s'\})\} = \cup_{s\in X}\breve{\mu}_L(\{s\})$. By (c), since L is closed under arbitrary unions, the upper closure $\breve{\mu}_L$ is additive, so that $\cup_{s\in X}\breve{\mu}_L(\{s\}) = \breve{\mu}_L(\cup_{s\in X}\{s\}) = \breve{\mu}_L(X)$, consequently $\delta_L(X) = \breve{\mu}_L(X)$. In particular, $\delta_L(\phi) = \phi \Leftrightarrow \breve{\mu}_L(\phi) = \phi \Leftrightarrow \phi \in L$.

(e) By Lemma 6.1 (d), $\mathsf{lfp}(\lambda X.\, \mathsf{post}(X) \cup \delta_L(X) \cup \Sigma_0) = \mathsf{lfp}(\lambda X.\, \mathsf{post}(X) \cup \breve{\mu}_L(X) \cup \Sigma_0)$. It turns out that

$$\begin{aligned}
\Sigma_0 \cup \mathsf{post}(X) \cup \delta_L(X) \subseteq X \Leftrightarrow & \quad \text{[by Lemma 6.1 (d)]}\\
\Sigma_0 \cup \mathsf{post}(X) \cup \breve{\mu}_L(X) \subseteq X \Leftrightarrow & \quad \text{[by set theory]}\\
\Sigma_0 \subseteq X \wedge \mathsf{post}(X) \subseteq X \wedge \breve{\mu}_L(X) \subseteq X \Leftrightarrow & \quad \text{[as } \breve{\mu}_L \text{ is a uco]}\\
\Sigma_0 \subseteq X \wedge \mathsf{post}(X) \subseteq X \wedge \breve{\mu}_L(X) = X \Leftrightarrow & \quad \text{[as } \breve{\mu}_L(X) = X\text{]}\\
\Sigma_0 \subseteq X \wedge \mathsf{post}(\breve{\mu}_L(X)) \subseteq X \wedge \breve{\mu}_L(X) = X \Leftrightarrow & \quad \text{[by set theory]}\\
\Sigma_0 \cup \mathsf{post}(\breve{\mu}_L(X)) \cup \breve{\mu}_L(X) \subseteq X &
\end{aligned}$$

Thus, by Knaster-Tarski theorem, $\mathsf{lfp}(\lambda X.\Sigma_0 \cup \mathsf{post}(\breve{\mu}_L(X)) \cup \breve{\mu}_L(X)) = \mathsf{lfp}(\lambda X.\Sigma_0 \cup \mathsf{post}(X) \cup \delta_L(X))$. We also have that:

$$\begin{aligned}
\Sigma_0 \cup \mathsf{post}(\breve{\mu}_L(X)) \cup \breve{\mu}_L(X) \subseteq X \Leftrightarrow & \quad \text{[by the equivalences above]}\\
\Sigma_0 \cup \mathsf{post}(\breve{\mu}_L(X)) \cup \breve{\mu}_L(X) \subseteq X = \breve{\mu}_L(X) \Leftrightarrow & \quad \text{[by set theory]}\\
\Sigma_0 \cup \mathsf{post}(\breve{\mu}_L(X)) \subseteq X = \breve{\mu}_L(X) \Leftrightarrow & \quad \text{[as } \breve{\mu}_L(X) = X\text{]}\\
\Sigma_0 \cup \mathsf{post}(X) \subseteq X = \breve{\mu}_L(X) \Leftrightarrow & \quad \text{[as } \breve{\mu}_L \text{ is a uco]}\\
\breve{\mu}_L(\Sigma_0 \cup \mathsf{post}(X)) \subseteq X = \breve{\mu}_L(X) \Leftrightarrow & \quad \text{[by set theory]}\\
\breve{\mu}_L(\Sigma_0 \cup \mathsf{post}(X)) \subseteq X &
\end{aligned}$$

Thus, by applying Knaster-Tarski theorem, $\mathsf{lfp}(\breve{\mu}_L(\Sigma_0 \cup \mathsf{post}(X))) \subseteq \mathsf{lfp}(\lambda X.\Sigma_0 \cup \mathsf{post}(\breve{\mu}_L(X)) \cup \breve{\mu}_L(X))$ follows. Moreover, if $F \triangleq \mathsf{lfp}(\breve{\mu}_L(\Sigma_0 \cup \mathsf{post}(X)))$, so that $F = \breve{\mu}_L(F) = \Sigma_0 \cup \mathsf{post}(F)$, then $\Sigma_0 \cup \mathsf{post}(\breve{\mu}_L(F)) \cup \breve{\mu}_L(F) = \Sigma_0 \cup \mathsf{post}(F) \cup \breve{\mu}_L(F) = F$, and this implies that $\mathsf{lfp}(\lambda X.\Sigma_0 \cup \mathsf{post}(\breve{\mu}_L(X)) \cup \breve{\mu}_L(X)) \subseteq \mathsf{lfp}(\breve{\mu}_L(\Sigma_0 \cup \mathsf{post}(X)))$. Therefore, $\mathsf{lfp}(\lambda X.\Sigma_0 \cup \mathsf{post}(\breve{\mu}_L(X)) \cup \breve{\mu}_L(X)) = \mathsf{lfp}(\breve{\mu}_L(\Sigma_0 \cup \mathsf{post}(X)))$. ◀

Proof of Corollary 6.2. It turns out that

$$\begin{aligned}
\mathsf{lfp}(\lambda X.\breve{\mu}_L(\Sigma_0 \cup \mathsf{post}(X))) \subseteq P \Leftrightarrow & \quad \text{[by Lemma 3.2 (a) for ucos]}\\
\exists \phi \in \breve{\mu}_L(\wp(\Sigma)).\Sigma_0 \cup \mathsf{post}(\phi) \subseteq \phi \wedge \phi \subseteq P \Leftrightarrow & \quad \text{[as } \breve{\mu}_L(\wp(\Sigma)) = L\text{]}\\
\exists \phi \in L.\Sigma_0 \subseteq \phi \wedge \mathsf{post}(\phi) \subseteq \phi \wedge \phi \subseteq P &
\end{aligned}$$

◀

Proof of Lemma 6.3. The proof of Lemma 6.1 (e) shows that $\mathsf{lfp}(\lambda X.\Sigma_0 \cup \mathsf{post}(\breve{\mu}_L(X)) \cup \breve{\mu}_L(X)) = \mathsf{lfp}(\breve{\mu}_L(\Sigma_0 \cup \mathsf{post}(X)))$. Moreover, since $\breve{\mu}_L$ is additive and $\breve{\mu}_L(\Sigma_0) = \Sigma_0$, we also have that $\breve{\mu}_L(\Sigma_0 \cup \mathsf{post}(X)) = \breve{\mu}_L(\Sigma_0) \cup \breve{\mu}_L(\mathsf{post}(X)) = \Sigma_0 \cup \breve{\mu}_L(\mathsf{post}(X))$, and this allows us to conclude. ◀

▶ **Lemma A.2.** *Let $I \in L$ and $\Sigma_0 \subseteq I$.*

(a) *there exists a counterexample to inductiveness of I iff $I \not\subseteq \widetilde{\mathsf{pre}}(I) \cap P$ iff $I \neq \hat{\mu}_L(\widetilde{\mathsf{pre}}(I) \cap I \cap P)$.*

(b) *If $s \in \Sigma$ is a counterexample to inductiveness of I then $\hat{\mu}_L(\widetilde{\mathsf{pre}}(I) \cap I \cap P) \subseteq Av_L(s)$.*

Proof.

(a) Under the assumption that $\Sigma_0 \subseteq I$, s is a counterexample to inductiveness of I iff $(s \in I \wedge \mathsf{post}(s) \not\subseteq I) \vee s \in I \cap \neg P$. Observe that $\mathsf{post}(s) \not\subseteq I$ iff $s \notin \widetilde{\mathsf{pre}}(I)$, so that $\exists s \in \Sigma. s \in I \wedge \mathsf{post}(s) \not\subseteq I$ iff $I \not\subseteq \widetilde{\mathsf{pre}}(I)$. Hence, $\exists s \in \Sigma. (s \in I \wedge \mathsf{post}(s) \not\subseteq I) \vee s \in I \cap \neg P$ iff $I \not\subseteq \widetilde{\mathsf{pre}}(I) \cap P$. Also:

$$\begin{aligned}
I = \hat{\mu}_L(\widetilde{\mathsf{pre}}(I) \cap I \cap P) \Leftrightarrow & \quad [\text{as } I \in L] \\
I = \hat{\mu}_L(I) = \hat{\mu}_L(\widetilde{\mathsf{pre}}(I) \cap I \cap P) \Leftrightarrow & \quad [\text{as } \widetilde{\mathsf{pre}}(I) \cap I \cap P \subseteq I] \\
I = \hat{\mu}_L(I) \subseteq \hat{\mu}_L(\widetilde{\mathsf{pre}}(I) \cap I \cap P) \Leftrightarrow & \quad [\text{as } \hat{\mu}_L \text{ is a lco}] \\
I = \hat{\mu}_L(I) \subseteq \widetilde{\mathsf{pre}}(I) \cap I \cap P \Leftrightarrow & \\
I \subseteq \widetilde{\mathsf{pre}}(I) \cap P &
\end{aligned}$$

(b) The proof of point (a) shows that if $s \in \Sigma$ is a counterexample to inductiveness of I then $s \in I$ and $s \notin \widetilde{\mathsf{pre}}(I) \cap P$. Then, $\widetilde{\mathsf{pre}}(I) \cap P \subseteq \neg\{s\}$, so that, by monotonicity of $\hat{\mu}_L$, $\hat{\mu}_L(\widetilde{\mathsf{pre}}(I) \cap P) \subseteq \hat{\mu}_L(\neg\{s\})$ and, in turn, $\hat{\mu}_L(\widetilde{\mathsf{pre}}(I) \cap I \cap P) \subseteq \hat{\mu}_L(\neg\{s\}) = Av_L(s)$. ◀

Proof of Theorem 6.4. Consider the following variation of Algorithm 1:

Algorithm 4 A modification of Algorithm 1.

$I_4 := \Sigma$;
while $\Sigma_0 \subseteq I_4$ **do** // Invariant: $I_4 \in L$
  if $(I_4 \smallsetminus (\widetilde{\mathsf{pre}}(I_4) \cap P) = \varnothing)$ **then return** I_4 *is an inductive invariant in* L;
  choose $s \in I_4 \smallsetminus (\widetilde{\mathsf{pre}}(I_4) \cap P)$;
  $I_4 := I_4 \cap \hat{\mu}_L(\{s\})$;
return *no inductive invariant in* L;

Algorithm 1 returns $I_1 \in L$ iff $I_1 = \mathsf{gfp}(\lambda X. \hat{\mu}_L(\widetilde{\mathsf{pre}}(X) \cap X \cap P))$. In this case, by Lemma A.2 (a), since $\Sigma_0 \subseteq I_1$ holds, I_1 is an (actually, the greatest) inductive invariant in L. Otherwise, Algorithm 1 returns "no inductive invariant in L". By Lemma A.2 (a), Algorithm 4 returns $I_4 \in L$ iff $I_4 \subseteq \widetilde{\mathsf{pre}}(I_4) \cap P$ iff $I_4 = \hat{\mu}_L(\widetilde{\mathsf{pre}}(I_4) \cap I_4 \cap P)$, otherwise it returns "no inductive invariant in L". Algorithm 2 returns $I_2 \in L$ iff I_2 is an inductive invariant, otherwise it returns "no inductive invariant in L". Let I_k^n be the current candidate invariant of Algorithm $k \in \{1, 2, 4\}$ at its n-th iteration and I_k be the output invariant of Algorithm k. By Lemma A.2 (b), $I_1^n \subseteq I_4^n = I_2^n$, so that $I_1 \subseteq I_4 = I_2$. Since I_k are fixpoints of $\lambda X. \hat{\mu}_L(\widetilde{\mathsf{pre}}(X) \cap X \cap P)$ and I_1 is the greatest fixpoint, it turns out that $I_1 = I_4 = I_2$. ◀

Decidable Inductive Invariants for Verification of Cryptographic Protocols with Unbounded Sessions

Emanuele D'Osualdo
Imperial College London, UK
e.dosualdo@ic.ac.uk

Felix Stutz
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
fstutz@mpi-sws.org

Abstract

We develop a theory of decidable inductive invariants for an infinite-state variant of the Applied π-calculus, with applications to automatic verification of stateful cryptographic protocols with unbounded sessions/nonces. Since the problem is undecidable in general, we introduce *depth-bounded protocols*, a strict generalisation of a class from the literature, for which our decidable analysis is sound and complete. Our core contribution is a procedure to check that an invariant is inductive, which implies that every reachable configuration satisfies it. Our invariants can capture security properties like secrecy, can be inferred automatically, and represent an independently checkable certificate of correctness. We provide a prototype implementation and we report on its performance on some textbook examples.

2012 ACM Subject Classification Theory of computation → Program verification

Keywords and phrases Security Protocols, Infinite-State Verification, Ideal Completions for WSTS

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.31

Related Version Full version of the paper available at `https://arxiv.org/abs/1911.05430`.

Supplementary Material Tool available at `https://doi.org/10.5281/zenodo.3950846`

Funding *Emanuele D'Osualdo*: EU Horizon 2020 Marie Curie Individual Fellowship.
Felix Stutz: supported by Imperial College London and International Max Planck Research School for Computer Science.

Acknowledgements We would like to thank Alwen Tiu, Roland Meyer and Véronique Cortier for the useful feedback.

1 Introduction

Security protocols implement secure communication over insecure channels, by using cryptography. Despite underpinning virtually every communication over the internet, new flaws that compromise security are routinely discovered in deployed protocols. Automatic protocol verification is highly desirable, but also very challenging: the space of possible attacks is infinite. Indeed, even under the assumption of perfect cryptography, security properties are undecidable [21]. The most problematic feature for decidability is the necessity of considering unboundedly many fresh random numbers, called *nonces*, to distinguish between various sessions of the protocol. There has been a proliferation of verification tools [11, 3, 35, 10, 4, 26] which can be categorised according to the way the undecidability issue is resolved. A first approach is to only consider a bounded number of sessions, possibly missing attacks. A second is to over-approximate the protocol's behaviour by representing nonces with less precision, possibly reporting spurious attacks. A third is to implement semi-algorithms, accepting that the tools might never terminate on some protocols.

In this paper, we devise a sound and complete analysis, i.e. one that always terminates with a correct answer, without need for approximations. We obtain this by developing decision procedures for proving invariants of a rich sub-class of protocols with unbounded sessions/nonces. An invariant is any property that holds for every reachable configuration. We introduce *depth-bounded protocols*, a strict generalisation of the class of [18], and prove that a class of invariants, called downward-closed, can be effectively represented using expressions that we call *limits*. Our core technical results are a decision procedure for limit inclusion and an algorithm called $\widehat{\text{post}}$ that computes, from a limit L, a finite union of limits that represent the (infinite) set of configurations reached in one step from L. By using these two components, we obtain an algorithm to check if a limit is inductive, i.e. $\widehat{\text{post}}(L) \subseteq L$. An inductive limit that contains the initial configuration is guaranteed to be an invariant for the protocol. We show how to use this to prove a number of properties including depth-boundedness itself (a semantic property), secrecy, and control-state reachability.

We define depth-bounded protocols as a subclass of a variant of the Applied π-calculus [31], with support for user-defined cryptographic primitives, secure and public channels, stateful principals, a Dolev-Yao-style intruder [17] supporting modelling of dishonest participants and leaks of old keys. In particular, our results apply to any set of cryptographic primitives that satisfy some simple axioms; examples include (a)symmetric encryption, blind signatures, hashes, XOR. To gain intuition about depth-boundedness, consider the set of messages $\Gamma_n = \{\mathsf{e}(k_1)_{k_2}, \mathsf{e}(k_2)_{k_3}, \ldots \mathsf{e}(k_{n-1})_{k_n}\}$ which "chains" key k_1 to k_2, k_2 to k_3 and so on, obtaining an encryption chain of length n. A depth-bounded protocol cannot produce, or be tricked to produce, such chains of unbounded length. Note that, when computing depth, we only consider chains that are essential: the set $\Gamma_n \cup \{k_n\}$ for example has depth 1 (for any n) because it is equivalent to the set $\{k_1, \ldots, k_n\}$. When there is a bound d on the depth of reachable configurations, we say that the protocol is depth-bounded. We built a proof-of-concept prototype tool to evaluate the approach, showing that many textbook protocols fall into the depth-bounded class.

More precisely, bounding depth alone is not enough to obtain decidability [18]: one needs to bound the size of messages too. For type-compliant protocols [2, 13] message size can be bounded without excluding any security violation. More generally, for typical protocols (including all our benchmarks), our inductive invariants can be computed on the size-bounded model, and then generalised to invariants for the unrestricted version of the protocol.

Our approach has a number of notable properties. First, once a suitable inductive invariant has been found, it can be provided as a certificate of correctness that can be independently checked. Second, the search of a suitable invariant can be performed both automatically (with a trade-off between precision and performance) or interactively. Third, supporting unbounded nonces makes it possible to reason about properties like susceptibility to known-plaintext attacks (Section 3.1). Finally, even coarse invariants inferred with our method can be used to prune the search space of other model checking procedures.

Related work. The pure π-calculus version of depth-boundedness was originally proposed in [27] and developed in [24, 36, 37]. Our work builds directly on [18], which introduced depth-boundedness for the special case of secrecy of protocols using symmetric encryption. We generalise to a strictly more expressive class of primitives and properties, a result that requires much more sophisticated techniques and yields more powerful algorithms.

Our theory of invariants is framed in terms of ideal completions [22], which, to the best of our knowledge, has not been instantiated to cryptographic protocols before. Our decidability proofs introduce substantial new proof techniques to deal with an active intruder while being parametric on the cryptographic primitives.

Types [16, 14, 15] can be used to capture and generalise common safe usages of cryptographic primitives, and reduce verification to constraints which can be solved efficiently. We speculate that our domain of limits and the associated algorithms could be used to define an expressive class of solvable constraints that could be integrated in type systems.

In [14, 23] two classes of protocols with unbounded nonces are shown to enjoy decidable verification. They consider a less general calculus ((a)symmetric encryption only, with atomic keys), different properties (only secrecy [23], trace equivalence [14]), and restrict protocols using (similar) syntactic conditions, obtaining classes that are orthogonal to ours.

ProVerif [4] and Tamarin [26] are two mature tools with support for a wide range of cryptographic primitives and expressive properties, and handle unbounded sessions. Both programs employ semi-algorithms and may diverge on verification tasks. ProVerif is known to terminate on so-called *tagged protocols* [6] which are incomparable to depth-bounded protocols. Tamarin offers an interactive mode when a proof cannot be carried out automatically. To the best of our knowledge, there is no characterisation for a class of protocols on which Tamarin is guaranteed to terminate.

Outline. Section 2 introduces the formal model and depth-bounded protocols. Section 3 presents our main theoretical results. In Section 4 we report on experiments with our tool and discuss limitations. All omitted proofs can be found in the full version of the paper [19].

2 Formal Model

We introduce a variant of the Applied π-calculus as our formal model of protocols. Following the Dolev-Yao intruder model, we treat cryptographic primitives algebraically. Assume an enumerable set of *names* $a, b, \dots \in \mathcal{N}$. A signature Σ of *constructors*, is a finite set of symbols f with their arity $\mathrm{ar}(\mathsf{f}) \in \mathbb{N}$. The set of messages over Σ is the smallest set $\mathbb{M}^\Sigma$ which contains all names, and is closed under application of constructors. The domain of finite sets of messages is $\mathbb{K}^\Sigma := \wp_f(\mathbb{M}^\Sigma)$. We define $\mathrm{size}(\mathsf{f}(M_1, \dots, M_n)) := 1 + \max\{\mathrm{size}(M_i) \mid 1 \le i \le n\}$, $\mathrm{size}(a) := 1$, and $\mathrm{names}(a) := \{a\}$, $\mathrm{names}(\mathsf{f}(M_1, \dots, M_n)) := \bigcup_{i=1}^n \mathrm{names}(M_i)$. Given $X \subseteq \mathcal{N}$ and $s \in \mathbb{N}$, we define $\mathbb{M}_s^{\Sigma,X} := \{M \in \mathbb{M}^\Sigma \mid \mathrm{names}(M) \subseteq X, \mathrm{size}(M) \le s\}$. As is standard, Γ, Γ' and Γ, M stand for $\Gamma \cup \Gamma'$ and $\Gamma \cup \{M\}$ respectively.

A *substitution* is a finite partial function $\theta \colon \mathcal{N} \rightharpoonup \mathbb{M}^\Sigma$; we write $\theta = [M_1/x_1, \dots, M_n/x_n]$, abbreviated with $[\vec{M}/\vec{x}]$, for the substitution with $\theta(x_i) = M_i$ for all $1 \le i \le n$. We write $M\theta$ for the application of substitution θ to the message M, and extend the notation to sets of messages $\Gamma\theta := \{M\theta \mid M \in \Gamma\}$. A substitution θ is a *renaming of* $X \subseteq \mathcal{N}$ if it is defined on X, injective, and with $\theta(X) \subseteq \mathcal{N}$.

▶ **Definition 1** (Intruder model). *A* derivability relation *for a signature* Σ*, is a relation* $\vdash \subseteq \mathbb{K}^\Sigma \times \mathbb{M}^\Sigma$*. The pair* $\mathbb{I} = (\Sigma, \vdash)$ *is an* (effective) intruder model *if* $\vdash$ *is a (decidable) derivability relation for* Σ*, and for all* $M, N \in \mathbb{M}^\Sigma$*,* $\Gamma, \Gamma' \in \mathbb{K}^\Sigma$*,* $a \in \mathcal{N}$*:*

$$M \vdash M \tag{Id}$$

$$\Gamma \subseteq \Gamma' \land \Gamma \vdash M \implies \Gamma' \vdash M \tag{Mon}$$

$$\Gamma \vdash M \land \Gamma, M \vdash N \implies \Gamma \vdash N \tag{Cut}$$

$$M_1, \dots, M_n \vdash \mathsf{f}(M_1, \dots, M_n) \quad \text{for every } \mathsf{f} \in \Sigma \text{ with } \mathrm{ar}(\mathsf{f}) = n \tag{Constr}$$

$$\Gamma\theta \vdash M\theta \iff \Gamma \vdash M \quad \text{for any } \theta \text{ renaming of } \mathrm{names}(\Gamma) \tag{Alpha}$$

$$\Gamma, a \vdash M \land a \notin \mathrm{names}(\Gamma, M) \implies \Gamma \vdash M \tag{Relevancy}$$

The knowledge ordering *for* $\mathbb{I}$ *is the relation* $\le_{\mathsf{kn}} \subseteq \mathbb{K}^\Sigma \times \mathbb{K}^\Sigma$ *such that* $\Gamma_1 \le_{\mathsf{kn}} \Gamma_2$ *if and only if* $\forall M \in \mathbb{M}^\Sigma \colon \Gamma_1 \vdash M \implies \Gamma_2 \vdash M$*. We write* $\Gamma_1 \sim_{\mathsf{kn}} \Gamma_2$ *if* $\Gamma_1 \le_{\mathsf{kn}} \Gamma_2$ *and* $\Gamma_2 \le_{\mathsf{kn}} \Gamma_1$*.*

$$\frac{M \in \Gamma}{\Gamma \vdash M}\ \textsc{Id} \qquad \frac{\Gamma \vdash K}{\Gamma \vdash \mathsf{p}(K)}\ \textsc{Pub} \qquad \frac{\Gamma, (M,N), M, N \vdash M'}{\Gamma, (M,N) \vdash M'}\ \mathrm{P_L} \qquad \frac{\Gamma \vdash M \qquad \Gamma \vdash N}{\Gamma \vdash (M,N)}\ \mathrm{P_R}$$

$$\frac{\Gamma, \mathsf{e}(M)_K \vdash K \qquad \Gamma, \mathsf{e}(M)_K, M, K \vdash N}{\Gamma, \mathsf{e}(M)_K \vdash N}\ \mathrm{S_L} \qquad \frac{\Gamma \vdash M \qquad \Gamma \vdash K}{\Gamma \vdash \mathsf{e}(M)_K}\ \mathrm{S_R}$$

$$\frac{\Gamma, \mathsf{a}(M)_{\mathsf{p}(K)} \vdash K \qquad \Gamma, \mathsf{a}(M)_{\mathsf{p}(K)}, M, K \vdash N}{\Gamma, \mathsf{a}(M)_{\mathsf{p}(K)} \vdash N}\ \mathrm{A_L} \qquad \frac{\Gamma \vdash M \qquad \Gamma \vdash N}{\Gamma \vdash \mathsf{a}(M)_N}\ \mathrm{A_R}$$

Figure 1 Deduction rules for the derivability relation of $\mathbb{I}_{\mathsf{en}}$.

The first three axioms deal exclusively with what it means to be a deduction relation: what is known can be derived (Id); the more is known the more can be derived (Mon); what can be derived is known (Cut). The (Constr) axiom ensures the intruder is able to construct arbitrary messages by composing known messages. The (Alpha) axiom justifies α-renaming in our calculus. The (Relevancy) axiom allows us to only consider boundedly many nonces maliciously injected by the intruder, at each step of the protocol.

In the rest of the paper, unless otherwise specified, we fix an arbitrary effective intruder model $\mathbb{I}$ and omit the corresponding superscripts.

▶ **Proposition 2.** *Given $\Gamma_1, \Gamma_2 \in \mathbb{K}^{\Sigma}$, $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$ if and only if $\forall M \in \Gamma_1 \colon \Gamma_2 \vdash M$. As a consequence, if $\vdash$ is decidable, so is $\leq_{\mathsf{kn}}$.*

Our framework uses the derivability relation as a black box and does not rely on the way it is specified (e.g. with a rewriting system or a deduction system). It is possible to formalise as an effective intruder model cryptographic primitives such as XOR, hashes and blind signatures. We present here, for illustration, a model of (a)symmetric encryption, and elaborate on extensions in Appendix A. We find it convenient to specify it with a sequent calculus in the style of [34]. This is an alternative to more intuitive natural-deduction-style rules, which has the key advantage of being CUT-free, while admitting CUT. This simplifies considerably the proofs of properties (e.g. Lemma 21) of the intruder model.

▶ **Example 3** (Model of Encryption)**.** Symmetric and asymmetric encryption can be modelled using the signature $\Sigma_{\mathsf{en}} = \{(\cdot,\cdot)\,, \mathsf{e}(\cdot).\,, \mathsf{a}(\cdot).\,, \mathsf{p}(\cdot)\}$, where (M,N) pairs messages M and N, $\mathsf{e}(M)_N$ represents the message M encrypted with symmetric key N, $\mathsf{a}(M)_N$ represents the message M encrypted with asymmetric key N, and $\mathsf{p}(K)$ is the public key associated with the private key K. The intruder model for (a)symmetric encryption is the model $\mathbb{I}_{\mathsf{en}} = (\Sigma_{\mathsf{en}}, \vdash)$ where $\vdash$ is defined by the deduction rules in Figure 1.

▶ **Proposition 4.** *The model Σ_{en} is an effective intruder model.*

2.1 A Calculus for Cryptographic Protocols

A common approach to model cryptographic primitives is to consider both constructors (e.g. encryption) and destructors (e.g. decryption). Here messages only contain constructors, and "destruction" is represented by pattern matching. Fix a finite signature $\mathcal{Q}$ of process names (ranged over by Q) each of which has a fixed arity $\mathrm{ar}(\mathsf{Q}) \in \mathbb{N}$. A protocol specification consists of an initial process P and a finite set Δ of (possibly recursive) definitions of the form $\mathsf{Q}[x_1, \ldots, x_n] := A$, with $\mathrm{ar}(\mathsf{Q}) = n$, where the syntax of P and A follows the grammar:

$$P ::= \mathbf{0} \mid \nu x.P \mid P \parallel P \mid \langle M \rangle \mid \mathsf{Q}[\vec{M}] \quad \text{(process)}$$
$$A ::= \overline{a}\langle M \rangle \mid a(\vec{x} : M).P \mid A + A \quad \text{(action)}$$

$$\frac{\mathsf{Q}_1[\vec{M}_1] \triangleq \overline{c}\langle N[\vec{M}'/\vec{x}]\rangle.P_1 + A_1 \qquad \mathsf{Q}_2[\vec{M}_2] \triangleq c(\vec{x}:N).P_2 + A_2}{\nu\vec{a}.(\langle\Gamma\rangle \parallel \mathsf{Q}_1[\vec{M}_1] \parallel \mathsf{Q}_2[\vec{M}_2] \parallel C) \to_\Delta \nu\vec{a}.(\langle\Gamma\rangle \parallel P_1 \parallel P_2[\vec{M}'/\vec{x}] \parallel C)}\ \textsc{Comm}$$

$$\frac{P \equiv P' \to_\Delta Q' \equiv Q}{P \to_\Delta Q}\ \textsc{Struct} \qquad \frac{\mathsf{Q}[\vec{M}] \triangleq \overline{c}\langle M\rangle.P + A \qquad \Gamma \vdash c}{\nu\vec{a}.(\langle\Gamma\rangle \parallel \mathsf{Q}[\vec{M}] \parallel C) \to_\Delta \nu\vec{a}.(\langle\Gamma\rangle \parallel \langle M\rangle \parallel P \parallel C)}\ \textsc{PubOut}$$

$$\frac{\mathsf{Q}[\vec{M}] \triangleq c(\vec{x}:N).P + A \qquad \Gamma, \vec{y} \vdash N[\vec{M}'/\vec{x}] \qquad \Gamma \vdash c \qquad \vec{y}\text{ fresh}}{\nu\vec{a}.(\langle\Gamma\rangle \parallel \mathsf{Q}[\vec{M}] \parallel C) \to_\Delta \nu\vec{a}.\nu\vec{y}.(\langle\Gamma\rangle \parallel \langle\vec{y}\rangle \parallel P[\vec{M}'/\vec{x}] \parallel C)}\ \textsc{PubIn}$$

Figure 2 Operational semantics.

We use the vector notation $\vec{x} = x_1, \ldots, x_n$ for lists of pairwise distinct names. In an action $a(\vec{x}:M).P$, we call $\vec{x}:M$ the *pattern*, and P the *continuation*; processes $\mathsf{Q}[\vec{M}]$ are called *process calls*. If $\Gamma = \{M_1, \ldots, M_k\}$ is a finite set of messages, then $\langle\Gamma\rangle := \langle M_1\rangle \parallel \ldots \parallel \langle M_k\rangle$. We define $P^0 := \mathbf{0}$ and $P^{n+1} := P \parallel P^n$. For brevity, we assume the special name **in** is known to the intruder. The internal action $\boldsymbol{\tau}$, is an abbreviation for $\mathbf{in}(x:x)$, for a fresh x. Processes of the form $\langle M\rangle$ or $\mathsf{Q}[\vec{a}]$ are called *sequential*. The names $\vec{x}$ are bound in both $\nu\vec{x}.P$ and $c(\vec{x}:M).P$. We denote the set of free names of a term P with $\mathrm{fn}(P)$ and the set of bound names with $\mathrm{bn}(P)$. As is standard, we require, wlog, that $\mathrm{fn}(P) \cap \mathrm{bn}(P) = \emptyset$. When nesting restrictions $\nu\vec{x}.\nu\vec{y}.P$, we implicitly assume wlog that $\vec{x}$ and $\vec{y}$ are disjoint. We assume there is at most one definition for each $\mathsf{Q} \in \mathcal{Q}$, and that for each definition $\mathsf{Q}[x_1, \ldots, x_n] := A$, $\mathrm{fn}(A) \subseteq \{x_1, \ldots, x_n\}$. The set $\mathbb{P}$ consists of all processes over an underlying signature $\mathcal{Q}$.

Structural congruence. We write $\overset{\alpha}{=}$ for standard α-equivalence. Structural congruence, $\equiv$, is the smallest congruence relation that includes $\overset{\alpha}{=}$, and is associative and commutative with respect to $\parallel$ and $+$ with $\mathbf{0}$ as the neutral element, and satisfies the standard laws: $\nu a.\mathbf{0} \equiv \mathbf{0}$, $\nu a.\nu b.P \equiv \nu b.\nu a.P$, and $P \parallel \nu a.Q \equiv \nu a.(P \parallel Q)$ if $a \notin \mathrm{fn}(P)$. Every process P is congruent to a process in *standard form*:

$$\nu\vec{x}.\big(\langle M_1\rangle \parallel \cdots \parallel \langle M_m\rangle \parallel \mathsf{Q}_1[\vec{N}_1] \parallel \cdots \parallel \mathsf{Q}_k[\vec{N}_k]\big) \tag{SF}$$

where every name in $\vec{x}$ occurs free in some subterm. We write $\mathrm{sf}(P)$ for the standard form of P, which is unique up to α-equivalence, and associativity and commutativity of parallel. We abbreviate standard forms with $\nu\vec{x}.(\langle\Gamma\rangle \parallel Q)$ where all the active messages are collected in Γ, and Q is a parallel composition of process calls. Let $\mathrm{sf}(P)$ be the expression (SF), we define $\mathrm{msg}(P) = \{M_1, \ldots, M_m\} \cup \bigcup_{i=1}^{k} \vec{N}_i$. Thus $\mathrm{msg}(P)$ is the set of messages appearing in a term. When $m = 0, k = 0, \vec{x} = \emptyset$, the expression (SF) is $\mathbf{0}$.

Reduction semantics. One can think of standard forms $\nu\vec{x}.(\langle\Gamma\rangle \parallel Q)$ as runtime configurations of the protocol. They capture, at a specific point in time, the current relevant names (which encode nonces/keys/data), the knowledge of the intruder Γ, and the local state of each participant. A sequential term $\mathsf{Q}[\vec{N}]$ represents a single participant in control state Q with local knowledge of messages $\vec{N}$.

Principals can communicate through channels; a channel known by the intruder is considered insecure. An input action over an insecure channel can be fired if the intruder can produce any message that matches the action's pattern. An output $\overline{c}\langle M\rangle$ to an insecure channel c leaks message M to the intruder, who can decide to forward it angelically to a corresponding input over c (modelling an honest step) or hijack the communication.

$$\begin{aligned}
\mathsf{S}[a,b,k_{as},k_{bs}] &:= \mathbf{in}(n_a : (n_a,b)).\nu k.\big(\langle \mathsf{e}(k)_{k_{bs}}\rangle \parallel \big\langle \mathsf{e}(k)_{(n_a,k_{as})}\big\rangle \parallel \mathsf{S}[a,b,k_{as},k_{bs}]\big)\\
\mathsf{A}_1[a,b,k_{as}] &:= \boldsymbol{\tau}.\nu n_a.(\langle (n_a,b)\rangle \parallel \mathsf{A}_2[a,b,k_{as},n_a] \parallel \mathsf{A}_1[a,b,k_{as}])\\
\mathsf{A}_2[a,b,k_{as},n_a] &:= \mathbf{in}\big(k : \mathsf{e}(k)_{(n_a,k_{as})}\big).\mathsf{A}_3[a,b,k_{as},k]\\
\mathsf{A}_3[a,b,k_{as},k] &:= \mathbf{in}(n_b : \mathsf{e}(n_b)_k).\langle \mathsf{e}(n_b)_{(k,k)}\rangle\\
\mathsf{B}_1[a,b,k_{bs}] &:= \mathbf{in}(k : \mathsf{e}(k)_{k_{bs}}).\nu n_b.\big(\langle \mathsf{e}(n_b)_k\rangle \parallel \mathsf{B}_2[a,b,k_{bs},n_b,k] \parallel \mathsf{B}_1[a,b,k_{bs}]\big)\\
\mathsf{B}_2[a,b,k_{bs},n_b,k] &:= \mathbf{in}(\mathsf{e}(n_b)_{(k,k)}).\mathsf{Secret}[k]
\end{aligned}$$

Figure 3 Formal model of Example 9.

We write $\mathsf{Q}[\vec{M}] \triangleq A$ if $\mathsf{Q}[\vec{x}] := A' \in \Delta$ and $A \overset{\alpha}{=} A'[\vec{M}/\vec{x}]$, up to commutativity and associativity of $+$. The transition relation $\to_\Delta$ is defined in Figure 2. In Rule PubIn, $\vec{y}$ denotes all fresh names introduced by the intruder in this step. Thanks to (Relevancy), one can wlog ignore transitions where $\mathrm{fn}(\vec{y}) \not\subseteq \mathrm{fn}(\vec{M'})$, since unused names would simply not contribute to the intruder knowledge. The sets $\mathrm{reach}_\Delta(P) := \{Q \mid P \to_\Delta^* Q\}$ and $\mathrm{traces}_\Delta(P) := \{Q_0 \cdots Q_n \mid P \equiv_{\mathsf{kn}} Q_0 \to_\Delta \cdots \to_\Delta Q_n\}$ collect the processes reachable from P and all the transition sequences from P respectively, given the definitions Δ. We omit Δ when unambiguous.

▶ **Definition 5** ($\equiv_{\mathsf{kn}}$)**.** Knowledge congruence*, $P \equiv_{\mathsf{kn}} Q$, is the smallest congruence that includes $\equiv$ and such that $\langle \Gamma_1\rangle \equiv_{\mathsf{kn}} \langle \Gamma_2\rangle$ if $\Gamma_1 \sim_{\mathsf{kn}} \Gamma_2$.*

Knowledge congruence is also characterised by

$$P_1 \equiv_{\mathsf{kn}} P_2 \iff \mathrm{sf}(P_1) \overset{\alpha}{=} \nu\vec{x}.(\langle \Gamma_1\rangle \parallel Q) \wedge \mathrm{sf}(P_2) \overset{\alpha}{=} \nu\vec{x}.(\langle \Gamma_2\rangle \parallel Q) \wedge \Gamma_1 \sim_{\mathsf{kn}} \Gamma_2.$$

Intuitively, modulo derivability, two processes $P \equiv_{\mathsf{kn}} Q$ are indistinguishable to the intruder and to the principals. Formally, if $P \equiv_{\mathsf{kn}} Q$ then the transitions systems $(P, \to_\Delta)$ and $(Q, \to_\Delta)$ are isomorphic. We thus close the reduction semantics under knowledge congruence: we add the rule that if $P \equiv_{\mathsf{kn}} P' \to_\Delta Q' \equiv_{\mathsf{kn}} Q$ then $P \to_\Delta Q$.

While knowledge congruence captures when two configurations are essentially the same, knowledge embedding formalises the notion of "sub-configuration".

▶ **Definition 6** (Knowledge embedding)**.** *The* knowledge embedding *relation $P_1 \sqsubseteq_{\mathsf{kn}} P_2$ holds if $P_1 \equiv \nu\vec{x}.(\langle \Gamma_1\rangle \parallel Q)$, $P_2 \equiv \nu\vec{x}.\nu\vec{y}.(\langle \Gamma_2\rangle \parallel Q \parallel Q')$ and $\Gamma_1 \leq_{\mathsf{kn}} \Gamma_2$.*

▶ **Proposition 7.** *$P_1 \equiv_{\mathsf{kn}} P_2$ if and only if $P_1 \sqsubseteq_{\mathsf{kn}} P_2$ and $P_2 \sqsubseteq_{\mathsf{kn}} P_1$.*

▶ **Theorem 8.** *Knowledge embedding is a simulation, that is, for all P, P' and Q, if $P \to Q$ and $P \sqsubseteq_{\mathsf{kn}} P'$ then there is a Q' such that $P' \to Q'$ and $Q \sqsubseteq_{\mathsf{kn}} Q'$.*

▶ **Example 9.** Consider the following toy protocol, given in Alice&Bob notation, meant to establish a new session key K between A and B through a trusted server S:

(1) $A \to S$: N_A, B
(2) $S \to B$: $\mathsf{e}(K)_{(N_A,K_{AS})}, \mathsf{e}(K)_{K_{BS}}$
(3) $B \to A$: $\mathsf{e}(K)_{(N_A,K_{AS})}, \mathsf{e}(N_B)_K$
(4) $A \to B$: $\mathsf{e}(N_B)_{(K,K)}$

Figure 3 shows the protocol formalised in our calculus. Assume the initial state is

$$P_0 = \nu a,b,k_{as},k_{bs}.(\mathsf{S}[a,b,k_{as},k_{bs}] \parallel \mathsf{A}_1[a,b,k_{as}] \parallel \mathsf{B}_1[a,b,k_{bs}] \parallel \langle a,b\rangle).$$

Step (1) is initiated by A_1 which sends some new name n_a to the server; since communication is over an insecure channel, the message is just output without indicating the intended recipient. The server receives the message (or any message the intruder may decide to forge instead)

and outputs the fresh key k encrypted with k_{bs} (the long-term key between B and S) and with the pair (n_a, k_{as}) (note the use of non-atomic encryption keys). In the protocol, these two messages are sent to B but we model step (2) by B_1 which just receives the message relevant to B. The forwarding of $\mathsf{e}(k)_{(n_a,k_{as})}$ from S to A is performed by the intruder instead of B in the model.

In the last two steps, modelled by B_2 and A_3, B sends a nonce n_b encrypted with k, to challenge A to prove she knows k, which she does by sending back $\mathsf{e}(n_b)_{(k,k)}$. At this point, B is convinced that by encrypting messages with k they will be only accessible to A. We model this by making B_2 transition to $\mathsf{Secret}[k]$ after a successful challenge. We always assume the definition $\mathsf{Secret}[k] := \mathbf{in}(k).\mathsf{Leak}[k]$. A transition to $\mathsf{Leak}[k]$ is only possible when the intruder can derive k so we can check whether the secrecy assertion holds by checking that no reachable process contains a call to $\mathsf{Leak}[k]$.

Notice how A_1 and B_1 spawn both the continuation of the session and (recursively) a process ready to start a new session. This creates the possibility of an unbounded number of sessions, each of which will involve fresh n_a, n_b, and k.

Threat model. Our reduction semantics follows the Dolev-Yao attacker model in representing the intruder's interference: the intruder mediates every communication over insecure channels, is able to create new names and analyse and construct messages from all the messages that have been communicated insecurely so far. Threat models that go beyond Dolev-Yao include dishonest participants and compromised old session keys. These aspects are not embedded in the semantics, but can be modelled through the process definitions. If we wanted to model compromised keys in Example 9, for instance, we could modify the definition of B_2 to $\mathsf{B}_2[a, b, k_{bs}, n_b, k] := \mathbf{in}(\mathsf{e}(n_b)_{(k,k)}).\mathsf{Secret}[k] + \mathbf{in}(\mathsf{e}(n_b)_{(k,k)}).\langle k\rangle$ which makes a non-deterministic choice to declare k a secret, or to consider it as old and reveal it.

▶ Remark 10 (Implementable patterns). Our calculus represents message deconstruction (e.g. decryption) with pattern matching. However, general pattern matching is too powerful: a pattern like $\mathbf{in}(x, k : \mathsf{e}(x)_k)$ would obtain *both* the key k and the plaintext x from an encrypted message! This is only a modelling problem: one should make sure all patterns can be implemented using the cryptographic primitives. Consider a pattern $\vec{x} : M$ and let $Z = \mathrm{names}(M) \setminus \vec{x}$; the pattern is *implementable*, if, for all $\theta \colon Z \rightharpoonup \mathbb{M}$, we have $M\theta, Z\theta \vdash y$ for all $y \in \vec{x}$.

2.2 Depth-Bounded Protocols

We can now define the class of depth-bounded protocols, a strict generalisation of the notion in [18]. While the definitions of [18] depend on fixing the intruder to symmetric encryption only, here we define it fully parametrically to the intruder model.

▶ **Definition 11** (Depth). *The* nesting of restrictions *of a term is given by the function* $\mathrm{nest}_\nu(\mathsf{Q}[\vec{a}]) := \mathrm{nest}_\nu(\langle M\rangle) := \mathrm{nest}_\nu(\mathbf{0}) := 0$, $\mathrm{nest}_\nu(\nu x.P) := 1 + \mathrm{nest}_\nu(P)$, $\mathrm{nest}_\nu(P \parallel Q) := \max(\mathrm{nest}_\nu(P), \mathrm{nest}_\nu(Q))$. *The* depth *of a term is defined as the minimal nesting of restrictions in its knowledge congruence class,* $\mathrm{depth}(P) := \min\{\mathrm{nest}_\nu(Q) \mid Q \equiv_{\mathsf{kn}} P\}$.

▶ **Lemma 12.** *Every Q is α-equivalent to a process Q' such that $|\mathrm{bn}(Q')| \leq \mathrm{nest}_\nu(Q)$.*

Consider for example $P = \nu a, b, c.(\langle a\rangle \parallel \langle \mathsf{e}(b)_a\rangle \parallel \langle \mathsf{e}(c)_b\rangle \parallel \langle c\rangle)$ which has $\mathrm{nest}_\nu(P) = 3$. The process P is knowledge-congruent to $Q = (\nu a.\langle a\rangle \parallel \nu b.\langle b\rangle \parallel \nu c.\langle c\rangle)$ which has $\mathrm{nest}_\nu(Q) = 1$; this gives us $\mathrm{depth}(P) = \mathrm{nest}_\nu(Q) = 1$. Although $\mathrm{bn}(Q) = \{a, b, c\}$, by α-renaming all

names to x we obtain $Q' = (\nu x.\langle x\rangle \parallel \nu x.\langle x\rangle \parallel \nu x.\langle x\rangle)$ which has the property $|\mathrm{bn}(Q')| \leq \mathrm{nest}_\nu(Q) \leq \mathrm{depth}(P)$. More generally, Lemma 12 says that processes of depth k can always be represented using at most k unique names, by reusing names in disjoint scopes.

Let $\mathbb{S}_s := \{P \in \mathbb{P} \mid \forall M \in \mathrm{msg}(P)\colon \mathrm{size}(M) \leq s\}$ be the set of processes containing messages of size at most s. The set $\mathbb{D}^X_{s,k}$ is the set of processes of depth at most $k \in \mathbb{N}$, with free names in X, and messages not exceeding size s:

$$\mathbb{D}^X_{s,k} := \{P \in \mathbb{S}_s \mid \mathrm{fn}(P) \subseteq X, \exists Q \in \mathbb{S}_s\colon Q \equiv_{\mathsf{kn}} P \wedge \mathrm{nest}_\nu(Q) \leq k\}.$$

When starting from some initial process P_0, every reachable process P has $\mathrm{fn}(P) \subseteq \mathrm{fn}(P_0)$ so X can always be fixed to be $\mathrm{fn}(P_0)$. We therefore omit X from the superscripts to unclutter notation. The set of processes reachable from P while respecting a size bound s is the set $\mathrm{reach}^s_\Delta(P) := \{Q \mid P \cdots Q \in \mathrm{traces}_\Delta(P) \cap \mathbb{S}^*_s\}$.

▶ **Definition 13.** *For some $s, k \in \mathbb{N}$, we say the process P is (s,k)-*bounded *(w.r.t. a finite set Δ of definitions) if $\mathrm{reach}^s_\Delta(P) \subseteq \mathbb{D}_{s,k}$, i.e. from P only processes of depth at most k can be reached, in traces respecting the size bound s.*

▶ **Example 14.** Example 9 is $(3,7)$-bounded. We defer the proof of this fact to Section 3.6.

▶ **Example 15** (Encryption Oracle)**.** The definition $\mathsf{E}[k] := \mathbf{in}(x : x).(\langle \mathsf{e}(x)_k\rangle \parallel E[k])$ leads to unboundedness as soon as the initial process contains $\mathsf{E}[k]$ for some k not known to the intruder, and size bound such that x can match messages of size greater than 1. In such case, the intruder can inject messages (c_i, c_{i+1}) for unboundedly many i, where c_i are intruder-generated nonces. Since k is secret, the resulting reachable configurations would contain "encryption chains" of the form $\nu k.\nu c_1, \ldots, c_n.(\langle \mathsf{e}(c_1, c_2)_k\rangle \parallel \langle \mathsf{e}(c_2, c_3)_k\rangle \parallel \ldots \langle \mathsf{e}(c_{n-1}, c_n)_k\rangle)$. When such chains appear in a set for unboundedly many $n \in \mathbb{N}$, the set is not depth-bounded. This *encryption oracle* pattern could be considered an anti-pattern because it can be exploited for a chosen-plaintext attack on the key k. The pattern can be usually modified or constrained to obtain a bounded protocol. One option is to limit the verification to only consider traces where x is of size 1.[1]

The two bounds s and k are very different in nature. For size, we ignore any trace that involves messages exceeding size s. Then we determine if the depth bound k is respected by all remaining traces. Ignoring traces exceeding s is acceptable for protocols not susceptible to type confusion attacks and is achieved in other tools by using typing. Our method can however be pushed beyond this limitation: In Section 4.1, we show how the results of our analysis on the traces of bounded message size, can be generalised to results that hold for the unrestricted set of traces.

3 Ideal Completions for Security Protocols

Our main technical contributions are the proofs needed to show that (s,k)-bounded protocols form a post-effective ideal completion in the sense of [8]. First we outline the significance and applications of this result, and then proceed with the proofs.

[1] In Tamarin one would obtain this by typing x:*fresh*.

3.1 Downward-Closed Invariants and Security Properties

Suppose we want to establish that a protocol P fulfils some security requirement. In a typical proof, one needs to establish many intermediate facts about executions of the protocol. For example, part of the argument may hinge on some key k being always unknown to the intruder. This kind of property is an *invariant* of the protocol: it holds at every step of an execution. Formally, an invariant of P (under definitions Δ and size constraint s) is any set of processes that includes $\mathrm{reach}^s_\Delta(P)$. For example, k is never leaked to the intruder in executions of the protocol P if the set of processes $\mathcal{S}_k := \{Q \mid \langle k \rangle \not\sqsubseteq_{\mathsf{kn}} Q\}$ – i.e. all the processes where k is not public – is an invariant of P. We will focus here on the class of $\sqsubseteq_{\mathsf{kn}}$-downward-closed invariants. Formally, given a set of processes X, its $\sqsubseteq_{\mathsf{kn}}$-*downward closure* is the set $X{\downarrow} := \{Q \mid \exists P \in X \colon Q \sqsubseteq_{\mathsf{kn}} P\}$. A set X is $\sqsubseteq_{\mathsf{kn}}$-*downward closed* if $X = X{\downarrow}$. Many properties of interest are naturally downward closed. For example, the set $\mathcal{S}_k$ above is downward-closed as $\langle k \rangle \not\sqsubseteq_{\mathsf{kn}} Q$ and $Q' \sqsubseteq_{\mathsf{kn}} Q$ implies $\langle k \rangle \not\sqsubseteq_{\mathsf{kn}} Q'$.

The problem we need to solve is, then, how to show that a given downward-closed set X is an invariant for a given protocol. Formally, that corresponds to checking $X \supseteq \mathrm{reach}^s_\Delta(P)$ which, by downward-closure of X, is equivalent to checking $X \supseteq \mathrm{reach}^s_\Delta(P){\downarrow}$. To prove the latter inclusion, our strategy is to find an inductive invariant that includes the initial state P and that is included in X. Let $\mathrm{post}^s_\Delta(X) := \{Q' \mid \exists Q \in X, Q \to Q' \in \mathbb{S}_s\}$ be the set of processes reachable in one step from processes in X. An invariant X is *inductive* if $X \supseteq \mathrm{post}^s_\Delta(X)$, which is equivalent to requiring $X \supseteq \mathrm{post}^s_\Delta(X){\downarrow}$ if X is downward-closed. Any inductive invariant that contains the initial process P will include $\mathrm{reach}^s_\Delta(P)$.
To turn this proof strategy into an algorithm, we need three components:

1. a recursively enumerable finite representation of downward-closed sets,
2. a way to decide inclusion between two downward-closed sets, given their representation,
3. an algorithm (called $\widehat{\mathrm{post}}^s_\Delta$) to compute, given a finite representation of a downward-closed set D, a finite representation of $\mathrm{post}^s_\Delta(D){\downarrow}$.

Unfortunately, downward-closed sets cannot be finitely represented in general, especially if one considers unbounded sessions/nonces. We will show, however, that we can devise solutions to all three items above for downward-closed subsets of $\mathbb{D}_{s,k}$, under a mild restriction on the intruder model. Solving problems 1 to 3 amounts to proving that $\mathbb{D}_{s,k}$ admits a *post-effective ideal completion* in the sense of [22, 8]. This implies that we can decide if the reachable configurations satisfy any given downward-closed property, by adapting the enumeration scheme presented in [8].

▶ **Theorem 16.** *Given a property $\mathcal{D} \subseteq \mathbb{P}$, and a protocol P with definitions Δ, we write $P, \Delta \models^s \mathcal{D}$ if $\mathrm{reach}^s_\Delta(P) \subseteq \mathcal{D}$. If $\mathcal{D} \subseteq \mathbb{D}_{s,k}$ is downward-closed, then $P, \Delta \models^s \mathcal{D}$ is decidable.*

Proof. The algorithm runs two semi-procedures, Prover and Refuter, in parallel. The first procedure, Prover, enumerates all the downward-closed subsets I of $\mathbb{D}_{s,k}$. For each I, Prover checks if

(a) $P \in I$,
(b) I is inductive, by checking $\widehat{\mathrm{post}}^s_\Delta(I) \subseteq I$, and
(c) $I \subseteq \mathcal{D}$.

If we find such a set I, then we have proven $\mathrm{reach}_\Delta(P) \subseteq \mathcal{D}$, and the overall algorithm can terminate returning "True". The second procedure, Refuter, enumerates all $Q \in \mathrm{reach}_\Delta(P)$ and checks if $Q \notin \mathcal{D}$, in which case the overall algorithm can terminate returning "False".

When $P, \Delta \models^s \mathcal{D}$ holds, then, in the worst case, Prover will eventually consider the finite representation of $\mathrm{reach}^s_\Delta(P){\downarrow}$, which satisfies checks (a) to (c) above. In the case where $P, \Delta \models^s \mathcal{D}$ does not hold, Refuter would eventually find a reachable process not in $\mathcal{D}$. In either case, the algorithm terminates with the correct answer. ◀

The above algorithm can be used to decide the following properties.

Deciding (s, k)-boundedness. The set $\mathbb{D}_{s,k}$ is itself downward-closed, so we can decide if P is (s, k)-bounded, by deciding $P, \Delta \models^s \mathbb{D}_{s,k}$.

Deciding control-state reachability and secrecy. Control-state reachability asks whether there is an execution of the protocol which reaches a process containing a process call $\mathsf{Q}[\ldots]$ for some given Q. Secrecy can be reduced to control-state reachability by introducing a definition $\mathsf{Secret}[m] := \mathbf{in}(m).\mathsf{Leak}[m]$ (for a special process identifier Leak with no definition). In the definition of the protocol one can call $\mathsf{Secret}[m]$ to mark some message m as a secret, and secrecy corresponds to asking control-state (un)reachability for Leak.

If P is (s, k)-bounded, control-state reachability for Q from P, can decided by $P, \Delta \models^s \mathcal{D}_{\mathsf{Q}}$, where $\mathcal{D}_{\mathsf{Q}}$ is the (downward-closed) subset of $\mathbb{D}_{s,k}$ of processes that do not contain calls to Q. Notice that, when P is arbitrary, the algorithm checks (s, k)-boundedness and control-state reachability at the same time.

Absence of misauthentication. A misauthentication happens when a principal a believes she shares a secret n with b but b believes she shares the secret n with some other entity c. To check this situation can never arise, we can produce the process $\mathsf{Auth}[a, b, n]$ when a believes to share the secret n with b. Absence of misauthentication can be decided by $P, \Delta \models^s \mathcal{A}$ where $\mathcal{A} = \{\, Q \in \mathbb{D}_{s,k} \mid \nu a, b, c, n.(\mathsf{Auth}[a, b, n] \parallel \mathsf{Auth}[b, c, n]) \not\sqsubseteq_{\mathsf{kn}} Q \,\}$.

Susceptibility to known-plaintext attacks. The task of guessing a symmetric key is made much easier if it is possible for the attacker to have access to an arbitrarily large number of known nonces encrypted with the same key k. We can model this situation by asking if:

$$\forall n \in \mathbb{N} \colon \exists Q \in \mathrm{reach}^s_\Delta(P) \colon R^n \sqsubseteq_{\mathsf{kn}} Q \qquad \text{where } R = \nu m.(\langle m \rangle \parallel \langle \mathsf{e}(m)_k \rangle) \tag{$\dagger$}$$

If ($\dagger$) holds for P then the intruder does have access to an unbounded supply of known messages m encrypted with the same key k. Interestingly, the property becomes meaningful only when considering unbounded number of nonces. Condition ($\dagger$) is equivalent to $\mathrm{reach}^s_\Delta(P)\!\downarrow \supseteq \{R^n \mid n \in \mathbb{N}\}\!\downarrow$. If we find a downward-closed inductive invariant I for P, such that $I \not\supseteq \{R^n \mid n \in \mathbb{N}\}\!\downarrow$, then we can be sure that ($\dagger$) does not hold, and P is not susceptible to known-plaintext attacks on k. We can therefore semi-decide ($\dagger$) by enumerating all candidate I. Contrary to the previous algorithms, we are not able to provide a Refuter procedure (we conjecture the problem is undecidable). We can get a decision procedure if, instead of unboundedly many plaintext-encrypted pairs we ask whether a sufficiently high, user-provided number N of such pairs can be produced. The problem can be extended to cover the case where the known-plaintext can be any message of size at most s.

Notice how, if the protocol is found to satisfy the property, the algorithms above can output an inductive invariant acting as an independently checkable certificate of correctness. Although the mentioned security properties alone do not cover the full security requirements, an effectively presented invariant can provide the foundations to prove further properties.

The algorithm of Theorem 16 relies on expensive enumeration schemes, which are mainly a theoretical device to prove decidability. In a more practical setting, the candidate invariants can be supplied by the user and refined interactively, avoiding the need for the enumeration of Prover, or they can be inferred as we describe in Section 3.6.

3.2 Bounded Processes are Well-Quasi-Ordered

We construct finite representations of downward-closed invariants by making use of the algebraic structure of the quasi-order $(\mathbb{D}_{s,k}, \sqsubseteq_{\mathsf{kn}})$. A relation $\sqsubseteq \subseteq S \times S$ over some set S is a *quasi-order* (qo) if it is reflexive and transitive. An infinite sequence $s_0, s_1, \ldots$ of elements of S is called *good* if there are two indexes $i < j$ such that $s_i \sqsubseteq s_j$. A qo $(S, \sqsubseteq)$ is called a *well quasi order* (wqo) if all its sequences are good.

We prove $(\mathbb{D}_{s,k}, \sqsubseteq_{\mathsf{kn}})$ is a wqo for any intruder model, by showing a correspondence between processes in $\mathbb{D}_{s,k}$ and finitely-labelled forests of height at most k, which we represent as nested multisets. The details can be found in the full version of the paper [19].

3.3 Limits and Ideal Decompositions

By exploiting the wqo structure of $\mathbb{D}_{s,k}$, we can provide a finite representation for its downward-closed sets. Let $(S, \sqsubseteq)$ be a qo. A set $D \subseteq S$ is an *ideal* if it is downward-closed and directed, i.e. for all $x, y \in D$ there is a $z \in D$ such that $x \sqsubseteq z$ and $y \sqsubseteq z$. We write $\mathrm{Idl}(S)$ for the set of ideals of S. It is well-known that in a well-quasi-order, every downward-closed set is equal to a canonical minimal finite union of ideals, its *ideal decomposition*. To represent downward-closed sets of $\mathbb{D}_{s,k}$ we will only need to provide finite representations of its ideals. We represent ideals using *limits*, which have the same syntax as processes augmented with a construct $-^{\omega}$ to represent an arbitrary number of parallel components.

▶ **Definition 17** (Limits). *We call* limits *the terms L formed according to the grammar:*

$$\mathbb{L} \ni L ::= \mathbf{0} \mid (R_1 \parallel \cdots \parallel R_n) \qquad R ::= B \mid B^{\omega} \qquad B ::= \langle M \rangle \mid \mathsf{Q}[\vec{M}] \mid \nu x.L$$

▶ **Definition 18** (Denotation of limits). *The denotation of L is the set $[\![L]\!] := [\,L\,]{\downarrow}$ where:*

$$[\,\mathbf{0}\,] := \{\mathbf{0}\} \qquad [\,L_1 \parallel L_2\,] := \{(P_1 \parallel P_2) \mid P_1 \in [\,L_1\,], P_2 \in [\,L_2\,]\}$$
$$[\,\mathsf{Q}[\vec{M}]\,] := \{\mathsf{Q}[\vec{M}]\} \qquad [\,B^{\omega}\,] := \textstyle\bigcup_{n\in\mathbb{N}} \{(P_1 \parallel \cdots \parallel P_n) \mid \forall i \le n : P_i \in [\,B\,]\}$$
$$[\,\langle M \rangle\,] := \{\langle M \rangle\} \qquad [\,\nu x.L\,] := \{\nu x.P \mid P \in [\,L\,]\}$$

We call the processes in $[\![L]\!]$ instances *of L. Define $\mathrm{nest}_{\nu}(L)$ to be as nest_{ν} on processes with the addition of the case $\mathrm{nest}_{\nu}(L^{\omega}) := \mathrm{nest}_{\nu}(L)$. It is easy to check that for each $P \in [\![L]\!]$, $\mathrm{depth}(P) \le \mathrm{nest}_{\nu}(L)$. We write $\mathbb{L}^{X}_{s,k}$ for the set of limit expressions L with free names in X that have $\mathrm{nest}_{\nu}(L) \le k$ and do not contain messages of size exceeding s. We often omit X and understand it is a fixed finite set of names.*

▶ **Theorem 19.** *Limits faithfully represent ideals: $I \in \mathrm{Idl}(\mathbb{D}_{s,k}) \iff \exists L \in \mathbb{L}_{s,k} \colon I = [\![L]\!]$.*

3.4 Decidability of Inclusion

Now we turn to decidability of inclusion between downward-closed sets. It is well-known that in a wqo, given the ideal decomposition of two downward-closed sets $D_1 = I_1 \cup \ldots \cup I_n$ and $D_2 = J_1 \cup \ldots \cup J_m$, we have $D_1 \subseteq D_2$ if and only if for all $1 \le i \le n$, there is a $1 \le j \le m$, such that $I_i \subseteq J_j$. Hence, decidability of ideals inclusion implies decidability of downward-closed sets inclusion.

We extend structural congruence to limits in the obvious way, with the addition of the law $\langle M \rangle^{\omega} \equiv \langle M \rangle$ obtaining that $L \equiv L'$ implies $[\![L]\!] = [\![L']\!]$. We can define a standard form for limits: every limit is structurally congruent to a limit of the form $\nu\vec{x}.(\langle \Gamma \rangle \parallel \prod_{i\in I} \mathsf{Q}_i[\vec{M}_i] \parallel \prod_{j\in J} B_j^{\omega})$ where every name in $\vec{x}$ occurs free at least once in the scope of the restriction, and

$$\lceil L\rceil^n := \begin{cases} L & \text{if } L \text{ is sequential or } \mathbf{0} \\ \lceil L_1\rceil^n \parallel \lceil L_2\rceil^n & \text{if } L = L_1 \parallel L_2 \\ \nu x.(\lceil L'\rceil^n) & \text{if } L = \nu x.L' \\ (\lceil B\rceil^n)^n & \text{if } L = B^\omega \end{cases} \qquad L \otimes n := \begin{cases} L & \text{if } L \text{ is sequential or } \mathbf{0} \\ L_1 \otimes n \parallel L_2 \otimes n & \text{if } L = L_1 \parallel L_2 \\ \nu x.(L' \otimes n) & \text{if } L = \nu x.L' \\ (B \otimes n)^n \parallel B^\omega & \text{if } L = B^\omega \end{cases}$$

Figure 4 The grounding $\lceil\text{-}\rceil^n : \mathbb{L}_{s,k} \to \mathbb{D}_{s,k}$ and extension $\text{-}\otimes n : \mathbb{L}_{s,k} \to \mathbb{L}_{s,k}$ operations on limits.

for all $j \in J$, B_j is also in standard form. When we write $\mathrm{sf}(L) \overset{\alpha}{=} \nu\vec{x}.(\langle\Gamma\rangle \parallel Q \parallel R)$ we imply that Q is a parallel composition of process calls $\prod_{i\in I}\mathsf{Q}_i[\vec{M}_i]$ (in which case we write $|Q|$ for $|I|$) and R is a parallel composition of iterated limits $\prod_{j\in J}B_j^\omega$.

To better manipulate limits, we introduce, in Figure 4, the n-th grounding $\lceil L\rceil^n$ and the n-th extension $L \otimes n$, of a limit L. Grounding replaces each -^ω with -^n, with the obvious property that $\lceil L\rceil^n \in [\![L]\!]$. An extension $L \otimes n$ produces a new limit with each sub-limit B^ω unfolded n times. Note that extension does not alter semantics: $[\![L]\!] = [\![L \otimes n]\!]$.

The absorption axiom. The decidability proof hinges on a characterisation of inclusion that requires an additional hypothesis on the intruder model.

▶ **Definition 20** (Absorbing intruder). *Fix an intruder model $\mathbb{I} = (\Sigma, \vdash)$. Let $\vec{x}$ and $\vec{y}$ be two lists of pairwise distinct names, Γ be a finite set of messages, and $\Gamma' = \Gamma[\vec{y}/\vec{x}]$. Moreover, assume that* $\mathrm{names}(\Gamma) \cap \vec{y} = \emptyset$. *We say $\mathbb{I}$ is* absorbing *if, for all messages M with* $\mathrm{names}(M) \subseteq \mathrm{names}(\Gamma)$, *we have that $\Gamma, \Gamma' \vdash M$ if and only if $\Gamma \vdash M$.*

▶ **Lemma 21.** $\mathbb{I}_{\mathsf{en}}$ *is absorbing.*

For the rest of the paper, we assume an absorbing intruder model. The absorption axiom has a technical definition, which becomes more intuitive if understood in the context of limits of the form $L = \left(\nu\vec{x}.(\langle\Gamma\rangle \parallel Q)\right)^\omega$. Imagine comparing the difference in knowledge between $\lceil L\rceil^1$ and $\lceil L\rceil^2$: we have $\mathrm{sf}(\lceil L\rceil^2) \overset{\alpha}{=} \left(\nu\vec{x}.\nu\vec{x}'.(\langle\Gamma\rangle \parallel \langle\Gamma'\rangle \parallel Q \parallel Q')\right)$, where $\Gamma' = \Gamma[\vec{x}'/\vec{x}]$. The absorption axiom tells us that if we want to check whether a process $\nu\vec{x}.\langle M\rangle$ is embedded in $\mathrm{sf}(\lceil L\rceil^2)$, we only need to check if M is derivable from Γ and we can ignore Γ'. In other words, we only need to check if $\nu\vec{x}.\langle M\rangle$ is embedded in $\lceil L\rceil^1$.

We are now ready to prove our main result: a small model property that shows decidability of limit inclusion. Let us present the intuition on the simpler problem of deciding inclusion when one of the limits is a single process P, i.e. deciding if $P \in [\![L]\!]$. Take $P = \nu a, b.(\langle \mathsf{e}(a)_b\rangle \parallel \mathsf{A}[b])$ and $L = \left(\nu x.(\langle x\rangle \parallel A[x])\right)^\omega$. Suppose we replicate the ω twice, obtaining the equivalent limit $L \otimes 2 \equiv \nu x_0, x_1.(\langle x_0\rangle \parallel A[x_0] \parallel \langle x_1\rangle \parallel A[x_1] \parallel L)$. The idea is that we can match P against the fixed part of $L \otimes 2$:

$$\begin{aligned} P \equiv_{\mathsf{kn}} \nu x_0, x_1.(\langle \mathsf{e}(x_0)_{x_1}\rangle \parallel \mathsf{A}[x_1]) &\sqsubseteq_{\mathsf{kn}} \nu x_0, x_1.(\langle \mathsf{e}(x_0)_{x_1}\rangle \parallel A[x_0] \parallel A[x_1]) \\ &\sqsubseteq_{\mathsf{kn}} \nu x_0, x_1.(\langle x_0\rangle \parallel \langle x_1\rangle \parallel A[x_0] \parallel A[x_1]) \in L \otimes 2 \end{aligned}$$

The first observation is therefore that if we can find some m such that P is embedded in the fixed part of $L \otimes m$, we have proven $P \in [\![L]\!]$. To turn this into an algorithm, we need to prove that there exists an n so that if we failed to embed P in the fixed part of $L \otimes m$, for any $m \leq n$ then P is not going to embed in $L \otimes m'$ for *every* m', and therefore $P \notin [\![L]\!]$. In other words, we need to know that after some threshold n, there is no point trying with bigger extensions. Take for example $P = \nu x, y.(\mathsf{B}[x, y]||B[y, x])$ and $L = (\nu x, y.\mathsf{B}[x, y])^\omega$. We can try and embed P into $L \otimes 2$ but we would fail as the fixed part expands to $\nu x_0, y_0.\mathsf{B}[x_0, y_0] \parallel \nu x_1, y_1.\mathsf{B}[x_1, y_1]$. It is easy to see that expanding further would not introduce new patterns in the fixed part of the limit which would help embed P.

Theorem 22 formalises the idea for general inclusion between two arbitrary limits L_1 and L_2: it proves that the threshold for expansion is the number of fixed restrictions of L_1 plus the number of fixed process calls of L_1, plus one; and it makes use of the absorption axiom to prove the threshold is sound even in the presence of knowledge.

▶ **Theorem 22** (Characterisation of Limits Inclusion). *Let L_1 and L_2 be two limits, with $\mathrm{sf}(L_1) \overset{\alpha}{=} \nu\vec{x}_1.(\langle\Gamma_1\rangle \parallel Q_1 \parallel \prod_{i\in I} B_i^\omega)$, and let $n = |\vec{x}_1| + |Q_1| + 1$. Then:*

$$[\![L_1]\!] \subseteq [\![L_2]\!] \iff \begin{cases} \mathrm{sf}(L_2 \otimes n) \overset{\alpha}{=} \nu\vec{x}_1, \vec{x}_2.(\langle\Gamma_2\rangle \parallel Q_1 \parallel Q_2 \parallel R_2) \;\; \textit{and} \;\; \Gamma_1 \leq_{\mathsf{kn}} \Gamma_2 & \textbf{(A)} \\ [\![\langle\Gamma_1\rangle \parallel \prod_{i\in I} B_i]\!] \subseteq [\![\langle\Gamma_2\rangle \parallel R_2]\!] & \textbf{(B)} \end{cases}$$

▶ **Theorem 23.** *Given $L_1, L_2 \in \mathbb{L}$ it is decidable whether $[\![L_1]\!] \subseteq [\![L_2]\!]$.*

Proof. Theorem 22 leads to a recursive algorithm. Given L_1 and L_2, one computes $\mathrm{sf}(L_1)$ and $\mathrm{sf}(L_2 \otimes n)$. For every α-renaming that makes condition **(A)** hold, one checks condition **(B)** (recursively). If no renaming makes both true then the inclusion does not hold. In the recursive case, there are fewer occurrences of ω in the limit on the left, eventually leading to the case where L_1 has no occurrence of ω, and only condition **(A)** needs to be checked. ◀

▶ **Example 24** (Limit inclusion). Consider the following two limits:

$$L_1 = \nu x_1.\big((\nu x_2.(\langle \mathsf{e}(x_2)_{x_1}\rangle \parallel \mathsf{A}[x_2] \parallel \mathsf{A}[x_1]))^\omega\big)$$
$$L_2 = \nu y_1, y_3.\big(\langle y_3\rangle \parallel \mathsf{A}[y_3]^\omega \parallel (\nu y_2.(\langle y_2\rangle \parallel \mathsf{A}[y_2]))^\omega\big)$$

We prove that $[\![L_1]\!] \subseteq [\![L_2]\!]$ by applying the recursive algorithm from Theorem 23. By Theorem 22, here the threshold for expansion is $n = 2$, but we will try with $n = 1$. In case we succeed, the inclusion holds. If not, we might have to increase n up to 2. This results in

$$\begin{aligned} L_2 \otimes 1 &= \nu y_1, y_3.\big(\langle y_3\rangle \parallel \mathsf{A}[y_3] \parallel \mathsf{A}[y_3]^\omega \parallel (\nu y_2.(\langle y_2\rangle \parallel \mathsf{A}[y_2]))^\omega \parallel (\nu y_2.(\langle y_2\rangle \parallel \mathsf{A}[y_2]))\big) \\ &\equiv_{\mathsf{kn}} \nu y_1, y_{22}, y_3.\big(\langle y_3\rangle \parallel \langle y_{22}\rangle \parallel \mathsf{A}[y_{22}] \parallel \mathsf{A}[y_3] \parallel \mathsf{A}[y_3]^\omega \parallel (\nu y_2.(\langle y_2\rangle \parallel \mathsf{A}[y_2]))^\omega\big) \end{aligned}$$

To try and match the fixed part of L_1 with $L_2 \otimes 1$, we could α-rename x_1 to y_1. This works for **(A)** but the remaining goal **(B)** cannot be shown as it does not hold because we cannot derive $\langle \mathsf{e}(x_2)_{y_1}\rangle$ from the knowledge on the right-hand side:

$$[\![\nu x_2.(\langle \mathsf{e}(x_2)_{y_1}\rangle \parallel \mathsf{A}[x_2] \parallel \mathsf{A}[x_1])]\!] \not\subseteq [\![\langle y_3\rangle \parallel \langle \mathsf{e}(y_{22})_{y_3}\rangle \parallel \mathsf{A}[x_1]^\omega \parallel (\nu y_2.(\langle \mathsf{e}(y_2)_{y_3}\rangle \parallel \mathsf{A}[y_2]))^\omega]\!]$$

Choosing the α-renaming $[x_1/y_3]$ leaves us instead with:

$$[\![\nu x_2.(\langle \mathsf{e}(x_2)_{y_3}\rangle \parallel \mathsf{A}[x_2] \parallel \mathsf{A}[x_1])]\!] \subseteq [\![\langle y_3\rangle \parallel \langle \mathsf{e}(y_{22})_{y_3}\rangle \parallel \mathsf{A}[x_1]^\omega \parallel (\nu y_2.(\langle \mathsf{e}(y_2)_{y_3}\rangle \parallel \mathsf{A}[y_2]))^\omega]\!]$$

which holds. It suffices to expand the latter limit by 1 and to choose the renaming $[x_2/y_2]$. Then, $\mathsf{e}(x_1)_{x_2} \leq_{\mathsf{kn}} x_1, x_2$ holds and the process calls match. Note that it is crucial to keep $\mathsf{A}[x_1]^\omega$ on the right side.

3.5 Computing Post-Hat

The last result we need is the decidability of a function $\widehat{\mathrm{post}}^s_\Delta(L)$ which, given a limit L, returns a finite set of limits $\{L_1, \ldots, L_n\}$ such that $\mathrm{post}^s_\Delta([\![L]\!])\!\downarrow = [\![L_1]\!] \cup \ldots \cup [\![L_n]\!]$. The challenge is representing all the possible successors of processes in $[\![L]\!]$ without having to

$$L = \nu a, b, k_{as}, k_{bs}.(\langle a, b\rangle \parallel \mathsf{A}_1[a, b, k_{as}]^\omega \parallel \mathsf{B}_1[a, b, k_{bs}]^\omega \parallel \mathsf{S}[a, b, k_{as}, k_{bs}]^\omega \parallel L_1^\omega)$$
$$L_1 = \nu n_a.\big(\langle n_a\rangle \parallel \mathsf{A}_2[a, b, k_{as}, n_a] \parallel L_2^\omega\big)$$
$$L_2 = \nu k.\big(\langle \mathsf{e}(k)_{(a,k_{as})}\rangle \parallel \langle \mathsf{e}(k)_{(b,k_{as})}\rangle \parallel \langle \mathsf{e}(k)_{(n_a,k_{as})}\rangle \parallel \langle \mathsf{e}(k)_{k_{bs}}\rangle \parallel \mathsf{Secret}[k]^\omega \parallel \mathsf{A}_3[a, b, k_{as}, k]^\omega \parallel L_3^\omega\big)$$
$$L_3 = \nu n_b.\big(\langle \mathsf{e}(n_b)_{(k,k)}\rangle \parallel \langle \mathsf{e}(n_b)_k\rangle \parallel \mathsf{B}_2[a, b, k_{bs}, n_b, k]\big)$$

Figure 5 An inductive invariant for Example 9.

enumerate $[\![L]\!]$. The key idea hinges again on the absorption axiom: we observe that to consider all possible process calls that may cause a transition, it is enough to unfold each ω in L by some bounded number b. Any transition taken from further unfoldings will give rise to successors that are congruent to some of the ones already considered.

The bound b used in extending a limit, is defined as a function of the process definitions and the intruder model. The arity of a pattern $\vec{x} : M$ is $|\vec{x}|$. Given a set of definitions Δ, $\beta(\Delta)$ is the maximum arity of patterns in Δ. The function $\gamma(\mathbb{I})$ returns the maximum arity of the constructors in the signature of $\mathbb{I}$.

▶ **Definition 25** ($\widehat{\text{post}}$). *Let* $b = \beta(\Delta) \cdot \gamma(\mathbb{I})^{s-1} + 1$ *and* $\text{sf}(L \otimes b) = \nu\vec{x}.(\langle\Gamma\rangle \parallel Q \parallel R)$,

$$\widehat{\text{post}}^s_\Delta(L) := \{\ \nu\vec{y}.\big(\langle\Gamma'\rangle \parallel Q' \parallel R\big)\ \mid\ \nu\vec{x}.\big(\langle\Gamma\rangle \parallel Q\big) \to_\Delta \nu\vec{y}.\big(\langle\Gamma'\rangle \parallel Q'\big) \in \mathbb{S}_s\ \}.$$

▶ **Theorem 26.** $\widehat{\text{post}}^s_\Delta(L) = \{L_1, \ldots, L_n\} \implies \text{post}^s_\Delta([\![L]\!])\downarrow = [\![L_1]\!] \cup \ldots \cup [\![L_n]\!]$.

3.6 Algorithmic Aspects

The limit L in Figure 5 represents an inductive invariant for Example 9, under size bound 3. It can be proven invariant by checking that $\widehat{\text{post}}^3_\Delta(L)$ is included in L. Since the initial process of Example 9, P_0, is in $[\![L]\!]$, the invariant certificates that any reachable process is $(3, 7)$-bounded (note $\text{nest}_\nu(L) = 7$) and satisfies secrecy. In fact, L^ω is also inductive, proving boundedness and secrecy for any process in $(P_0)^\omega$. Since $[\![\nu k.(\nu x.(\langle x, \mathsf{e}(x)_k\rangle))^\omega]\!] \not\subseteq [\![L^\omega]\!]$, L^ω provides proof that the protocol is not susceptible to known-plaintext attacks where arbitrary known nonces are available to the intruder encrypted with the same key.[2] The algorithms for inclusion and $\widehat{\text{post}}$ can be used to check inductiveness given a candidate invariant such as L. This leaves open the question of how to efficiently generate candidates. The algorithm of Theorem 16 can in principle be used to enumerate all candidate invariants, with an impractically high complexity. Luckily, a more directed inference of invariants can be obtained by a *widening operator*, in the style of [37]; in fact, the invariant L was automatically inferred from P_0 using the widening of our prototype tool. The basic observation behind invariant inference, is that from a sequence of transitions $P_1 \to^* P_2$ with $P_1 \sqsubseteq_{\mathsf{kn}} P_2$, one can deduce that the same sequence can be simulated from P_2 (by Theorem 8) obtaining a P_3 with $P_2 \sqsubseteq_{\mathsf{kn}} P_3$ and so on. The embedding between P_1 and P_2, is justified by $P_1 \equiv_{\mathsf{kn}} \nu\vec{x}.(\langle\Gamma_1\rangle \parallel Q)$, $P_2 \equiv_{\mathsf{kn}} \nu\vec{x}.(\langle\Gamma_1\rangle \parallel Q \parallel P)$; we can extrapolate the difference P and accelerate the sequence of transitions by constructing the limit $\nu\vec{x}.(\langle\Gamma_1\rangle \parallel Q \parallel P^\omega)$. This operation can be extended to limits. The end product is a finite union of limits which is inductive by construction. This procedure requires exploration of transitions and many inclusion checks, a costly combination. To obtain a more practical algorithm, we devised two techniques: inductiveness checks through "incorporation", and a coarser widening.

[2] The property could be extended to cover composite known-plaintext messages (of some maximum size s), and the generation of a sufficiently high number N of plaintext-encrypted pair with the same key.

Table 1 Experimental results. Columns: **Infer**ence of invariant fully automatic (F) or interactive (I); **C**heck of inductiveness; **S**ecrecy proved (✓), not holding (×), not modelled (○).

Name	Infer	C	S	Name	Infer	C	S	Name	Infer	C	S
Ex.9	4.4s F	1.8s	✓	NHS	5.0s F	1.6s	○	YAH	7.8s F	2.5s	○
OR	3.4s F	1.9s	○	NHSs	6.8s F	1.7s	✓	YAHs1	12.3s F	2.6s	✓
ORl	25.0s F	3.5s	×	NHSr	90.9s I	20.8s	×	YAHs2	8.8s F	2.0s	✓
ORa	13.8s I	5.0s	○	KSL	37.0s F	9.8s	✓	YAHlk	11.9s I	17.3s	✓
ORs	9.8s F	2.0s	✓	KSLr	200.6s F	31.4s	✓	ARPC	0.5s F	0.1s	✓

Consider the inductiveness check $[\![\widehat{\text{post}}^s_\Delta(L)]\!] \subseteq [\![L]\!]$ implemented by checking that, for each transition considered by $\widehat{\text{post}}$ the resulting limit L' is included by L. We observe that L' and L will share the context of the transition: L can be rewritten to $C[\mathsf{Q}[\vec{M}]]$ and L' to $C[P]$ for some P. To prove inclusion of $C[P]$ in $C[\mathsf{Q}[\vec{M}]]$ it is then sufficient to show that P is embedded in $C[\mathsf{Q}[\vec{M}]]$. We call this check an *incorporation* of P in C. See Appendix D for an example. Although incomplete in general, incorporation can prove inductiveness in many practical examples, in a remarkably faster way.

There are cases, however, where the incorporation check fails on inductive invariants. Consider for example an inductive invariant represented by the union of two incomparable limits L_1, L_2. Suppose that for some $P \in [\![L_1]\!]$ there is P' with $P \to P' \in [\![L_2]\!] \setminus [\![L_1]\!]$. Then, incorporation of P' in L_1 would fail. To side-step this problem we replace union of limits with parallel composition: $[\![L_1]\!] \cup [\![L_2]\!] \subseteq [\![L_1 \parallel L_2]\!]$ by downward closure of $[\![\text{-}]\!]$. Using this over-approximation, we can try to aim for an inductive invariant which consists of exactly one limit, and for which the incorporation check suffices. This approximation is incomplete: some protocols require inductive invariants consisting of unions of incomparable limits.

4 Evaluation

We built a proof-of-concept tool implementing limit inclusion, inductivity check, incorporation and a coarse widening. Currently, the tool only supports symmetric encryption, but we plan to add support for asymmetric encryption, signatures and hashing. The source code and all the test protocol models are available at [20] where we also provide a tutorial-style explanation of the methodology. We summarise our experiments[3] in Table 1. The tool is instructed to compute, using the widening, an invariant for the provided model, under given message size assumptions. When an invariant can be found, it represents a proof that the model is depth-bounded. If the inferred invariant is leak-free, secrecy is also proven. We model a number of well-known protocols under various threat scenarios (e.g. with or without leak of old keys): Needham-Schröder (NHS), Otway-Rees (OR), Kehne-Schönwälder-Landendörfer (KSL), Yahalom (YAH) and Andrew RPC (ARPC). For any example containing a problematic encryption oracle pattern (see Example 15) we constrain the input message of the oracle to be a nonce (message of size 1). With the exception of NHSr, ORa and YAHlk, all the invariants were obtained fully automatically. For YAHlk we had to combine two widened limits (the timing is the sum of the time spent computing each limit); for NHSr and ORa we had to tweak a partially widened limit manually to make it inductive. To simulate using invariants as correctness certificates, for each example we re-checked them for inductiveness.

[3] Tests run with Python 2.7, z3-solver v4.8, 8GB RAM, Intel CPU i5, on Linux.

4.1 Limitations

Message size bounds. Bounding message size in the analysis is not always acceptable, as it may miss type confusion attacks. Such patterns arise, for example, in XOR-based protocols, where considering only typed runs is not appropriate. This is problematic because in most practical examples bigger size bounds lead to unboundedness in depth, as illustrated by Example 15. However, not all is lost: our limits can be extended to provide invariants with unbounded message size. The idea, detailed in Appendix B, is to first analyse a protocol with a size bound that ensures depth-boundedness, obtaining an inductive invariant as one or more limits; then we annotate such limits with "wildcards" ($\star$) on occurrences of names. A wildcarded name $a^\star$ stands for an arbitrary message (of arbitrary size). By introducing the appropriate wildcards, one can obtain an inductive invariant for the protocol with no message size bounds. For Example 15, $\mathsf{E}[k] \parallel \left(\nu x.\langle \mathsf{e}(x)_k \rangle\right)^\omega$ is an inductive invariant if we restrict x to be of size 1. Since x is never inspected , injecting larger terms in it would not lead to new behaviour. We can therefore generalise the limit to $\mathsf{E}[k] \parallel \left(\nu x.\langle \mathsf{e}(x^\star)_k \rangle\right)^\omega$ which is an inductive invariant for the protocol with no size bounds.

Since verification with no size bounds is undecidable, this extension is by necessity incomplete: there are downward-closed sets that cannot be represented by extended limits. However, since protocols typically achieve resistance to type confusion attacks by making sure that messages of the wrong type are discarded by honest principals, this extension is very effective. In fact, all the size bounds in our benchmarks can be lifted using this technique.

Inherent unboundedness. There are two ways a protocol may fall outside of our class. The first is when unboundedly many participants in a session form a ring/list topology, like the recursive protocol of [30, §6]. One can provide a partial solution by using an under-approximate model with a ring/list of fixed size, or an over-approximate model by using an unbounded star topology. The second way is when the intruder can produce irreducible encryption chains and the participants would inspect them generating new behaviour. In such cases not even extended limits can help. We deem this situation unlikely to be desirable in a realistic protocol.

5 Conclusions and Future Work

We presented a theory of decidable inductive invariants for depth-bounded cryptographic protocols. We showed how one can infer inductive invariants and evaluated the approach through a prototype implementation.

From a theoretical perspective, it would be interesting to determine precise complexity bounds for inclusion, for general intruder models. We can show that $\sqsubseteq_{\mathsf{kn}}$ is NP-complete for any intruder model that has polynomially decidable $\leq_{\mathsf{kn}}$ [33].

A direction for further improvement is extending the class of supported properties. In particular, we plan to study how invariants can be used to automatically prove diff-equivalence [5] without bounding sessions/nonces.

Finally, we intend to explore ways in which our invariants can be integrated in existing tools such as ProVerif and Tamarin. For instance, Tamarin performs a backwards search to find possible attacks. Our invariants could provide a pruning technique to avoid exploring paths that are unreachable from the initial state. Similar combinations of forward and backward search have been shown to improve performance dramatically for analyses of infinite-state systems such as Petri nets [7].

References

1 Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under (many more) equational theories. In *CSFW*, pages 62–76. IEEE Computer Society, 2005.

2 Myrto Arapinis and Marie Duflot. Bounding messages for free in security protocols - extension to various security properties. *Inf. Comput.*, 239:182–215, 2014.

3 Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. In *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005.

4 Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Foundations and Trends in Privacy and Security*, 1(1-2):1–135, 2016.

5 Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *J. Log. Algebr. Program.*, 75(1):3–51, 2008. doi:10.1016/j.jlap.2007.06.002.

6 Bruno Blanchet and Andreas Podelski. Verification of cryptographic protocols: tagging enforces termination. *Theor. Comput. Sci.*, 333(1-2):67–90, 2005.

7 Michael Blondin, Alain Finkel, Christoph Haase, and Serge Haddad. Approaching the coverability problem continuously. In *TACAS*, volume 9636 of *Lecture Notes in Computer Science*, pages 480–496. Springer, 2016.

8 Michael Blondin, Alain Finkel, and Pierre McKenzie. Handling infinitely branching well-structured transition systems. *Inf. Comput.*, 258:28–49, 2018.

9 Michael Burrows, Martín Abadi, and Roger M. Needham. A logic of authentication. In *SOSP'89*, pages 1–13. ACM, 1989.

10 Rohit Chadha, Vincent Cheval, Ştefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. *ACM Trans. Comput. Log.*, 17(4):23:1–23:32, 2016.

11 Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. DEEPSEC: deciding equivalence properties in security protocols theory and practice. In *IEEE Symposium on Security and Privacy*, pages 529–546. IEEE Computer Society, 2018.

12 Yannick Chevalier, Ralf Küsters, Michaël Rusinowitch, and Mathieu Turuani. An NP decision procedure for protocol insecurity with XOR. In *LICS*, pages 261–270. IEEE Computer Society, 2003.

13 Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. Typing messages for free in security protocols: The case of equivalence properties. In *CONCUR*, volume 8704 of *Lecture Notes in Computer Science*, pages 372–386. Springer, 2014.

14 Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. Decidability of trace equivalence for protocols with nonces. In *CSF'15*, pages 170–184. IEEE Computer Society, 2015.

15 Véronique Cortier, Niklas Grimm, Joseph Lallemand, and Matteo Maffei. A type system for privacy properties. In *ACM Conference on Computer and Communications Security*, pages 409–423. ACM, 2017.

16 Morten Dahl, Naoki Kobayashi, Yunde Sun, and Hans Hüttel. Type-based automated verification of authenticity in asymmetric cryptographic protocols. In *ATVA*, volume 6996 of *Lecture Notes in Computer Science*, pages 75–89. Springer, 2011.

17 Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Trans. Information Theory*, 29(2):198–207, 1983.

18 Emanuele D'Osualdo, Luke Ong, and Alwen Tiu. Deciding secrecy of security protocols for an unbounded number of sessions: The case of depth-bounded processes. In *CSF*, pages 464–480. IEEE Computer Society, 2017.

19 Emanuele D'Osualdo and Felix Stutz. Decidable inductive invariants for verification of cryptographic protocols with unbounded sessions. *CoRR*, abs/1911.05430, 2019. `arXiv:1911.05430`.

20 Emanuele D'Osualdo and Felix Stutz. Lemma9, version 1.0, 2020. `doi:10.5281/zenodo.3950846`.

21 Nancy A. Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. Undecidability of bounded security protocols. In *Workshop on Formal Methods and Security Protocols*, 1999.

22 Alain Finkel and Jean Goubault-Larrecq. Forward analysis for WSTS, part I: completions. In *STACS*, volume 3 of *LIPIcs*, pages 433–444, 2009.

23 Sibylle B. Fröschle. Leakiness is decidable for well-founded protocols. In *POST'15*, pages 176–195, 2015.

24 Reiner Hüchting, Rupak Majumdar, and Roland Meyer. Bounds on mobility. In *CONCUR*, volume 8704 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 2014.

25 Gavin Lowe. Some new attacks upon security protocols. In *Ninth IEEE Computer Security Foundations Workshop, March 10 - 12, 1996, Dromquinna Manor, Kenmare, County Kerry, Ireland*, pages 162–169. IEEE Computer Society, 1996.

26 Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin. The TAMARIN prover for the symbolic analysis of security protocols. In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013.

27 Roland Meyer. On boundedness in depth in the π-calculus. In Giorgio Ausiello, Juhani Karhumäki, Giancarlo Mauri, and C.-H. Luke Ong, editors, *IFIP TCS*, volume 273 of *IFIP*, pages 477–489. Springer, 2008.

28 Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Commun. ACM*, 21(12):993–999, 1978.

29 David J. Otway and Owen Rees. Efficient and timely mutual authentication. *Operating Systems Review*, 21(1):8–10, 1987.

30 Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998.

31 Mark D. Ryan and Ben Smyth. Applied pi-calculus. In *Formal Models and Techniques for Analyzing Security Protocols*, volume 5 of *Cryptology and Information Security Series*, pages 112–142. IOS Press, 2011.

32 Benedikt Schmidt, Simon Meier, Cas J. F. Cremers, and David A. Basin. Automated analysis of Diffie-Hellman protocols and advanced security properties. In *CSF*, pages 78–94. IEEE Computer Society, 2012.

33 Felix Stutz. Decidable inductive invariants for verification of cryptographic protocols with unbounded sessions. Master's thesis, Saarland University, 2019.

34 Alwen Tiu, Rajeev Goré, and Jeremy E. Dawson. A proof theoretic analysis of intruder theories. *Logical Methods in Computer Science*, 6(3), 2010.

35 Alwen Tiu, Nam Nguyen, and Ross Horne. SPEC: an equivalence checker for security protocols. In *APLAS*, volume 10017 of *Lecture Notes in Computer Science*, pages 87–95, 2016.

36 Thomas Wies, Damien Zufferey, and Thomas A. Henzinger. Forward analysis of depth-bounded processes. In *FoSSaCS*, volume 6014 of *Lecture Notes in Computer Science*, pages 94–108. Springer, 2010.

37 Damien Zufferey, Thomas Wies, and Thomas A. Henzinger. Ideal abstractions for well-structured transition systems. In *VMCAI'12'*, volume 7148 of *LNCS*, pages 445–460. Springer, 2012.

A Support for Other Cryptographic Primitives

We claim the assumptions we make on the intruder model are mild, and are satisfied by the symbolic models of many cryptographic primitives. We illustrated the treatment of (a)symmetric encryption; treatment of hashes and blind signatures is entirely analogous. By using the sequent calculus formalisation of [34] one can trivially extend our proofs to prove all these primitives form an effective absorbing intruder.

Supporting XOR requires a bit more analysis on the algebraic properties of the primitives. We take as reference the model of XOR analysed in [1, 12]. The two constructors $\oplus$ (of arity 2) and 0 (of arity 0) are added to the set of constructors. Their algebraic properties are formalised through a congruence relation:

$$M_1 \oplus (M_2 \oplus M_3) \cong (M_1 \oplus M_2) \oplus M_3 \qquad M_1 \oplus M_2 \cong M_2 \oplus M_1 \tag{1}$$

$$M \oplus 0 \cong M \qquad M \oplus M \cong 0 \tag{2}$$

The results of [1, 12] establish that the laws (2) can be always orientated from left to right. Formally, one can define the rewriting system $\rightsquigarrow$ with two rules $M \oplus 0 \rightsquigarrow M$ and $M \oplus M \rightsquigarrow 0$; and the congruence $\cong_{\mathsf{AC}}$ defined by laws (1). Then the relation $\rightsquigarrow_{\mathsf{AC}} := (\cong_{\mathsf{AC}} \circ \rightsquigarrow)$ is terminating, and confluent modulo $\cong_{\mathsf{AC}}$. The set of normal forms $M{\Downarrow} := \{N \mid M \rightsquigarrow^*_{\mathsf{AC}} N \not\rightsquigarrow_{\mathsf{AC}}\}$ is then guaranteed to be finite, computable, and such that $M \cong N \iff (M{\Downarrow} \cap N{\Downarrow}) \neq \emptyset$.

We can harmonise the equational theory of XOR with the deduction system of Figure 1 by adding the rules

$$\frac{}{\Gamma \vdash 0}\ \textsc{Xor}_0 \qquad \frac{\Gamma \vdash M_1 \qquad \Gamma \vdash M_2}{\Gamma \vdash M_1 \oplus M_2}\ \textsc{Xor}_{\mathrm{R}} \qquad \frac{\Gamma, M_1, M_2, M_1 \oplus M_2 \vdash N}{\Gamma, M_1, M_2 \vdash N}\ \textsc{Xor}_{\mathrm{L}}$$

$$\frac{\Gamma, M, M{\Downarrow} \vdash N}{\Gamma, M \vdash N}\ {\Downarrow}_{\mathrm{L}} \qquad \frac{\Gamma \vdash N \qquad N \in M{\Downarrow}}{\Gamma \vdash M}\ {\Downarrow}_{\mathrm{R}}$$

The deduction system accurately models XOR, even if it uses $\Downarrow$ instead of $\cong$: as proven in [12, Prop. 1], one can always restrict the intruder to manipulate messages in normal form without loosing expressive power.

We are left to prove that the derivability satisfies the effective absorbing intruder axioms. Decidability has been proven in [1, 12]. The axioms of Definitions 1 and 20 are easily satisfied by the same arguments we used for $\mathbb{I}_{\mathsf{en}}$. The proofs of (Relevancy) and absorption make use of the fact that if $N \in M{\Downarrow}$ then $\mathrm{fn}(N) \subseteq \mathrm{fn}(M)$.

We conjecture Diffie-Hellman exponentiation (following the model of e.g. [32]) can be shown to satisfy our axioms in the same way we treated XOR. The main issue with Diffie-Hellman, with respect to (Relevancy) and absorption, is the inverses law $M * M^{-1} \cong 1$: by using the law from right to left, one can involve arbitrary names in a derivation. This could be handled in the same way we handle cancellation of XOR, by normalising derivations so that the law is always applied left to right. We leave the formal development of this remark as future work.

A delicate point is the bounded message size assumption. With advanced primitives like XOR it is easier (but not inevitable) to encounter protocols for which it is impossible to extend the results on the bounded model to the unbounded case.

B Towards Unbounded Message Size

The analysis presented in Section 3 considers only traces (and attacks) involving messages of size smaller than some given bound s. We show here how the results of the analysis can be generalised to inductive invariants for the full set of traces, with no restriction on the size. Because of undecidability of the general problem, this generalisation will not be precise for every protocol: there are protocols for which the most precise generalised limit is trivial (i.e. does not ensure any non-trivial property of the protocol). For the protocols in our benchmarks however, one can get precise invariants by generalising the ones inferred by the tool.

We first introduce the syntax and semantics of generalised limits, and describe how we can use them to generalise our benchmarks. Finally, although studying how to automate this generalisation is beyond the scope of this paper, we briefly sketch how $\widehat{\mathsf{post}}$ can be adapted to work on generalised limits.

Aside: finer size bounds via typing. In our development, we assumed a global size bound s. To have finer control on the message sizes, as we do in our tool, one can introduce a primitive form of typing. Assuming wlog that all pattern variables are unique, a *typing* is a partial function $\mathsf{ty} \colon \mathcal{N} \rightharpoonup \mathbb{N}$, assigning to each pattern variable a maximum size for the messages it can match. A typing induces a typed transition relation $\to_{\Delta,\mathsf{ty}}$ which only matches patterns with subsitutions respecting ty; $\mathrm{reach}^{\mathsf{ty}}_{\Delta}(P)$ collects all the terms reachable from P through $\to_{\Delta,\mathsf{ty}}$. A typing ty of P, Δ is s-bounding if $\mathrm{reach}^{\mathsf{ty}}_{\Delta}(P) \subseteq \mathbb{S}_s$. One can check if ty is s-bounding on-the-fly while computing $\widehat{\mathsf{post}}$.

B.1 Generalised Limits

Recall the "encryption oracle" of Example 15: $E[k] = \mathbf{in}(x : x).(\langle \mathsf{e}(x)_k \rangle \parallel E[k])$. Without any restrictions on the pattern variable x, there is no way to prevent the encryption chains described in Example 15. However, we can use the 2-bounding typing $\mathsf{ty} = [x \mapsto 1]$, to obtain the limit $\mathsf{E}[k] \parallel \left(\nu m.\langle \mathsf{e}(m)_k \rangle\right)^{\omega}$ which is inductive (wrt $\to_{\Delta,\mathsf{ty}}$) and contains the initial state $\mathsf{E}[k]$. Since x is never inspected, injecting larger messages in it would not lead to new behaviour. We represent this arbitrary injection of messages in a limit by introducing a "wildcard" annotation on occurrences of names $a^\star$.

Technically, we duplicate the set of names $\mathcal{N}$ to a disjoint set of $\star$-annotated names $\mathcal{N}^\star := \{a^\star \mid a \in \mathcal{N}\}$, and we allow messages to contain names from $\mathcal{N} \uplus \mathcal{N}^\star$. All the definitions of this paper can be adapted straightforwardly to support wildcards by simply not distinguishing between a and $a^\star$. To stress the fact that a set of processes/limits contains annotations, we annotate the set with a wildcard, e.g. $\mathbb{L}^\star_{s,k}$.

▶ **Definition 27** (Wildcard semantics). *Given $P \in \mathbb{P}^\star$, its* wildcard semantics *is defined as*

$$\mathcal{W}[\![P]\!] := \left\{ P' \;\middle|\; \begin{array}{l} \mathrm{sf}(P) = \nu\vec{x}.(\langle\Gamma\rangle \parallel Q),\ P' \equiv_{\mathsf{kn}} \nu\vec{x},\vec{c}.\big((\langle\Gamma\rangle \parallel Q)[\vec{M}/\vec{y}^\star]\big) \\ \vec{y}^\star = \mathrm{names}(P) \cap \mathcal{N}^\star, \mathrm{names}(\vec{M}) \subseteq \vec{x} \cup \vec{c} \end{array} \right\}$$

For $L \in \mathbb{L}^\star_{s,k}$, define $\mathcal{W}[\![L]\!] := \{\, \mathcal{W}[\![P]\!] \mid P \in [\![L]\!] \,\}$.

With the help of wildcards, we can then take a limit L, annotate it with wildcards obtaining a limit $L^\star$. Then we can check that the wildcards generalise the limit enough to make it inductive wrt $\to_\Delta$ (i.e. without restricting the message size):

$$\forall P \in \mathcal{W}[\![L^\star]\!], \forall Q\colon P \to_\Delta Q \implies Q \in \mathcal{W}[\![L^\star]\!] \tag{3}$$

For our encryption oracle example, the annotated limit $\mathsf{E}[k] \parallel \big(\nu m.\langle \mathsf{e}(m^\star)_k\rangle\big)^\omega$ represents an inductive invariant for the unrestricted semantics.

For a more realistic application of generalised limits, consider our running Example 9. The limit of Figure 5 is inductive with respect to the typing $\mathsf{ty} = [n_a \mapsto 1]$. To remove the size bound we only need to annotate the occurrence of n_a in L_2 obtaining the limit $L^\star$:

$$\begin{aligned}
L^\star &= \nu a, b, k_{as}, k_{bs}.(\langle a, b\rangle \parallel \mathsf{A}_1[a, b, k_{as}]^\omega \parallel \mathsf{B}_1[a, b, k_{bs}]^\omega \parallel \mathsf{S}[a, b, k_{as}, k_{bs}]^\omega \parallel L_1^\omega)\\
L_1 &= \nu n_a.\big(\langle n_a\rangle \parallel \mathsf{A}_2[a, b, k_{as}, n_a] \parallel L_2^\omega\big)\\
L_2 &= \nu k.\big(\langle \mathsf{e}(k)_{(n_a^\star, k_{as})}\rangle \parallel \langle \mathsf{e}(k)_{k_{bs}}\rangle \parallel \mathsf{Secret}[k]^\omega \parallel \mathsf{A}_3[a, b, k_{as}, k]^\omega \parallel L_3^\omega\big)\\
L_3 &= \nu n_b.\big(\langle \mathsf{e}(n_b)_{(k,k)}\rangle \parallel \langle \mathsf{e}(n_b)_k\rangle \parallel \mathsf{B}_2[a, b, k_{bs}, n_b, k]\big)
\end{aligned}$$

Indeed this generalised limit is inductive: the intruder can send any message (M, b) to the server $\mathsf{S}[a, b, k_{as}, k_{bs}] := \mathbf{in}(x : (x, b)).\big(\dots\langle \mathsf{e}(k)_{(x,k_{as})}\rangle\dots\big)$, which will use it as part of the encryption key (M, k_{as}). None of the input patterns of the other processes would however be able to match the message $\mathsf{e}(k)_{(M,k_{as})}$ unless $M = n_a$; thus this attack attempt will not generate new behaviour, and our generalised limit captures all the reachable configurations without assuming bounds on the size of messages.

Similarly, it is not difficult to annotate the limits of our benchmarks[4] so that the generalised limits satisfy condition (3):

- ARPC: There is only one reasonable annotation: $(n_x^\star, (k, b))$. This is still inductive even though it can be fed back as we can have a different substitutions for $n_x^\star$.
- KSL: We annotate $\mathsf{B}_2[a, b, k_{bb}, k_{bs}, n_x^\star, n_y]$ in L_2 and $\mathsf{e}(nz, (b, n_x^\star,))_{k_{xy}}$ in L_4. A knows the actual n_x as it produced it and hence it is still inductive.
- KSLr: Additionally to KSL, we annotate $\mathsf{e}(m_x^\star, m_y)_{k_{xy}}$ and $\mathsf{B}_5[a, b, k_{bb}, k_{bs}, k_{ab}, t_y, m_x^\star, m_y]$ in L_7. The new message cannot flow into $\mathsf{B}_3[-]$ or $\mathsf{D}_2[-]$ as both pattern-match names on the second position which are different from m_y. However, due to the wildcard as parameter which could become m_y, $\mathsf{B}_4[-]$ can produce $\mathsf{B}_5[-]$'s input message. This is still covered and inductive but does not comply with a normal re-authentification run.
- NHS: We annotate $\mathsf{e}(n^\star, (b, \mathsf{e}(k, a)_{k_{bs}}))_{k_{as}}$ in L_2. A can only match on the original n so it is still inductive.
- NHSr: Same annotations as in NHS. Additionally, we annotate $\mathsf{e}(one, s^\star)_k$. in L_3 and hence have to annotate the s in $\mathsf{Secret}[s^\star]$ as well as $\mathsf{Leak}[s^\star]$. This covers all the enabled transitions and is hence inductive.
- NHSs: The same annotation as in NHS works.
- OR/ORl: We annotate $\mathsf{e}(n_y, (m^\star, (a, b)))_{k_{bs}}$ in L_2. This does not enable any new transition so it is inductive.
- ORs: Same annotation as OR/ORl. But we also have to annotate the key k_{xy} which we declare as secret which is the difference to ORl and hence does not work here.
- ORa: Similar annotation to OR/ORl. Analogously, inductivity is preserved.
- YAH: We annotate $\mathsf{e}(a, (n_a^\star, n_b))_{k_{bs}}$ in L_3. This can be fed back to $\mathsf{B}_2[-]$ and hence we also annotate $\mathsf{e}(n_b)_{(n_a^\star, n_b)}$ in L_5. This is still inductive.

[4] For the full limits for each benchmark see [20].

- YAHs1: We annotate as in YAH. The declaration of a secret happens only if A plays back and hence it is inductive.
- YAHs2: Without the size constraint for the key, the invariant obtained for YAHs1 is also one for YAHs2.
- YAHlk: We annotate $\mathsf{e}(one, s^\star)_k$ in L_4 which represents the case where the key k is leaked. This annotation covers all newly enabled transitions. In case k is not leaked, there are no new transitions. Hence, it is inductive.

B.2 Towards Automating Generalisation

Automating the check of condition (3) is beyond the scope of this paper, but we can sketch how one would approach the problem by adapting our $\widehat{\text{post}}$ definition to work on generalised limits. The idea is that one can design a computable function $\widehat{\text{post}}^{\mathsf{ty}}_{\Delta,\star}$, a "symbolic" version of $\widehat{\text{post}}$, and $\Subset$ satisfying:

$$L_1^\star \Subset L_2^\star \implies \mathcal{W}[\![L_1^\star]\!] \subseteq \mathcal{W}[\![L_2^\star]\!] \tag{4}$$

$$\text{post}(\mathcal{W}[\![L^\star]\!]) \subseteq \mathcal{W}[\![\widehat{\text{post}}^{\mathsf{ty}}_{\Delta,\star}(L^\star)]\!] \tag{5}$$

With these components, one can check (3) by checking $\widehat{\text{post}}^{\mathsf{ty}}_{\Delta,\star}(L^\star) \Subset L^\star$. Note that (4) is an implication, and (5) a subset relation: an equivalence would be impossible to achieve due to undecidability of the general problem; we therefore only require a sound over-approximation.

Designing $\Subset$ requires defining an approximate version of $\leq_{\mathsf{kn}}$ which can work on $\star$-annotated sets of messages. A precise version of this can only be defined on a per-intruder-model basis. A generic definition of $\Subset$ could simply extend the non-annotated inclusion check with the axioms $x^\star \leq_{\mathsf{kn}} x^\star$ and $M \leq_{\mathsf{kn}} x^\star$.

Designing a suitable $\widehat{\text{post}}^{\mathsf{ty}}_{\Delta,\star}(L^\star)$ similarly depends on the choice of intruder-model. One could, for example, define a symbolic matching function $\text{match}(\Gamma^\star, \vec{x} : N^\star)$ returning a finite set of substitutions such that

$$\Gamma \vdash N^\star\theta \wedge \Gamma \in \mathcal{W}[\![\Gamma^\star]\!] \implies \exists \sigma \in \text{match}(\Gamma^\star, \vec{x} : N^\star) \colon N^\star\theta \in \mathcal{W}[\![N^\star\sigma]\!]$$

and use it in $\widehat{\text{post}}^{\mathsf{ty}}_{\Delta,\star}$ to find all symbolic redexes. It is relatively straightforward to define a match function that is precise enough to check the generalised limits of our benchmarks. Exploring the design of these symbolic analyses in general is left for future work.

C Benchmarks

The Needham-Schröder protocol [28] is modelled with and without secrecy (NHS/NHSs). The NHSr version models leaks of old session keys, which leads to a replay attack (and hence the invariant is leaky). We provide four models of the Otway-Rees protocol [29]. OR does not model secrecy and is used to prove the protocol depth-bounded. ORl models secrecy but the inferred invariant contains a genuine leak, which is the result of a known type-confusion attack. The attack substitutes a composite message for some input x that is (wrongly) assumed to be a nonce by a principal. ORa models authentication and the invariant shows the genuine misauthentication based on this attack. ORs models the same situation with the assumption that x is of size one; with this assumption the inferred invariant is not leaky.

ARPC models Lowe's modified BAN concrete ARPC protocol [25] (we model succ(-) with pairs, i.e. succ((zero, $-$)) is (one, $-$)). We modelled the Kehne-Schönwälder-Landendörfer protocol [25], as modified by Lowe, with (KSLr) and without (KSL) re-authentication.

We produced four models of the Yahalom protocol [9]. Our first model (YAH) does not model secrecy and is used to establish depth-boundedness. The protocol has a type-confusion attack (similar to the case of Otway-Rees) which does not lead to a leak of a secret. This is

modelled in YAHs1. We rule out this type-confusion by adding a size assumption in YAHs2. YAHlk is a variant where an additional fresh nonce is exchanged at the beginning of each session, which makes the protocol secret even when old session keys can be leaked, a property entailed by the inferred invariant.

D Example of Incorporation

Consider the limit $L = \nu y.\Big(\big(\nu x.(\mathsf{A}[x] \parallel \mathsf{B}[y,x]^\omega)\big)^\omega\Big)$. To compute $\widehat{\text{post}}$ we consider the limit $L \otimes 1 = \nu y.\Big(\big(\nu x.(\mathsf{A}[x] \parallel \mathsf{B}[y,x] \parallel \mathsf{B}[y,x]^\omega)\big) \parallel \big(\nu x.(\mathsf{A}[x] \parallel \mathsf{B}[y,x]^\omega)\big)^\omega\Big)$ and assume

$$\mathsf{A}[x] \to \nu z.(A[z] \parallel \mathsf{B}[y,z]) \parallel \mathsf{B}[y,x] = P.$$

Then, $\widehat{\text{post}}$ will contain the limit $L' = C[P]$ for $C[\bullet] = \nu y.\Big(\big(\nu x.(\ \bullet \parallel \mathsf{B}[y,x] \parallel \mathsf{B}[y,x]^\omega)\big) \parallel \big(\nu x.(\mathsf{A}[x] \parallel \mathsf{B}[y,x]^\omega)\big)^\omega\Big)$. Since we are trying to prove inclusion between L and $L' = C[P]$, and L is equivalent to $L \otimes 1 = C[\mathsf{A}[x]]$, it seems likely that the inclusion could be proven by matching C with C in the two limits, and by matching P in $L \otimes 1$. Intuitively:

$$\nu y.\Big(\big(\nu x.(\ \mathsf{A}[x]\ \parallel \mathsf{B}[y,x] \parallel \underbrace{\mathsf{B}[y,x]^\omega}\ )\big) \parallel \underbrace{\big(\nu x.(\mathsf{A}[x] \parallel \mathsf{B}[y,x]^\omega)\big)}\ \Big)^\omega\Big)$$

$$\nu y.\Big(\big(\nu x.(\overbrace{\nu z.(\mathsf{A}[z] \parallel \mathsf{B}[y,z])}\ \parallel\ \overbrace{\mathsf{B}[y,x]}\ \parallel \mathsf{B}[y,x] \parallel\ \mathsf{B}[y,x]^\omega\ )\big) \parallel \big(\nu x.(\mathsf{A}[x] \parallel \mathsf{B}[y,x]^\omega)\big)^\omega\Big)$$

where the part in blue is P and the context C is the part in black. The arrows indicate the sublimits that can be further unfolded to show the inclusion of $C[P]$.

Formally, we want to check that $\forall L' \in \widehat{\text{post}}^s_\Delta(L)\colon [\![L']\!] \subseteq [\![L]\!]$. By construction, $L' = C[P]$ for some context C such that $L \otimes b = C[\mathsf{Q}[\vec{M}]]$. Then, if we can find P in the fixed part of $\text{sf}(L \otimes b)$, conditions **(A)** and **(B)** of Theorem 22 are automatically satisfied because of the shared context C.

Algebraic Invariants for Linear Hybrid Automata

Rupak Majumdar
Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany
rupak@mpi-sws.org

Joël Ouaknine
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
Department of Computer Science, Oxford University, UK
joel@mpi-sws.org

Amaury Pouly
Université de Paris, IRIF, CNRS, F-75013 Paris, France
amaury.pouly@irif.fr

James Worrell
Department of Computer Science, Oxford University, UK
jbw@mpi-sws.org

Abstract

We exhibit an algorithm to compute the strongest algebraic (or polynomial) invariants that hold at each location of a given guard-free linear hybrid automaton (i.e., a hybrid automaton having only unguarded transitions, all of whose assignments are given by affine expressions, and all of whose continuous dynamics are given by linear differential equations). Our main tool is a control-theoretic result of independent interest: given such a linear hybrid automaton, we show how to discretise the continuous dynamics in such a way that the resulting automaton has precisely the same algebraic invariants.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models; Theory of computation → Models of computation

Keywords and phrases Hybrid automata, algebraic invariants

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.32

Funding *Rupak Majumdar*: Supported by the European Research Council under Grant Agreement 610150 (ERC Synergy Grant ImPACT) and DFG grant 389792660 as part of TRR 248 (see `https://perspicuous-computing.science`).
Joël Ouaknine: Supported by ERC grant AVS-ISS (648701) and DFG grant 389792660 as part of TRR 248 (see `https://perspicuous-computing.science`).
Amaury Pouly: Part of this work was done at MPI-SWS.
James Worrell: Supported by EPSRC Fellowship EP/N008197/1.

Acknowledgements We thank Khalil Ghorbal for helpful discussions on the subject of this paper.

1 Introduction

Invariants are one of the most fundamental and useful notions in the quantitative sciences, appearing in a wide range of contexts, from gauge theory, dynamical systems, and control theory in physics, mathematics, and engineering to program verification, static analysis, abstract interpretation, and programming language semantics (among others) in computer science. In spite of decades of scientific work and progress, automated invariant synthesis remains a topic of active research, particularly in the fields of computer-aided verification and program analysis, and plays a central role in methods and tools seeking to establish correctness properties of computer systems; see, e.g., [8], and particularly Sec. 8 therein.

In this paper, we consider the task of computing *strongest algebraic inductive invariants* for *guard-free linear hybrid automata*. Hybrid automata are a formalism for describing systems or processes that combine discrete and continuous evolutions over their state variables (see, for example, [6]). A hybrid automaton is therefore equipped with a finite set of real-valued variables, as well as a finite set of control location (or modes). In each location, the variables evolve in continuous time according to some dynamics. Transitions between control locations may effect discrete updates (also known as *resets*) to these variables. A hybrid automaton is *guard-free* if the transitions do not have any guards, or preconditions, in order to be fired, and it is *linear* if the discrete updates on the variables consist entirely of affine transformations, and the continuous dynamics within each control location are defined by linear differential equations.[1]

An *invariant* assigns to each control location a fixed set of real values in such a way that through any trajectory of the hybrid automaton, the values of the variables always remain within the invariant. The invariant is *inductive* provided, informally speaking, that it is itself preserved by the (continuous and discrete) dynamics of the hybrid automaton. Finally, an invariant is *algebraic* (or polynomial) if it consists in a collection of varieties (or algebraic sets), i.e., positive Boolean combination of polynomial equalities. As it happens, the strongest polynomial invariant (i.e., smallest variety with respect to set inclusion) is obtained by taking the Zariski closure of the set of reachable configurations in each control location; such an invariant is always inductive provided that the dynamics are Zariski continuous.

There is a rich history of research into the computation of algebraic invariants for various classes of (discrete) computer programs; we refer the reader to our recent paper [7] and references therein. There has also been a substantial amount of work on algebraic invariant generation for hybrid systems, albeit in more recent years. One of the earliest pieces of work on this topic is by Rodrígez-Carbonell and Tiwari [15], who consider linear dynamical systems (i.e., linear hybrid automata with a single discrete location and no transition) and show how to compute strongest algebraic invariants for these. They then leverage abstract-interpretation techniques to derive algebraic invariants for linear hybrid automata, however without guarantees on the strength of the invariants. In [17], Sankaranarayanan et al. compute algebraic invariants for polynomial hybrid systems directly using constraint solving over template invariants (without however guaranteeing to obtain the strongest invariant). In subsequent work, Sankaranarayanan shows how to compute strongest algebraic invariants up to a fixed degree [16] for the same class of automata. Using different analytic techniques, Ghorbal and Platzer show in [5] how to compute algebraic invariants and differential invariants for polynomial hybrid automata; more precisely, they show that it is decidable whether a collection of algebraic sets forms an algebraic invariant; however they do not provide a procedure to guarantee that a given invariant is the strongest possible. In fact, while algebraic invariants for various classes of systems is a well-studied topic (see e.g., [2, 4, 13, 14, 1, 9] and references therein), as far as we know, none of these papers unconditionally guarantee the *strongest* algebraic invariants when applied to hybrid automata.

Main results. Our contributions in the present paper are threefold. **First,** for the class of guard-free linear hybrid automata, building on our recently developed invariant-generation techniques for affine programs [7], we show how to compute strongest algebraic invariants.

[1] Following [6], the control locations of hybrid automata can also be equipped with "invariant conditions" meant to enforce discrete transitions when the continuous variables reach the limit of their allowed range; however the hybrid automata considered in this paper do not feature any such invariant conditions.

Our main technical tools come from linear algebra, algebraic geometry, and Diophantine geometry. **Second,** we show how one can discretise a guard-free linear hybrid automaton in such a way that the discretised version has precisely the same algebraic invariants as the original one; we are not aware of any such result in the extant control-theoretic and cyber-physical systems literature. **Third,** we show that as soon as equality guards are allowed, even for the restricted class of linear *switching systems*, there cannot exist an algorithm for computing strongest algebraic invariants, thereby establishing clearly a hard theoretical limit on how far the work presented here can be extended.

We now provide a slightly more detailed overview of our approach and results. Our main theorem uses an effective reduction from guard-free linear hybrid automata to affine programs that preserves algebraic invariants. More formally, given a hybrid automaton we show how to construct a finite (discrete-time) affine program that has the same set of variables and the same algebraic invariants as the original hybrid automaton. Thus we reduce the problem of computing strongest algebraic invariants for hybrid automata to the analogous problem for affine programs. In particular, this allow us to use a result in [7] to compute strongest algebraic invariants for hybrid automata. In fact, we show a stronger discretisation result that replaces the continuous dynamics with a finite set of discrete actions, which already preserves algebraic invariants.

In Section 7 we consider a simple but important class of hybrid systems, with purely continuous dynamics, called *switching systems*. A switching system [10] can transition arbitrarily between modes, but the variables are not reset when changing mode. It is natural to think of mode switches as being determined by an external controller which provides inputs to the system. We show that for a switching system, the Zariski closure of the set of reachable configurations is an irreducible variety. We exploit this feature to give a conceptually simple algorithm to compute the strongest algebraic invariant of a given switching system.

On the other hand, we consider the problem of computing the strongest algebraic invariant for switching systems that are augmented with the ability to test variables for zero on transitions. Here again the dynamics are exclusively continuous. We show that it is undecidable in general to compute a strongest algebraic invariant for such systems. Roughly speaking, we prove this result by defining a simulation of an arbitrary Minsky machine $\mathcal{C}$ by a hybrid automaton $\mathcal{A}$, such that we can effectively determine whether the set of reachable configurations of $\mathcal{C}$ is infinite from the strongest algebraic invariant of $\mathcal{A}$.

2 Examples

2.1 Bouncing ball

Consider a ball that bounces on horizontal slabs, as illustrated in Figure 1. The ball is moving at constant horizontal speed c and is subject to gravity along the vertical axis. We assume that there is no friction and that collisions are perfectly elastic. We do not want to make any assumption on the location of the slabs, to obtain the most general system. Thus from the system's point of the view, the positions of the slabs are "nondeterministic" and the slabs can "appear" at any moment.

We fix an arbitrary coordinate system in which the ball starts at position $(0, h)$ with initial velocities $(c, 0)$. The system is modelled using a single discrete location. There are three constants c (the horizontal speed), h (the initial height), and g (approximately $9.8ms^{-2}$). Technically speaking we could also include m (the mass of the ball), but it will not feature in the final set of invariants that we derive nor in our differential equations. There are five variables: t, x, y, v_x, and v_y, where t is the time variable. The differential equations

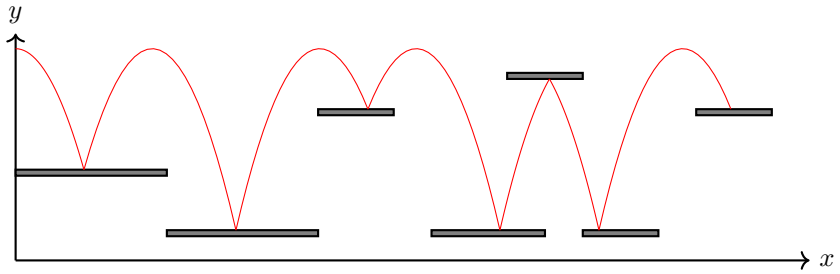


Figure 1 A ball bouncing on horizontal slabs.

are simply Newton's laws of motion. There is a single reset with no guards modelling the action of the ball bouncing on each of the slabs (notably ensuring that the vertical velocity is instantly inverted). We obtain the hybrid system described in Figure 2a.

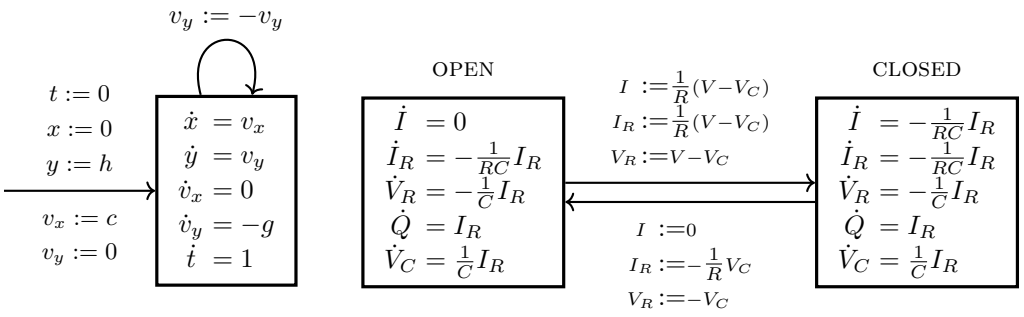
Now we come to the invariants. The most obvious is that the horizontal speed is constant: $v_x = c$, which in turn entails that $x = tc$. Next, consider an inertial coordinate system moving horizontally to the right at speed c. In that system energy must be conserved. Initially there is no kinetic energy, and all the potential energy amounts to mgh (where m is the mass of the ball). At any subsequent time t, the sum of the kinetic and potential energy must therefore sum to that value, i.e., $\frac{1}{2}mv_y^2 + mgy = mgh$, so $v_y^2 + 2g(y-h) = 0$. These must be the only invariants: the ambient space is 5-dimensional (given that our variables are t, x, y, v_x, and v_y), and the corresponding variety (with 3 equations) is two dimensional. But the system has indeed exactly two degrees of freedom, since t and v_y can be set to arbitrary values (provided $|v_y| \leq gt$), and once t and v_y are fixed, every other variable is fixed. In summary, the strongest algebraic invariant is the conjunction of the following three equations:

$$v_x = c, \qquad x = tc, \qquad v_y^2 + 2g(y-h) = 0. \tag{1}$$

One way to obtain the invariants in (1) is to construct an affine program (i.e., a hybrid system with trivial continuous dynamics) over the same set of locations and variables as the original hybrid automaton and that moreover has the same algebraic invariants. One can then apply Theorem 2 to compute a strongest invariant of the latter. In the case at hand, such an affine program can be obtained by a direct time-discretisation construction in which the continuous flow of variables is replaced by a self-loop – see Figure 4a – which performs a simultaneous assignment $x := x + v_x$, $y := y + v_y - \frac{1}{2}g$, $v_y := v_y - g$ and $t := t + 1$. Here, essentially, we have replaced a system of linear differential equations of the form $\dot{x}(t) = Ax(t)$ with an analogous system of linear difference equations $x(t+1) = e^A x(t)$; this is sound because the discrete semigroup $\{e^{An} : n \in \mathbb{N}\}$ is Zariski dense is the semigroup $\{e^{At} : t \geq 0\}$ (see Proposition 13). Unfortunately, due to the presence of the matrix exponential, this naive construction in general yields affine programs with transcendental constants, which precludes applying the algorithm described in Theorem 2. As it happens, in the case at hand, the matrix e^A has rational entries. We refer to Section B of the Appendix for more details of how the affine program is obtained.

2.2 RC circuit

Consider an RC circuit with a switch, as illustrated in Figure 3. When the switch is on, the capacitor is connected to a battery and charges. When the switch is off, the capacitor discharges through the resistor. The battery has constant voltage V, the resistor has resistance R and the capacitor has capacity C. There are 5 variables: the current I in the wire between the battery and the switch, the voltages V_R and V_C across the resistor and



(a) Bouncing ball. **(b)** RC circuit.

Figure 2 Examples of hybrid systems: (a) bouncing ball, (b) RC circuit.

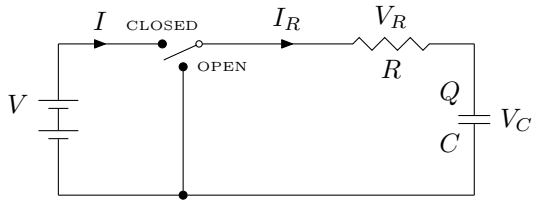


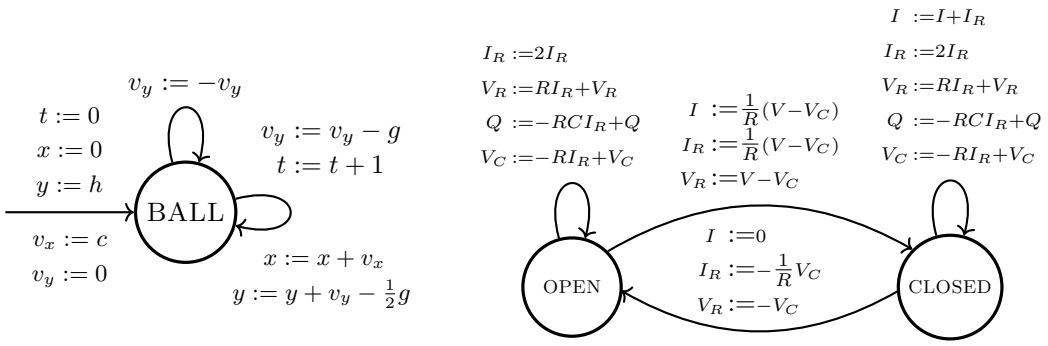
Figure 3 An RC circuit with a switch to disconnect the battery.

capacitor respectively, the current I_R flowing through the resistor, and finally the charge Q held by the capacitor. All derivatives are with respect to time, though this time we choose not to include an explicit variable for the passage of time. There are two discrete locations, OPEN and CLOSED, corresponding to the two possible positions of the switch. We assume the switch starts in the OPEN position. When switching from OPEN to CLOSED, all variables but Q and V_C experience a reset. We obtain the hybrid system described in Figure 2b.

In this example, there is one set of invariants per location. In both locations, we clearly have the invariants associated with the passive components: $Q = CV_C$ and $V_R = RI_R$. In the OPEN location, we further have the invariants $I = 0$ and $V_R = -V_C$. On the other hand, in the CLOSED location, we have $I = I_R$ and $V = V_R + V_C$. These must be the only invariants: once Q is fixed, all variables are uniquely determined. In summary, the invariants are

$$\begin{array}{lllll} \text{OPEN:} & Q = CV_C, & V_R = RI_R, & I = 0, & V_R = -V_C, \\ \text{CLOSED:} & Q = CV_C, & V_R = RI_R, & I = I_R, & V_R = V - V_C. \end{array}$$

The above invariants were obtained by producing an affine program with the same set of invariants as the hybrid system in Figure 2b, and applying Theorem 2 to compute a strongest invariant of the latter. However, in this case, the naive discretisation procedure that was used in Section 2.1 yields transcendental constants (for the reasons described in Section 2.1). In fact the affine program that we construct, shown in Figure 4b, is the result of applying the more abstract discretisation procedure that is described in Section 6. The core idea of the latter procedure is, given a square matrix A with rational coefficients, to produce a matrix B, with algebraic coefficients, such that the respective semigroups $\{e^{At} : t \geq 0\}$ and $\{B^n : n \in \mathbb{N}\}$ have the same Zariski closure (see Proposition 9). The algebraic nature of this construction makes it more subtle to relate the dynamics of the resulting affine program to the original hybrid system (we invite the reader to compare the automata shown respectively in Figures 2b and 4b). We refer to Section B of the Appendix for more details of how the affine program is obtained.



(a) Affine program modelling a bouncing ball. **(b)** Affine program modelling an RC circuit.

Figure 4 Time discretisation of the hybrid systems in Figure 2.

3 Mathematical Background

Let $\mathbb{K}$ be a field. Given a set $X \subseteq \mathbb{K}^n$, we denote by $\mathbf{I}(X)$ the ideal of polynomials in $\mathbb{K}[x_1, \ldots, x_n]$ that vanish on X. Given an ideal $I \subseteq \mathbb{K}[x_1, \ldots, x_n]$, we denote by $V(I) \subseteq \mathbb{K}^n$ the set of common zeroes of the polynomials in I. A set $X \subseteq \mathbb{K}^n$ is said to be an *affine variety* (also called an *algebraic set*) if $X = V(I)$ for some ideal $I \subseteq \mathbb{K}[x_1, \ldots, x_n]$. By the Hilbert Basis Theorem, every affine variety can be described as the set of common zeroes of finitely many polynomials. We identify $\mathrm{GL}_n(\mathbb{K})$, the set of $n \times n$ invertible matrices with entries in $\mathbb{K}$, with the variety $\{(A, y) \in \mathbb{K}^{n^2+1} : \det(A) \cdot y = 1\}$.

Given an affine variety $X \subseteq \mathbb{K}^n$, the *Zariski topology* on X has as closed sets the subvarieties of X, i.e., those sets $A \subseteq X$ that are themselves affine varieties in $\mathbb{K}^n$. Given an arbitrary set $S \subseteq X$, we write $\overline{S}$ for its closure in the Zariski topology on X.

A set $S \subseteq X$ is *irreducible* if for all closed subsets $A_1, A_2 \subseteq X$ such that $S \subseteq A_1 \cup A_2$ we have either $S \subseteq A_1$ or $S \subseteq A_2$. It is well known that the Zariski topology on a variety is Noetherian. In particular, any closed subset A of X can be written as a finite union of *irreducible components*, where an irreducible component of A is a maximal irreducible closed subset of A.

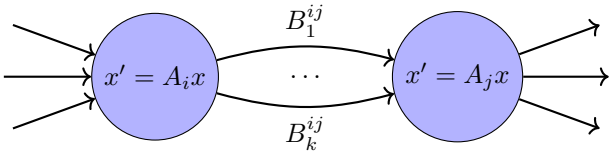
The class of *constructible* subsets of a variety X is obtained by taking all finite Boolean combinations (including complementation) of Zariski closed subsets. Suppose that the underlying field $\mathbb{K}$ is algebraically closed. Since the first-order theory of algebraically closed fields admits quantifier elimination, the constructible subsets of X are exactly the subsets of X that are first-order definable over $\mathbb{K}$.

A function $f : \mathbb{K}^m \to \mathbb{K}^n$ is said to be a *polynomial map* if there exist polynomials $p_1, \ldots, p_n \in \mathbb{K}[x_1, \ldots, x_m]$ such that $f(\boldsymbol{a}) = (p_1(\boldsymbol{a}), \ldots, p_n(\boldsymbol{a}))$ for all $\boldsymbol{a} \in \mathbb{K}^m$. Recall that polynomial maps are Zariski-continuous and thus $f(\overline{X}) \subseteq \overline{f(X)}$ for a polynomial map f. In particular matrix multiplication is a Zariski-continuous map $\mathbb{K}^{n^2} \times \mathbb{K}^{n^2} \to \mathbb{K}^{n^2}$.

Given a complex variety $V \subseteq \mathbb{C}^n$, the intersection $V \cap \mathbb{R}^n$, which is a real variety, can be computed effectively. Indeed if V is represented by the ideal $I \subseteq \mathbb{C}[x_1, \ldots, x_n]$, then $V \cap \mathbb{R}^n$ is represented by the ideal generated by $\{p_1, p_2 \in \mathbb{R}[x_1, \ldots, x_n] : p_1 + ip_2 \in I\}$ Given $S \subseteq \mathbb{R}^n$, write $\overline{S}^{\mathbb{R}}$ for its real Zariski closure and $\overline{S}$ for its complex Zariski closure (i.e., we treat the complex Zariski closure of $S \subseteq \mathbb{R}^n$ as the default). It is straightforward to verify that $\overline{S}^{\mathbb{R}} = \overline{S} \cap \mathbb{R}^n$.

4 Algebraic Invariants for Hybrid Automata

We are concerned with computing strongest algebraic invariants for the subclass of hybrid automata that has no guards, linear discrete updates, and linear continuous dynamics. Each location of such an automaton specifies a linear differential equation ($x' = Ax$), and each transition between states (nondeterministic choices are allowed) specifies a linear transformation ($x \mapsto Bx$). Such a hybrid automaton can be pictured as follows:



Formally, such an automaton $\mathcal{A}$ in dimension d is a tuple (Q, A, E, T), where Q is a finite set of locations, $A = \{A_q : q \in Q\}$ is a family of real $d \times d$ matrices, $E \subseteq Q \times \mathbb{R}^{d\times d} \times Q$ is a set of transitions labelled by real $d \times d$ matrices, and $\{T_q : q \in Q\}$ is a family of algebraic subsets of $\mathbb{R}^d$. Matrix $A_q \in \mathbb{R}^{d\times d}$ describes the continuous dynamics at location $q \in Q$ and $T_q \subseteq \mathbb{R}^d$ is the set of initial states in location q. We assume that the entries of all matrices are algebraic numbers and that the polynomials defining T_q have algebraic coefficients.

We will consider the subclasses of automata with the following restrictions:

- *affine programs*: $A_q = 0$ for all $q \in Q$, and E is finite,
- *constructible affine programs*: $A_q = 0$ for all $q \in Q$, and E is constructible,
- *switching systems*: $E = \{(p, I_n, q) : p, q \in Q\}$, i.e., every pair of locations is connected by an edge that does not update the variables,
- *linear hybrid automata*: E is finite.

In affine programs, variables are only updated on discrete edges: there is no continuous evolution within locations. At the other end of the spectrum, in switching systems variables only evolve continuously, and there are no discrete updates. The full class of linear hybrid automata accommodate both discrete and continuous updates to the variables.

The *collecting semantics* of $\mathcal{A}$ assigns to each location $q \in Q$ the set $S_q \subseteq \mathbb{R}^d$ of states that can occur in location q during a run of the automaton, starting from a configuration $(q, \boldsymbol{a})$ for some $\boldsymbol{a} \in T_q$. Formally, this the smallest family (with respect to set inclusion) such that

$$\begin{array}{lcll} S_q & \supseteq & T_q & \text{for all } q \in Q, \\ S_q & \supseteq & BS_p & \text{for all } (p, B, q) \in E, \\ S_q & \supseteq & e^{A_q t} S_q & \text{for all } t \in \mathbb{R}_{\geqslant 0}. \end{array}$$

Equivalently, let the operator $\Phi_{\mathcal{A}} : \mathcal{P}(\mathbb{R}^d)^Q \to \mathcal{P}(\mathbb{R}^d)^Q$ be defined by

$$\big(\Phi_{\mathcal{A}}(S)\big)_q = T_q \cup \bigcup_{t\geqslant 0} e^{A_q t} S_q \cup \bigcup_{(p,B,q)\in E} BS_p.$$

Then S is the least fixed-point of $\Phi_{\mathcal{A}}$ with respect to set inclusion. Such a least fixed-point exists because $\Phi_{\mathcal{A}}$ is monotone.

In general we say that a family of sets $\{S'_q\}_{q\in Q}$, with $S'_q \subseteq \mathbb{R}^d$ is an *inductive invariant* if $\Phi_{\mathcal{A}}(S') \subseteq S'$, i.e., the family is pre-fixed-point of $\Phi_{\mathcal{A}}$. If each set S'_q is algebraic then we moreover say that $\{S'_q\}_{q\in Q}$ is an *inductive algebraic invariant*.

Given $P \in \mathbb{R}[x_1, \ldots, x_d]$, we say that the relation $P = 0$ holds at location q if P vanishes on S_q. We are interested in computing at each location $q \in Q$ a finite set of polynomials that generates the ideal $I_q := \mathbf{I}(S_q) \subseteq \mathbb{R}[x_1, \ldots, x_d]$ of all polynomial relations that hold at location q. The real variety $V_q := V(I_q) = \overline{S_q}^{\mathbb{R}}$ corresponding to I_q is the Zariski closure of S_q viewed a subset of the affine space $\mathbb{R}^d$.

Note that the collection $V(\mathcal{A}) := \{V_q : q \in Q\}$ defines an inductive algebraic invariant. Inductiveness amounts to the following two claims:

- for every edge $(p, B, q) \in E$, we have $BV_p \subseteq V_q$,
- for every $q \in Q$ and $t \in \mathbb{R}_{\geqslant 0}$, we have $e^{A_q t} V_q \subseteq V_q$.

The first point follows from the fact that $x \mapsto Bx$ is Zariski-continuous; the second point likewise follows from the fact that for every $t \in \mathbb{R}_{\geqslant 0}$ the map $x \mapsto e^{A_q t} x$ is Zariski-continuous.

▶ **Lemma 1.** *For an automaton $\mathcal{A}$, $V(\mathcal{A})$ is the least fixpoint of the map $X \mapsto \overline{\Phi_{\mathcal{A}}(X)}$.*

Proof. Let S denote the collecting semantics of $\mathcal{A}$, then

$$\begin{aligned} V(\mathcal{A}) &= \overline{S} && \text{by definition of } V(\mathcal{A}) \\ &= \overline{\Phi_{\mathcal{A}}(S)} && \text{by definition of } S \\ &= \overline{\Phi_{\mathcal{A}}(\overline{S})} && \text{by Zariski-continuity of } \Phi_{\mathcal{A}} \\ &= \overline{\Phi_{\mathcal{A}}(V(\mathcal{A}))} \end{aligned}$$

thus $V(\mathcal{A})$ is indeed a fixed-point. Conversely, let X be such that $X = \overline{\Phi_{\mathcal{A}}(X)}$. Then X is closed and clearly $\Phi_{\mathcal{A}}(X) \subseteq \overline{\Phi_{\mathcal{A}}(X)} = X$ so it is a pre-fixpoint of $\Phi_{\mathcal{A}}$. By virtue of S being the least (pre-)fixpoint of $\Phi_{\mathcal{A}}$ we must have $S \subseteq X$. But then $\overline{S} \subseteq X$ i.e., $V(\mathcal{A}) \subseteq X$. ◀

The discussion above shows that $V(\mathcal{A})$, the Zariski closure of the collecting semantics, is the least inductive invariant of $\mathcal{A}$. Previously we have shown how to compute the Zariski closure of the collecting semantics of an affine program:

▶ **Theorem 2** ([7])**.** *There is an algorithm that given a constructible affine program $\mathcal{A}$ computes $V(\mathcal{A}) = \{V_q : q \in Q\}$ – the real Zariski closure of its collecting semantics.*

Note that this theorem also applies to (finite) affine programs since those are particular instances of constructible affine programs.

The main result of the current paper extends the above result by accommodating the continuous dynamics of hybrid automata:

▶ **Theorem 3.** *There is an algorithm that given a guard-free linear hybrid automaton $\mathcal{A}$ computes $\{V_q : q \in Q\}$ – the real Zariski closure of its collecting semantics.*

5 Proof of Theorem 3

The proof of Theorem 3 has two main ingredients. First, in Subsection 5.1, we show how to compute the Zariski closure of the orbit of a single continuous linear dynamics (i.e., the continuous evolution in a single state). Then, in Subsection 5.2, we show that computing the real Zariski closure of the collecting semantics of a linear hybrid automaton can be effectively reduced to computing the real Zariski closure of the collecting semantics of a constructible affine program. At this point, we can apply the algorithm from Theorem 2.

5.1 Linear Continuous Dynamics

The first step towards computing Zariski closure in the general case is to be able to handle the case of one differential equation. Write $\mathbb{A}$ for the field of algebraic numbers. Let $A \in \mathbb{A}^{d\times d}$ and $x_0 \in \mathbb{A}^d$, then the solution to $x(0) = x_0$, $x'(t) = Ax(t)$ is given by $x(t) = e^{At}x_0$. We are interested in computing the Zariski closure of the orbit $\{x(t) : t \in \mathbb{R}_{\geqslant 0}\}$. Since the map $\phi : M \mapsto Mx_0$ is Zariski-continuous, we have that

$$\overline{\{x(t) : t \in \mathbb{R}_{\geqslant 0}\}} = \overline{\{e^{At}x_0 : t \in \mathbb{R}_{\geqslant 0}\}} = \overline{\phi(\{e^{At} : t \in \mathbb{R}_{\geqslant 0}\})} = \overline{\phi(\overline{\{e^{At} : t \in \mathbb{R}_{\geqslant 0}\}})}$$

and thus it suffices to compute $\overline{\{e^{At} : t \in \mathbb{R}_{\geqslant 0}\}}$. Furthermore, let $\mathcal{O}_A := \{e^{At} : t \in \mathbb{R}\}$, which is a commutative group. We claim that $\overline{\{e^{At} : t \in \mathbb{R}_{\geqslant 0}\}} = \overline{\mathcal{O}_A}$. The left-to-right inclusion is clear. The converse inclusion comes from the fact that an exponential polynomial (in one variable), being an analytic function, vanishes over $\mathbb{R}_{\geqslant 0}$ if and only if it vanishes over $\mathbb{R}$. Thus it suffices to compute $\overline{\mathcal{O}_A}$.

The following lemma gives a description of the ideal of the variety $\overline{\mathcal{O}_A}$ when A is diagonal.

▶ **Lemma 4.** *Let $A = \mathrm{diag}(\lambda_1, \ldots, \lambda_d) \in \mathbb{A}^{d\times d}$ be a diagonal matrix, then*

$$\overline{\mathcal{O}_A} = \{\mathrm{diag}(z_1, \ldots, z_d) : \forall p \in I, p(z_1, \ldots, z_d) = 0\} ,$$

where $I = \langle z^a - z^b : a - b \in L\rangle$ and $L = \{n \in \mathbb{Z}^d : n_1\lambda_1 + \cdots + n_d\lambda_d = 0\}$. Furthermore, one can compute a basis for L considered as an abelian group under addition.

Proof. Clearly $e^{At} = \mathrm{diag}(e^{\lambda_1 t}, \ldots, e^{\lambda_d t})$. Since the set of diagonal matrices is closed, then the closure is of the form

$$\overline{\mathcal{O}_A} = \{\mathrm{diag}(z_1, \ldots, z_n) : p_1(z) = \cdots = p_k(z) = 0\}$$

for some polynomials $p_1, \ldots, p_k$. Thus, the ideal I of the closure is generated by all polynomials p such that $p(e^{\lambda_1 t}, \ldots, e^{\lambda_d t}) = 0$ for all $t \in \mathbb{R}$. Let J be the ideal of all polynomials $x^a - x^b$ with $a, b \in \mathbb{N}^d$ such that $\lambda \cdot a = \lambda \cdot b$. Clearly $J \subseteq I$ since if $\lambda \cdot a = \lambda \cdot b$, then $(e^{\lambda_1 t})^{a_1} \cdots (e^{\lambda_d t})^{a_d} - (e^{\lambda_1 t})^{b_1} \cdots (e^{\lambda_d t})^{b_d} = e^{(\lambda\cdot a)t} - e^{(\lambda\cdot b)t} = 0$ for all $t \in \mathbb{R}$. Conversely, assume by contradiction that $p \in I \setminus J$ and write $p = \sum_{i=1}^{r} b_i m_i$ for some $b_i \in \mathbb{A}$ and monomials $m_1, \ldots, m_r$. Further choose p so that r is minimal. For each monomial $m_i(x) = x_1^{a_1} \cdots x_d^{a_d}$, let $\mu_i := \lambda \cdot a$. Then we must have $\mu_i \neq \mu_j$ for $i \neq j$ because otherwise $m_i - m_j \in J$ and $p - b_i(m_i - m_j) \in I \setminus J$ would have fewer terms than p. Since the maps $t \mapsto e^{\mu_1 t}, \ldots, t \mapsto e^{\mu_d t}$ are linearly independent, it follows that $b_1 = \cdots = b_r = 0$ which is contradiction. Thus we must have have $I = J$. It is immediate to see that J is generated by all polynomials $x^a - x^b$ such that $\lambda \cdot (a - b) = 0$, thus it suffices to compute $L = \{a \in \mathbb{Z}^d : \lambda \cdot a = 0\}$, the set of additive relations of λ. Notice that L is an additive subgroup of $\mathbb{Z}^d$ and as such must be finitely generated. An upper bound on the size of the elements of a basis of L can be found in [11] and therefore we can compute a basis for L and thus J, I and $\overline{\mathcal{O}}_A$. ◀

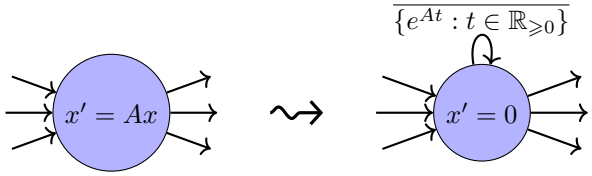
A corollary of this result is that we can compute $\overline{\mathcal{O}_A}$ by separating the diagonal and nilpotent parts of A. The fact that this closure is computable is well known (see, e.g., [15]), however the particular form of the polynomials determining the Zariski closure is the important part of Lemma 4 for our purposes. In particular, Lemma 4 is instrumental in the proof of Proposition 6, which reduces the problem of computing the strongest algebraic invariants of hybrid automata to the analogous problem for affine programs.

▶ **Proposition 5.** *There is an algorithm that given $A \in \mathbb{A}^{d\times d}$, computes $\overline{\{e^{At} : t \in \mathbb{R}_{\geqslant 0}\}}$.*

Proof. We can write $A = P(D+N)P^{-1}$ where P is invertible, D is diagonal, N is nilpotent, and D and N commute. Notice that $\mathcal{O}_D$ only consists of diagonal matrices and $\mathcal{O}_N$ of unipotent matrices. Since $\mathcal{O}_A$ is a commutative group, and D and N commute, we have that $\mathcal{O}_A = P(\mathcal{O}_D \cdot \mathcal{O}_N)P^{-1}$, and thus $\overline{\mathcal{O}_A} = P\overline{\mathcal{O}_D \cdot \mathcal{O}_N}P^{-1}$. All the operations in this equation are effective thus it suffices to compute $\overline{\mathcal{O}_D}$ as explained above, and $\overline{\mathcal{O}_N}$. But since N is nilpotent, $q_n(t) := e^{Nt}$ is really a polynomial in Nt and thus in t. It follows that $\overline{\mathcal{O}_N} = \overline{q_n(\mathbb{R})} = \overline{q_n(\mathbb{R})} = \overline{q_n(\mathbb{C})}$ which we know how to compute. ◀

5.2 From Continuous Dynamics to Constructible Discrete Dynamics

In the previous section, we saw that given a linear continuous system, one can compute its strongest inductive algebraic invariant. The main obstacle to generalize this approach to linear hybrid system is the mixture of discrete and continuous dynamics. In particular, the invariants are not irreducible. The idea to work around this problem is to replace the continuous evolution of the variables in each location of a hybrid automaton with an infinite set of discrete transitions. Graphically this corresponds to rewriting the automaton as follows:



The key point is that this infinite set is very special: it is constructible (and in fact algebraic), and hence falls into the scope of our previous paper [7].

▶ **Proposition 6.** *Given a linear hybrid automaton $\mathcal{A}$, one can compute a constructible affine program $\mathcal{A}'$ that has the same algebraic invariants, i.e., $V(\mathcal{A}) = V(\mathcal{A}')$.*

Proof. The idea is to replace each continuous dynamics $x' = A_q x$ by a closed (and thus constructible) set of discrete transitions: $\overline{\{e^{A_q t} : t \in \mathbb{R}_{\geqslant 0}\}}$, which we know how to compute thanks to Proposition 5.

Formally, let $\mathcal{A} = (Q, A, E, T)$ be a linear hybrid automaton. Define the constructible affine program $\mathcal{A}' = (Q, A', E', T)$ where $A'_q = 0$ for all $q \in Q$ and

$$E' = E \cup \left\{(q, X, q) : q \in Q, X \in \overline{\mathcal{O}_{A_q}}\right\}$$

where $\mathcal{O}_A := \left\{e^{At} : t \in \mathbb{R}_{\geqslant 0}\right\}$. Note that it is constructible because $\overline{\mathcal{O}_{A_q}}$ is closed (and thus constructible).

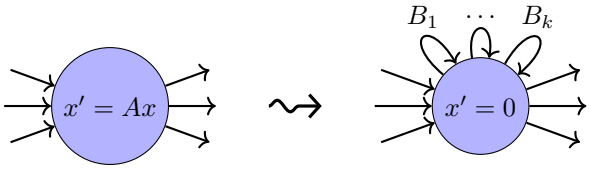
We will now relate the operators $\Phi_{\mathcal{A}}$ and $\Phi_{\mathcal{A}'}$. Given $X \in \mathcal{P}(\mathbb{C}^d)^Q$ we have

$$\begin{aligned}
\overline{\Phi_{\mathcal{A}}(X)_q} &= \overline{T_q \cup (\mathcal{O}_{A_q} \cdot X_q) \cup \bigcup_{(p,B,q)\in E} BX_p} \\
&= \overline{T_q \cup \overline{\mathcal{O}_{A_q} \cdot X_q} \cup \bigcup_{(p,B,q)\in E} BX_p} \\
&= \overline{T_q \cup (\overline{\mathcal{O}_{A_q}} \cdot X_q) \cup \bigcup_{(p,B,q)\in E} BX_p} \\
&= \overline{\Phi_{\mathcal{A}'}(X)_q}\,.
\end{aligned}$$

Thus the maps $X \mapsto \overline{\Phi_{\mathcal{A}}(X)}$ and $X \mapsto \overline{\Phi_{\mathcal{A}'}(X)}$ are identical. But, by Lemma 1, $V(\mathcal{A})$ is the least fixpoint of $X \mapsto \overline{\Phi_{\mathcal{A}}(X)}$ and $V(\mathcal{A}')$ is the least fixpoint of $X \mapsto \overline{\Phi_{\mathcal{A}'}(X)}$. We conclude that $V(\mathcal{A}') = V(\mathcal{A})$. ◀

6 From Continuous Dynamics to Finite Discrete Dynamics

The proof of Theorem 3 used an effective reduction from a linear hybrid automaton to a constructible affine program with the same set of algebraic invariants. In this section, we show a stronger result – of independent control-theoretic interest – that one can in fact reduce a linear hybrid automaton to a *finite* affine program with the same set of algebraic invariants. The idea is to replace the continuous evolution of the variables in each location of a hybrid automaton with a finite set of discrete transitions. Graphically this corresponds to rewriting the automaton as follows:



The result is stronger because finite affine programs are also constructible.

Mathematically, the task is as follows: *Given* $A \in \mathbb{A}^{d\times d}$, *find* $B_1, \ldots, B_k \in \mathbb{A}^{d\times d}$ *such that* $\overline{\{e^{At} : t \in \mathbb{R}\}} = \overline{\langle B_1, \ldots, B_k\rangle}$. There is a conceptually simple approach to this problem: namely for every matrix $A \in \mathbb{A}^{d\times d}$ and rational number τ we have $\overline{\{e^{At} : t \in \mathbb{R}\}} = \overline{\langle e^{A\tau}\rangle}$ (see Section A). But this does not fulfil our desiderata, since it is not possible in general to find $\tau \in \mathbb{R}$ such that $e^{A\tau}$ has exclusively algebraic entries. Nevertheless given $A \in \mathbb{A}^{d\times d}$ it is possible to find $B \in \mathbb{A}^{d\times d}$ such that $\overline{\{e^{At} : t \in \mathbb{R}\}} = \overline{\langle B\rangle}$. The idea is to construct B such that there is a correspondence between the set of additive relations satisfied by the eigenvalues of A and the multiplicative relations satisfied by the eigenvalues of B.

▶ **Proposition 7.** *Let* $a_1, \ldots, a_d \in \mathbb{C}$ *be algebraic numbers. Then we can compute rational numbers* $\lambda_1, \ldots, \lambda_d$ *such that* $a_1 n_1 + \cdots + a_d n_d = 0$ *iff* $\lambda_1^{n_1} \cdots \lambda_d^{n_d} = 1$ *for all* $n_1, \ldots, n_d \in \mathbb{Z}$.

Proof. Let s be the dimension of the $\mathbb{Q}$-vector space spanned by $a_1, \ldots, a_d$. By computing a basis over $\mathbb{Q}$ for the number field $\mathbb{Q}(a_1, \ldots, a_d)$ and the respective rational coordinates of $a_1, \ldots, a_d$ with respect to this basis, we obtain an $s \times d$ integer matrix A such that for every integer vector $\boldsymbol{x} = (n_1, \ldots, n_d) \in \mathbb{Z}^d$ we have $n_1 a_1 + \cdots + n_d a_d = 0$ iff $A\boldsymbol{x} = 0$.

Now write $A = PBQ$, where B is an $s \times d$ matrix in Smith normal form and P, Q are unimodular square matrices. Since B has rank s it has the form $B = \begin{pmatrix} D & 0 \end{pmatrix}$ for D an $s \times s$ diagonal matrix of full rank.

We define positive integers $\mu_1, \ldots, \mu_d$ as follows. Choose $\mu_1, \ldots, \mu_s$ to be the first s prime numbers and let $\mu_{s+1} = \ldots = \mu_d = 1$. Write $Q = (q_{ij})$ and define $\lambda_i = \mu_1^{q_{1i}} \cdots \mu_d^{q_{di}}$ for $i \in \{1, \ldots, d\}$.

Then for all $n_1, \ldots, n_d \in \mathbb{Z}$ we have

$$\begin{aligned}
\lambda_1^{n_1} \cdots \lambda_d^{n_d} = 1 \quad &\Leftrightarrow \quad \mu_1^{(Q\boldsymbol{x})_1} \cdots \mu_d^{(Q\boldsymbol{x})_d} = 1 \\
&\Leftrightarrow \quad (Q\boldsymbol{x})_1 = 0, \ldots, (Q\boldsymbol{x})_s = 0 \\
&\Leftrightarrow \quad BQ\boldsymbol{x} = 0 \quad (\text{since } B = \begin{pmatrix} D & 0 \end{pmatrix}) \\
&\Leftrightarrow \quad PBQ\boldsymbol{x} = 0 \quad (\text{since } P \text{ is invertible}) \\
&\Leftrightarrow \quad A\boldsymbol{x} = 0 \\
&\Leftrightarrow \quad a_1 n_1 + \cdots + a_d n_d = 0\,.
\end{aligned}$$

◀

▶ **Corollary 8.** *Let* D *be a* $d \times d$ *diagonal matrix with algebraic entries. Then there exists a diagonal matrix* D'*, of the same dimension and with rational entries, such that* $\overline{\langle e^D\rangle} = \overline{\langle D'\rangle}$.

Proof. Write $D = \mathrm{diag}(a_1, \ldots, a_d)$ and let rational numbers $\lambda_1, \ldots, \lambda_d$ be chosen as in Proposition 7, i.e., such that $a_1 n_1 + \cdots + a_d n_d = 0$ iff $\lambda_1^{n_1} \cdots \lambda_d^{n_d} = 1$ for all $n_1, \ldots, n_d \in \mathbb{Z}$. Define $D' = \mathrm{diag}(\lambda_1, \ldots, \lambda_d)$. By Lemma 4, the ideal of the variety $\overline{\langle e^D \rangle}$ is generated by $z^n - z^m$ such that $(n_1 - m_1)a_1 + \cdots + (n_d - m_d)a_d = 0$. On the other hand, it follows from [3, Lemma 6] that the ideal of the variety $\overline{\langle D' \rangle}$ is generated by $z^n - z^m$ such that $\lambda_1^{n_1 - m_1} \cdots \lambda_d^{n_d - m_d} = 1$. But by construction the additive relations of a are the same as the multiplicative relations of λ, therefore the ideals are the same. It follows $\overline{\langle e^D \rangle} = \overline{\langle D' \rangle}$. ◀

▶ **Proposition 9.** *Let $A \in \mathbb{Q}^{d \times d}$ be a rational matrix. Then there exists an algebraic matrix B such that $\overline{\langle B \rangle} = \overline{\langle e^A \rangle} = \overline{\{e^{At} : t \in \mathbb{R}\}}$.*

Proof. Let P be an invertible matrix such that $A = P^{-1}(D+N)P$ with $D = \mathrm{diag}(a_1, \ldots, a_d)$ diagonal and N a nilpotent Jordan matrix. By Corollary 8 there exists a rational diagonal matrix D' such that $\overline{\langle D' \rangle} = \overline{\langle e^D \rangle}$. We now define $B := P^{-1}(D'e^N)P$ where $D' = \mathrm{diag}(\lambda_1, \ldots, \lambda_d)$. Note that e^N is a matrix of rational numbers. Then we have:

$$\overline{\langle e^A \rangle} = P^{-1}\overline{\langle e^D e^N \rangle}P = P^{-1}\overline{\langle e^D \rangle \cdot \langle e^N \rangle}P = P^{-1}\overline{\overline{\langle D' \rangle} \cdot \overline{\langle e^N \rangle}}P = P^{-1}\overline{\langle D' e^N \rangle}P \ = \overline{\langle B \rangle}\,. \quad ◀$$

▶ **Proposition 10.** *Given a linear hybrid automaton $\mathcal{A}$, one can compute a finite affine program $\mathcal{A}'$ that has the same algebraic invariants, i.e., $V(\mathcal{A}) = V(\mathcal{A}')$.*

Proof. Suppose that $\mathcal{A} = (Q, A, E, T)$. We define $\mathcal{A}' = (Q, A', E', T)$, where $A'_q = 0$ for all $q \in Q$ and

$$E' = E \cup \{(q, B_q, q) : q \in Q\}\,,$$

with B_q is an algebraic matrix such that $\overline{\langle e^{A_q} \rangle} = \overline{\langle B_q \rangle}$ for all $q \in Q$. The existence of the matrices B_q is guaranteed by Proposition 9. In other words, we obtain $\mathcal{A}'$ from $\mathcal{A}$ by setting the derivative of all variables to 0 in every location and by adding a compensatory selfloop edge to every location.

The reasoning that $V(\mathcal{A}) = V(\mathcal{A}')$ is entirely analogous to that in the proof of Proposition 6. ◀

7 Switching Systems

Finally, we consider the special case of *switching systems*, that is, hybrid systems in which every pair of locations is connected by an edge and such that variables are not updated on discrete edges. It is known that reachability is undecidable even for this restricted class of systems [12]. We show two results: first, we give a simple algorithm to compute strongest algebraic invariants for this class, benefitting from the fact that the strongest algebraic invariant is an irreducible variety (Subsection 7.1); second, we show undecidability of computing a strongest algebraic invariant if guards are introduced (Subsection 7.2).

7.1 Computing Algebraic Invariants

We compute strongest algebraic invariants for switching systems building on Proposition 5.

▶ **Proposition 11.** *Algorithm Semigroup-Closure terminates and outputs the Zariski closure of the sub-semigroup of $\mathrm{GL}_d(\mathbb{C})$ generated by $\{e^{A_1 t}, \ldots, e^{A_k t} : t \geq 0\}$.*

Proof. First note that the effectiveness of Line 2 relies on Proposition 5.

Procedure Semigroup-Closure($A_1, \ldots, A_k$).

input : $A_1, \ldots, A_k \in \mathbb{A}^{d \times d}$
$H := \{I_d\}$
$G_1 := \overline{\{e^{A_1 t} : t \geq 0\}}; \ldots; G_k := \overline{\{e^{A_k t} : t \geq 0\}}$
repeat
  $H_{old} := H$
  for $i \in \{1, \ldots, k\}$ **do**
    $H := \overline{H \cdot G_i}$
until $H_{old} = H$
output : H

Now we argue that H in the algorithm is always an irreducible variety. For this it suffices to show that if $X \subseteq \mathrm{GL}_d(\mathbb{C})$ is an irreducible variety then so is $Y := \overline{X \cdot \overline{\{e^{At} : t \geq 0\}}}$ for any matrix $A \in \mathbb{C}^{d \times d}$. First observe that if X and Z are irreducible sets then so is $\overline{X \cdot Z}$, thus is it enough to show that $G = \overline{S}$ is irreducible, where $S = \{e^{At} : t \geq 0\}$. But S is a semigroup, thus G is a group. This makes G a linear algebraic group, which is therefore irreducible if and only if it is (Zariski-)connected. But G being the closure of S means it is enough to show that S is Zariski-connected, and hence enough to show that it is Euclidean-connected. The latter is trivial since every element of S is path-connected to I_d.

Now a strictly increasing chain $H_1 \subseteq H_2 \subseteq \cdots$ of irreducible sub-varieties of $\mathrm{GL}_d(\mathbb{C})$ has length at most the dimension of $\mathrm{GL}_d(\mathbb{C})$, which is d^2. Thus Algorithm Semigroup-Closure terminates after at most d^2 iterations of the outer loop. It is clear that the terminating value of the algorithm is the Zariski closure of the sub-semigroup of $\mathrm{GL}_d(\mathbb{C})$ generated by $\{e^{A_1 t}, \ldots, e^{A_k t} : t \geq 0\}$. ◀

Now consider a switching system $\mathcal{A} = (Q, A, E, T)$. Let G be the Zariski closure of the semigroup generated by the matrices $e^{A_q t}$, for $q \in Q$ and $t \geq 0$, which can be computed by Proposition 11. Then $V(\mathcal{A}) = \{V_q : q \in Q\}$, the real Zariski closure of the collecting semantics of $\mathcal{A}$, is such that $V_q = \overline{G \cdot X} \cap \mathrm{GL}_d(\mathbb{R})$ for every $q \in Q$, where $X = \bigcup_{q \in Q} T_q$. But then $V(\mathcal{A})$ is computable from G and T.

7.2 Undecidability for Switching Systems with Guards

Finally, we show that computing the strongest algebraic inductive invariant becomes undecidable if we introduce guards to switching systems. Specifically, we consider switching systems with no discrete updates on the variables but with equality guards on the discrete mode changes. For such systems there is a smallest algebraic inductive invariant, which can be obtained as the (location-wise) intersection of the family of all algebraic inductive invariants. However, as we show in this section, this invariant is no longer computable. In other words, in the presence of equality guards the analog of Theorem 3 fails. (We remark also that the discrete mode changes no longer induce Zariski continuous maps on configurations if there are equality guards. Hence we cannot necessarily recover the smallest algebraic inductive invariant as the Zariski closure of the collecting semantics).

▶ **Theorem 12.** *There is no algorithm that computes the strongest algebraic inductive invariant for the class of switching systems with equality guards.*

Proof Sketch (see full proof in appendix). The idea is to simulate a 2-counter machine in such a way that if the machine has an infinite run then the strongest invariant has dimension 2, and otherwise it has dimension 1. Since the dimension of an algebraic set can be effectively determined; this concludes the sketch. ◀

References

1 Hirokazu Anai and Volker Weispfenning. Reach set computations using real quantifier elimination. In *Hybrid Systems: Computation and Control, 4th International Workshop, HSCC 2001, Rome, Italy, March 28-30, 2001, Proceedings*, volume 2034 of *Lecture Notes in Computer Science*, pages 63–76. Springer, 2001.

2 Michele Boreale. Complete algorithms for algebraic strongest postconditions and weakest preconditions in polynomial odes. *Sci. Comput. Program.*, 193:102441, 2020.

3 H. Derksen, E. Jeandel, and P. Koiran. Quantum automata and algebraic groups. *J. Symb. Comput.*, 39(3-4):357–371, 2005.

4 Stephan Falke and Deepak Kapur. When is a formula a loop invariant? In *Logic, Rewriting, and Concurrency - Essays dedicated to José Meseguer on the Occasion of His 65th Birthday*, volume 9200 of *Lecture Notes in Computer Science*, pages 264–286. Springer, 2015.

5 Khalil Ghorbal and André Platzer. Characterizing algebraic invariants by differential radical invariants. In *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, volume 8413 of *Lecture Notes in Computer Science*, pages 279–294. Springer, 2014.

6 Thomas A. Henzinger. The theory of hybrid automata. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27-30, 1996*, pages 278–292. IEEE Computer Society, 1996.

7 Ehud Hrushovski, Joël Ouaknine, Amaury Pouly, and James Worrell. Polynomial invariants for affine programs. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 530–539, 2018.

8 Z. Kincaid, J. Cyphert, J. Breck, and T. W. Reps. Non-linear reasoning for invariant synthesis. *PACMPL*, 2(POPL):54:1–54:33, 2018.

9 Gerardo Lafferriere, George J. Pappas, and Sergio Yovine. Symbolic reachability computation for families of linear vector fields. *J. Symb. Comput.*, 32(3):231–253, 2001. doi:10.1006/jsco.2001.0472.

10 Daniel Liberzon. *Switching in Systems and Control*. Birkhauser/Springer, 2003.

11 D. W. Masser. Linear relations on algebraic groups. In *New Advances in Transcendence Theory*. Cambridge University Press, 1988.

12 Joël Ouaknine, Amaury Pouly, João Sousa Pinto, and James Worrell. Solvability of matrix-exponential equations. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 798–806, 2016.

13 Enric Rodríguez-Carbonell and Deepak Kapur. Automatic generation of polynomial invariants of bounded degree using abstract interpretation. *Sci. Comput. Program.*, 64(1):54–75, 2007. doi:10.1016/j.scico.2006.03.003.

14 Enric Rodríguez-Carbonell and Deepak Kapur. Generating all polynomial invariants in simple loops. *J. Symb. Comput.*, 42(4):443–476, 2007. doi:10.1016/j.jsc.2007.01.002.

15 Enric Rodríguez-Carbonell and Ashish Tiwari. Generating polynomial invariants for hybrid systems. In Manfred Morari and Lothar Thiele, editors, *Hybrid Systems: Computation and Control*, pages 590–605, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

16 Sriram Sankaranarayanan. Automatic invariant generation for hybrid systems using ideal fixed points. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2010, Stockholm, Sweden, April 12-15, 2010*, pages 221–230. ACM, 2010.

17 Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Constructing invariants for hybrid systems. *Formal Methods in System Design*, 32(1):25–55, 2008.

A Time Discretisation

▶ **Proposition 13.** *For a rational matrix $A \in \mathbb{Q}^{d\times d}$ we have*

$$\overline{\{e^{At} : t \in \mathbb{R}\}} = \overline{\langle e^A \rangle}.$$

Proof. Suppose that A is diagonalisable – say $A = U^{-1}DU$ for some invertible matrix U and $D = \mathrm{diag}(a_1, \ldots, a_d)$. It suffices to prove that $\overline{\{e^{Dt} : t \in \mathbb{R}\}} = \overline{\langle e^D \rangle}$.

Consider a multiplicative relationship among the eigenvalues of e^D – say $(e^{a_1})^{n_1} \cdots (e^{a_d})^{n_d} = 1$, where $n_1, \ldots, n_d \in \mathbb{Z}$. Then $a_1 n_1 + \cdots + a_d n_d \in (2\pi i)\mathbb{Z}$. But since $a_1, \ldots, a_d$ are algebraic numbers, we must in fact have $a_d n_d + \cdots + a_d n_d = 0$. It follows that $a_1 t n_1 + \cdots + a_d t n_d = 0$ for all $t \in \mathbb{R}$ and hence $(e^{a_1 t})^{n_1} \cdots (e^{a_d t})^{n_d} = 1$ for all $t \in \mathbb{R}$, i.e., the same multiplicative relation also holds among the eigenvalues of e^{Dt}.

Since the ideal of all polynomial relations satisfied by $\langle e^D \rangle$ is generated by the multiplicative relations satisfied by the eigenvalues of e^D, we have that for any $t \in \mathbb{R}$, matrix e^{Dt} satisfies all polynomial relations satisfied by $\langle e^D \rangle$. This proves the proposition in case A is diagonalisable.

Next, suppose that A is nilpotent. The fact that $\overline{\{e^{At} : t \in \mathbb{R}\}} = \overline{\langle e^A \rangle}$ is already shown in Section 3.3 of Derksen, Jeandel, and Koiran.

The general case can by handled by reduction to the diagonalisable and nilpotent cases as in Proposition 5. ◀

Proposition 13 crucially relies on the fact that π does not appear in the description of A. Indeed, consider the case that $A = \begin{pmatrix} 2\pi i \end{pmatrix} \in \mathbb{C}^{1\times 1}$. Then $\left\{e^{At} : t \in \mathbb{R}\right\} = \{z \in \mathbb{C} : |z| = 1\}$ is the unit circle. However $\left\{e^{An} : n \in \mathbb{Z}\right\} = \{1\}$ is a singleton. Such an example is possible because the map $z \in \mathbb{C} \mapsto e^{Az}$ is not Zariski-continuous in general.

B Examples

We explain how to discretise the hybrid system in Section 2.1 into a corresponding affine program. The first step is to get rid of the continuous dynamics entirely. Let $X = (x, y, v_x, v_y, t, 1)$ be the state, where we add 1 to make it linear, then the continuous behaviour can be rewritten as $\dot{X} = AX$. Intuitively, we can replace the continuous dynamics by infinitely many discrete transitions e^{At} for $t \geqslant 0$. The key observation (Proposition 13) is that we can in fact replace this infinite set with just one matrix, e^A, without changing the smallest algebraic invariant of the system. The strongest algebraic invariant is then obtained by our previous result on affine programs (Theorem 2). In this example, we have

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -g \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \qquad e^A = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & -\frac{1}{2}g \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -g \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

We can now rephrase the discrete transition $X := e^A X$ as $x := x + v_x$, $y := y + v_y - \frac{1}{2}g$, $v_y := v_y - g$ and $t := t + 1$. By construction, this exactly corresponds to a time-discretisation with 1 unit of time. We then obtain the equivalent, for algebraic invariant, affine program depicted in Figure 4a. For instance, one can check that (1) is indeed invariant under this new transition ($x', y', \ldots$ denotes the value after the transition):

$$x' - t'c = x + v_x - (t+1)c = x - tc - c + v_x = 0 \quad \text{since } x = tc \text{ and } v_x = c,$$
$$\begin{aligned} {v'_y}^2 + 2g(y' - h) &= (v_y - g)^2 + 2g(y + v_y - \tfrac{1}{2}g - h) \\ &= v_y^2 + 2g(y - h) = 0. \end{aligned}$$

We next describe how to use the procedure in Section 6 to transform the hybrid system shown in Figure 2b to the equivalent (with respect to algebraic invariants) affine program in Figure 4b. Let $X = (I, I_R, V_R, Q, V_C)$ be the state, then the continuous behaviour in location OPEN (resp. CLOSED) can be rewritten as $\dot{X} = AX$ (resp. $\dot{X} = BX$). We proceed as in the previous example and replace the continuous dynamics by two discrete transitions e^A and e^B. Unfortunately in this case, the resulting matrices

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{RC} & 0 & 0 & 0 \\ 0 & -\frac{1}{C} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{1}{C} & 0 & 0 & 0 \end{bmatrix}, \qquad e^A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & e^{-\frac{1}{RC}} & 0 & 0 & 0 \\ 0 & (e^{-\frac{1}{RC}} - 1)R & 1 & 0 & 0 \\ 0 & (1 - e^{-\frac{1}{RC}})RC & 0 & 1 & 0 \\ 0 & (1 - e^{-\frac{1}{RC}})R & 0 & 0 & 1 \end{bmatrix}$$

have non-algebraic entries in general. Indeed, since RC is algebraic, $e^{-\frac{1}{RC}}$ is never algebraic and this prevents us from computing the strongest algebraic invariant using Theorem 2. We circumvent this issue by constructing another matrix, call it $\hat{A}$, with algebraic coefficients such that if we replace e^A by $\hat{A}$ then the two affine programs have the same algebraic invariants. We show in Section 6 how to compute such a matrix, which is this example produces

$$\hat{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & R & 1 & 0 & 0 \\ 0 & -RC & 0 & 1 & 0 \\ 0 & -R & 0 & 0 & 1 \end{bmatrix}, \qquad \hat{B} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & R & 1 & 0 & 0 \\ 0 & -RC & 0 & 1 & 0 \\ 0 & -R & 0 & 0 & 1 \end{bmatrix}.$$

We then obtain the equivalent, for algebraic invariant, affine program depicted in Figure 4b. For instance, one can check that the OPEN invariant is indeed invariant by $\hat{A}$:

$$\begin{aligned} Q' - CV'_C &= (-RCI_R + Q) - C(-RI_R + V_C) = Q - CV_C = 0 \\ V'_R - RI'_R &= (RI_R + V_R) - R(2I_R) = V_R - RI_R = 0 \\ V'_R + V'_C &= (RI_R + V_R) + (-RI_R + V_C) = V_R + V_C = 0. \end{aligned}$$

C Proof of Theorem 12

Recall that a non-deterministic 2-counter machine M consists of two counters C and D and a list of n instructions. Each instruction increments one of the counters, decrements one of the counters, or tests one of the counters for zero. After executing a counter update or a successful test, the machine proceeds nondeterministically to one of two specified instructions. The machine halts if it executes a test instruction whose condition is false. Given an instruction i, if j is one of the two possible successors of i then we call the pair (i, j) a *transition* of M. Initially M starts with both counters zero and instruction 1 is the first to be executed. A configuration of M is a triple consisting of the current instruction and the current counter values. The problem of whether such a machine M can reach infinitely many configurations from its initial configuration is undecidable.

Corresponding to such a 2-counter machine M we define a linear hybrid automaton $\mathcal{A} = (Q, A, E, q)$ in dimension 3. We think of $\mathcal{A}$ as having continuous variables c, d, t, where c and d respectively correspond to the counters of M. Each variable has constant derivative in each location, which is zero unless otherwise specified. For each instruction i of M we postulate a location q_i of $\mathcal{A}$ and for each transition (i, j) of M we postulate a location $q_{i,j}$ of $\mathcal{A}$. Variable t has slope 1 in each location q_i and slope -1 in each location $q_{i,j}$. For every transition (i, j) of M, automaton $\mathcal{A}$ has an edge from q_i to $q_{i,j}$ with guard $t = 1$ and an edge from $q_{i,j}$ to q_j with guard $t = 0$. Intuitively, if an execution of $\mathcal{A}$ correctly simulates a run of M then $\mathcal{A}$ spends one time unit in each location, alternating between locations q_i that correspond to instructions of M and locations $q_{i,j}$ that correspond to transitions of M.

Suppose that the i-th instruction of M performs an incrementation $C := C + 1$. Then variable C has slope 1 in location q_i. Likewise if the i-th instruction if $C := C - 1$, then variable c has slope -1 in location q_i. If the i-th instruction of M is the zero test $C = 0$, then the edge from location q_i to $q_{i,j}$ in A has guard $c = 0$. There are corresponding constructions for counter operations and tests on counter D.

This completes the description of $\mathcal{A}$. We now claim that:

1. If M can only reach finitely many configurations from the initial configuration then $V(\mathcal{A}) = \{V_q : q \in Q\}$, the Zariski closure of the collecting semantics, is an inductive invariant that has dimension one.
2. If infinitely many configurations are reachable from the initial configuration of M then the smallest inductive invariant has dimension strictly greater than one.

To prove the claim, note that for each reachable configuration (i, z_1, z_2) of M, the collecting semantics $\Phi(\mathcal{A})_{q_i}$ contains a half-line L containing the point $(z_1, z_2, 0)$, whose direction is determined by the slopes of the respective variables of $\mathcal{A}$ in location q_i. The Zariski closure of L is the affine hull of L, i.e., the corresponding full line containing L. Crucially, the points added to L to obtain the full line are all predecessors of L under the flow relation of $\mathcal{A}$. In particular the Zariski closure is inductive: it remains closed under the transition relation of $\mathcal{A}$. In particular, if M can only reach finitely many configurations, then the Zariski closure of the collecting semantics consists of finitely many lines in each location (and so has dimension one) and is moreover an inductive invariant.

Suppose M can reach infinitely many configurations. Since any algebraic inductive invariant must in particular contain the Zariski closure of the collecting semantics, it follows that any algebraic inductive invariant must contain infinitely many lines in some location and thus must have dimension strictly greater than one.

Since the dimension of an algebraic set can be effectively determined, we conclude that it is not possible to compute the smallest algebraic invariant of a linear hybrid automaton with equality guards (even with a no discrete updates of the variables).

A General Approach to Derive Uncontrolled Reversible Semantics

Ivan Lanese
Focus Team, University of Bologna/Inria, Bologna, Italy
ivan.lanese@gmail.com

Doriana Medić
Focus Team, University of Bologna/Inria, Sophia Antipolis, France
doriana.medic@gmail.com

Abstract

Reversible computing is a paradigm where programs can execute backward as well as in the usual forward direction. Reversible computing is attracting interest due to its applications in areas as different as biochemical modelling, simulation, robotics and debugging, among others. In concurrent systems the main notion of reversible computing is called *causal-consistent reversibility*, and it allows one to undo an action if and only if its consequences, if any, have already been undone.

This paper presents a general and automatic technique to define a causal-consistent reversible extension for given forward models. We support models defined using a reduction semantics in a specific format and consider a causality relation based on resources consumed and produced. The considered format is general enough to fit many formalisms studied in the literature on causal-consistent reversibility, notably Higher-Order π-calculus and Core Erlang, an intermediate language in the Erlang compilation. Reversible extensions of these models in the literature are ad hoc, while we build them using the same general technique. This also allows us to show in a uniform way that a number of relevant properties, causal-consistency in particular, hold in the reversible extensions we build. Our technique also allows us to go beyond the reversible models in the literature: we cover a larger fragment of Core Erlang, including remote error handling based on links, which has never been considered in the reversibility literature.

2012 ACM Subject Classification Theory of computation → Concurrency; Computing methodologies → Concurrent computing methodologies

Keywords and phrases Reversible computing, causality, process calculi, Erlang

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.33

Related Version A full version of the paper is available at `https://hal.archives-ouvertes.fr/hal-02902204`.

Funding This work has been partially supported by French ANR project DCore ANR-18-CE25-0007. *Ivan Lanese*: also partially supported by INdAM as member of GNCS (Gruppo Nazionale per il Calcolo Scientifico).

Acknowledgements The authors thank the reviewers for their helpful comments and suggestions.

1 Introduction

Reversible computing considers systems that can compute backward, recovering past states, as well as forward. The studies on reversible computing gained in popularity in the 60's, thanks to the observation that only irreversible actions need to produce heat [21]. Beyond obtaining computing machinery with low heat dissipation, reversible computing found its application in a wide range of fields, from biochemical modelling [6, 12, 19, 41] to simulation [8], robotics [34], programming [35, 46, 29] and program debugging [5, 16, 36, 28]. The main objective of the theoretical computer science community in this research area has been to provide a foundational understanding of reversibility. Nowadays, in the literature, there is a

number of formalisms describing different approaches to reversibility with the purpose to better understand its properties and characteristics, e.g. reversible computation in process algebras [10, 42, 26], Petri Nets [40, 37], event structures [47], logic circuits [14], etc.

In a sequential system, backward computation is obtained by undoing forward actions in reverse order of execution, starting from the last one. Undoing a forward action can be seen as a backward action. In a concurrent setting, where many processes are running at the same time, identifying the last action is not an easy task, and may sometimes be impossible. Therefore, alternative approaches have been considered. Here we consider the *causal-consistent approach* [10, 42, 27], which focuses on the causality relations between actions to decide which actions can be undone. Consequently, while designing a reversible model following the causal-consistent approach, one needs to take care of storing information on the past of the system, to be able to recover past states, but also causality information, to know which forward steps can be undone at a given moment. In order to show that a reversible model follows the causal-consistent approach, a number of properties need to be proved [10]. The most relevant are the Loop Lemma, showing that each action can be undone, the Square Lemma, showing that the chosen notion of causality is compatible with the semantics, and Causal Consistency, showing that the correct information is stored. More recently [32], Causal Safety and Causal Liveness have also been proposed, stating that an action can be undone if and only if its consequences, if any, are undone beforehand.

The aim of this paper is to explore how to mechanically obtain a causal-consistent reversible extension of a given forward-only model. This is in sharp contrast with most of the reversible models in the concurrency literature, which have been defined manually. An advantage of building the reversible model in this way is that the properties mentioned before are satisfied by construction. The only other work we are aware of providing an automatic technique is [42], which considers process calculi defined in a specific SOS format [43]. Differently from [42], we focus on forward systems defined using a reduction semantics (Section 2.1). While this is more limited since it does not consider open systems, our approach can deal with systems that do not fit the model in [42]. This is the case for both our case studies, namely higher-order π [44] and Core Erlang [7].

Given a forward-only system, we aim at building its *uncontrolled* [27] causal-consistent reversible extension. Here with uncontrolled we mean that at any moment both forward actions and backward actions are possible, and there is no policy on which action to prefer. Uncontrolled semantics is the basis for a reversible model, on top of which control policies selecting the actions to be done or undone can be added [11, 24, 2, 25].

Our approach works in two main steps. First, we attach a unique identifier, called *key*, to every entity (process, messages, etc.) of the forward system, and then we enrich the model with *memories*, where past information is stored (Section 2.2). After defining our method, we show that the reversible models built using it satisfy the properties of causal-consistent reversible models discussed above (Section 3). We prove them using a novel approach [32], which consists in showing that the system satisfies a few basic axioms.

To show the generality of our method, we apply it to two case studies: higher-order π-calculus [44] (used as a running example) and Core Erlang [7] (Section 4.2). After obtaining the corresponding reversible models, we show that, while syntactically different, they have the same behaviour as the ones in the literature [26, 30]. We also show how our approach can be used to go further than what it is in the literature. As an example, we extend reversible Core Erlang to also support Core Erlang constructs for remote error handling based on links (Section 4.3). Such an extension has never been considered in the reversibility literature.

Due to space limitations, further details are in the Appendix while proofs are in [23].

2 Our Approach

In this section we formally introduce our approach. We first define the constraints that the forward-only model we take in input needs to satisfy, and then we describe how to derive the syntax and semantics of the corresponding causal-consistent reversible model.

To give a better intuition about our approach, we will use as a running example its instantiation on the asynchronous Higher-Order π-calculus [44].

2.1 Forward model

We assume a forward model equipped with a reduction semantics. The syntax of the forward model is structured in two levels. The lower level is composed by entities, e.g., processes, messages and resources, ranged over by P, Q. There are no restrictions on the syntax of the lower level. The upper level needs to follow the structure below:

$$N ::= P \mid op_n(N_1, \ldots, N_n) \mid \mathbf{0}$$

Essentially, a system is obtained by composing entities using composition operators op_n, where n is the operator arity. Among the composition operators we assume a binary parallel composition operator, thus $N_1 \mid N_2$ represents the parallel composition of two systems. Additionally, $\mathbf{0}$ represents the empty system. Notably, $\mathbf{0}$ is not an entity.

Below we recall the syntax of HOπ-calculus and show how it fits in our framework.

▶ **Example 1.** The classical syntax of HOπ-calculus [44] is as follows:

$$P, Q ::= a\langle P\rangle \mid a(X) \triangleright P \mid (P \mid Q) \mid \nu a\,(P) \mid X \mid \mathbf{0}$$

Process variables are represented with X and channel names with a, b, c. Process $a\langle P\rangle$ sends message P over channel a while $a(X) \triangleright P$ denotes a process which receives a message on channel a and replaces it for X inside P. There is no continuation after output since the calculus is asynchronous. We denote parallel composition with $P \mid Q$ and its neutral element with $\mathbf{0}$. Restriction of name a inside P is written $\nu a\,(P)$. The binders are $\nu a\,(P)$ and $a(X) \triangleright P$, where the scope of name a and variable X is process P. We denote the set of free names of process P with $\mathtt{fn}(P)$.

In order to fit our framework we need to separate entities from systems. In this case, an entity is any HOπ process whose topmost operator is neither a parallel composition nor a restriction nor $\mathbf{0}$. The syntax of systems is thus as follows

$$N := P \mid (N_1 \mid N_2) \mid \nu a\,(N) \mid \mathbf{0}$$

where parallel composition and $\mathbf{0}$ are the operators required by our framework and restriction is an infinite family of unary operators with one instance for each name a. ⌟

Thanks to the syntax above, a generic system can be represented as a term $T[P_1, \ldots, P_n]$, where $T[\bullet_1, \ldots, \bullet_n]$ is a context with n numbered holes built from composition operators, possibly including parallel composition, and $\mathbf{0}$. The term $T[P_1, \ldots, P_n]$ is obtained by replacing $\bullet_i$ with P_i for each $i \in \{1, \ldots, n\}$.

We complement our syntax with a structural congruence, specified by axioms of the form

$$T[P_1, \ldots, P_n] \equiv T'[P'_1, \ldots, P'_n]$$

and closed under contexts, reflexivity, symmetry and transitivity. As can be seen from the rule format, structural congruence cannot change the number of entities in a term. Also, it is understood that P_i and P'_i refer to the same entity, which can however evolve while

$$\text{(Scm-Act)}\ \frac{}{P_1 \mid \ldots \mid P_n \rightarrowtail T[Q_1, \ldots, Q_m]} \qquad \text{(Eqv)}\ \frac{N \equiv N' \quad N \rightarrowtail N_1 \quad N_1 \equiv N_1'}{N' \rightarrowtail N_1'}$$

$$\text{(Scm-Opn)}\ \frac{N_i \rightarrowtail N_i'}{op_n(N_0, \ldots, N_i, \ldots, N_n) \rightarrowtail op_n(N_0, \ldots, N_i', \ldots, N_n)} \qquad \text{(Par)}\ \frac{N \rightarrowtail N'}{N \mid N_1 \rightarrowtail N' \mid N_1}$$

Figure 1 Forward rules structure; Scm- rules are schemas.

preserving its identity. This assumption will become clearer later on, when we introduce keys (to track the identity) and the causality relation. We assume structural rules ensuring that parallel composition is associative, commutative, and has $\mathbf{0}$ as neutral element.

We illustrate below that the structural congruence of HOπ satisfies the requirements.

▶ **Example 2.** Sample HOπ structural rules are as follows, the full structural congruence is in Appendix A.

$$\text{(Alpha)}\ \ \nu a\, P \equiv \nu b\, P\{b/a\} \quad \text{if } b \notin \mathtt{fn}(P) \qquad \text{(ParC)}\ \ P \mid Q \equiv Q \mid P$$
$$\text{(ResF)}\ \ (\nu a\, P) \mid Q \equiv \nu a\, (P \mid Q) \quad \text{if } a \notin \mathtt{fn}(Q)$$

Rule (Alpha) is α-conversion. Rule (Alpha) is seen in our framework as an infinite family of rules (and the same for rule (ResF) for scope extrusion), for each a, P and b satisfying the side condition. Hence, no side condition is needed in the instance. Note that P on the left-hand side and $P\{b/a\}$ on the right-hand side are understood to be the same entity. Rule (ParC) establishes commutativity of parallel composition as required. It exploits contexts of the form $\bullet_1 \mid \bullet_2$ and $\bullet_2 \mid \bullet_1$. ⌟

The reduction semantics of the forward model needs to have the format described in Figure 1, which includes two rules ((Par) and (Eqv)), which need to belong to the semantics, and two schemas ((Scm-Act) and (Scm-Opn)). The semantics can contain any number of instances of the schemas (possibly an infinite number), obtained by replacing all placeholders with terms of the corresponding category (e.g., P_1 with an entity, T with a context, and so on). One may notice that rule (Par) is an instance of schema (Scm-Opn): this means that such an instance is required. Anyway, being an instance, we do not need to deal with it explicitly in the following.

Rule schema (Scm-Act) allows one to specify interactions between entities. It is understood that such an interaction consumes the entities $P_1, \ldots, P_n$ and produces the entities $Q_1, \ldots, Q_m$. This intuition will be captured by keys and the causality relation. The created entities are composed in a term $T[Q_1, \ldots, Q_m]$, where T is a context built from composition operators. Rules in this schema, together with rule (Par), allowing a system to execute inside a parallel composition, define the behaviour of parallel composition. The behaviour of other operators is described by rule schema (Scm-Opn). Notably, this schema allows a single entity to execute at each step. Rule (Eqv) allows one to exploit structural congruence.

We see below how the rule for communication of HOπ fits the format given in Figure 1. The full semantics of HOπ and the explanation of how it fits the format is in Appendix A.

▶ **Example 3.** The communication rule (Act) of HOπ, where process Q is received and bound to variable X, is defined as:

$$(\text{Act})\ \frac{}{a\langle Q\rangle \mid a(X) \triangleright P \rightarrowtail P\{Q/X\}}$$

Rule (Act) can be seen as an infinite family of rules fitting schema (Scm-Act). Notice that the number of entities in the resulting process may vary, e.g., in:

$$a\langle \nu b\,(b\langle P\rangle \mid b(Y) \triangleright Y \mid c\langle Q\rangle)\rangle \mid a(X) \triangleright X \rightarrowtail \nu b\,(b\langle P\rangle \mid b(Y) \triangleright Y \mid c\langle Q\rangle)$$

the resulting process has three entities $b\langle P\rangle$, $b(Y) \triangleright Y$ and $c\langle Q\rangle$, composed using a context $T = \nu b\,(\bullet_1 \mid \bullet_2 \mid \bullet_3)$. ⌟

▶ **Example 4.** The CCS reduction $\overline{a}.P + Q | !a.R \rightarrowtail_{CCS} P | !a.R | R$, where the output $\overline{a}$ synchronises with the replicated input $!a$ and Q is discarded, can be seen as an instance of schema (Scm-Act) as well. Indeed, the two parallel entities $\overline{a}.P + Q$ and $!a.R$ interact to produce the three entities P, $!a.R$ and R on the right-hand side (assuming P and R to be single entities). ⌟

2.2 Definition of the Reversible System

In order to define the causal-consistent reversible extension of a given system, one first needs to extend the forward semantics so to keep track of past states. This information will be used by the backward semantics. In particular, we use unique *keys* to distinguish identical entities which have different history, and *memories* to recall parts of the system which have been changed by a computational step. More in detail, each entity of a system is labelled with its unique key. Also, each step of the system produces a memory allowing one to undo it. We refer to systems extended with keys and memories as *configurations*.

▶ **Definition 5.** *The syntax of configurations R is defined by the following grammar:*

$$R ::= k : P \mid op_n(R_1, \ldots, R_n) \mid \boldsymbol{0} \mid [R\,;C] \qquad C ::= T[k_1 : \bullet_1, \ldots, k_m : \bullet_m]$$

where operators op_n are the same as in the forward system and T is a context composed of operators op_n and $\boldsymbol{0}$. Also, $\bullet_i$ are numbered holes, to be filled by the processes with keys k_i.

Intuitively, a memory $\mu = [R\,;C]$ is composed of the configuration R which gave rise to the forward step and the context C of the configuration resulting from it.

▶ **Example 6.** The syntax of the reversible HOπ-calculus is defined as:

$$R ::= k : P \mid (R_1 \mid R_2) \mid \nu a\,(R) \mid \mathbf{0} \mid [R\,;C]$$

where entities P are as in the underlying calculus and a unique key k is attached to each of them. Note that now parallel composition and restriction operators are applied to configurations. Finally, memories are also part of the syntax. ⌟

We now define the structural congruence and the forward and backward operational semantics for the reversible system. As in the original model, we can represent a reversible system as $T[k_1 : P_1, \ldots, k_n : P_n]$, where T is a context built from operators op_n and $\mathbf{0}$. The main difference w.r.t. the original calculus is that now each entity is labelled with its key. We have one structural rule for each structural rule of the original semantics, with the same context T, but now entities are labelled with keys, and keys on both sides are the same.

$$T[k_1 : P_1, \ldots, k_n : P_n] \equiv T'[k_1 : P'_1, \ldots, k_n : P'_n]$$

We define below the function $\texttt{key}(\cdot)$ that computes the set of keys in a configuration R:

$$\text{(F-Scm-Act)}\ \frac{P_1 \mid \ldots \mid P_n \rightarrowtail T[Q_1, \ldots, Q_m] \qquad j_1, \ldots, j_m \text{ are fresh keys}}{k_1 : P_1 \mid \ldots \mid k_n : P_n \twoheadrightarrow T[j_1 : Q_1, \ldots, j_m : Q_m] \mid [k_1 : P_1 \mid \ldots \mid k_n : P_n \,; T[j_1 : \bullet_1, \ldots, j_m : \bullet_m]]}$$

$$\text{(F-Scm-Opn)}\ \frac{R_i \twoheadrightarrow R_i' \quad (\mathtt{key}(R_i') \setminus \mathtt{key}(R_i)) \cap (\mathtt{key}(R_0, \ldots, R_{i-1}, R_{i+1}, \ldots, R_n) = \emptyset}{op_n(R_0, \ldots, R_i, \ldots, R_n) \twoheadrightarrow op_n(R_0, \ldots, R_i', \ldots, R_n)}$$

$$\text{(F-Eqv)}\ \frac{R \equiv R' \quad R \twoheadrightarrow R_1 \quad R_1 \equiv R_1'}{R' \twoheadrightarrow R_1'}$$

Figure 2 Forward rules of the uncontrolled reversible semantics.

$$\text{(B-Scm-Act)}\ \frac{\mu = [k_1 : P_1 \mid \ldots \mid k_n : P_n \,; T[j_1 : \bullet_1, \ldots, j_m : \bullet_m]]}{T[j_1 : Q_1, \ldots, j_m : Q_m] \mid \mu \rightsquigarrow k_1 : P_1 \mid \ldots \mid k_n : P_n}$$

$$\text{(B-Scm-Opn)}\ \frac{R_i' \rightsquigarrow R_i}{op_n(R_0, \ldots, R_i', \ldots, R_n) \rightsquigarrow op_n(R_0, \ldots, R_i, \ldots, R_n)} \qquad \text{(B-Eqv)}\ \frac{R \equiv R' \quad R \rightsquigarrow R_1 \quad R_1 \equiv R_1'}{R' \rightsquigarrow R_1'}$$

Figure 3 Backward rules of the uncontrolled reversible semantics.

▶ **Definition 7.** *The* set of keys *of a configuration R, written as* $\mathtt{key}(R)$*, is defined as:*

$$\mathtt{key}(k : P) = \{k\} \qquad \mathtt{key}(op_n(R_1, \ldots, R_n)) = \mathtt{key}(R_1) \cup \ldots \cup \mathtt{key}(R_n)$$
$$\mathtt{key}(\boldsymbol{0}) = \emptyset \qquad \mathtt{key}([R \,; C]) = \mathtt{key}(R) \cup \mathtt{key}(C)$$

The forward rules of the uncontrolled reversible semantics are in Figure 2. For schemas (F-Scm-Act) and (F-Scm-Opn) we have one instance for each instance of the corresponding schema in the original semantics. For schema (F-Scm-Act) the main difference w.r.t. the original schema is that entities are labelled with keys and a memory stores information on the performed step. More precisely, entities $Q_1, \ldots Q_m$ on the right-hand side have fresh keys $j_1, \ldots, j_m$. Also, the left configuration $R = k_1 : P_1 \mid \ldots \mid k_n : P_n$ is saved in a memory $\mu = [R; C]$ together with the context $C = T[j_1 : \bullet_1, \ldots, j_m : \bullet_m]$ of the resulting configuration. In this way, the structure of the obtained system and the newly generated keys are recorded. They will be needed to perform the corresponding backward step. As far as the schema (F-Scm-Opn) is concerned, the only novelty is the side condition ensuring that keys introduced during the step are fresh for the whole system. The structural congruence rule (F-Eqv) is the same as in the original semantics (but structural congruence preserves keys).

The backward rules, depicted in Figure 3, are symmetric w.r.t. the forward ones. With rule schema (B-Scm-Act) the forward action that produced term $T[j_1 : Q_1, \ldots, j_m : Q_m]$ is undone. The past state of the system $k_1 : P_1 \mid \ldots \mid k_n : P_n$ is restored from the memory μ. The context $C = T[j_1 : \bullet_1, \ldots, j_m : \bullet_m]$ inside μ additionally ensures that all entities produced by the forward action, together with the term composing them, are available in the configuration and are consumed by the backward step.

▶ **Definition 8** (Uncontrolled reversible semantics)**.** *The reduction relation $\twoheadrightarrow$ (resp. $\rightsquigarrow$), defined as the smallest relation closed under the forward (resp. backward) rules, defines the forward (resp. backward) reversible semantics. The semantics, denoted by $\rightarrow$, is the union of the forward semantics $\twoheadrightarrow$ and the backward semantics $\rightsquigarrow$ (i.e. $\rightarrow = \twoheadrightarrow \cup \rightsquigarrow$).*

▶ **Example 9.** Below, we give the communication rule of the forward and backward reversible semantics for the HOπ-calculus. The other rules can be found in Appendix A.

$$\text{(F-Act)}\ \frac{a\langle P\rangle \mid a(X)\triangleright P' \rightarrowtail P'\{P/X\} \qquad j_1,\dots j_m \text{ are fresh keys} \quad \text{and} \quad P'\{P/X\} = T[Q_1,\dots,Q_m]}{k_1 : a\langle P\rangle \mid k_2 : a(X)\triangleright P' \twoheadrightarrow T[j_1 : Q_1,\dots,j_m : Q_m] \mid [k_1 : a\langle P\rangle \mid k_2 : a(X)\triangleright P' \,;T[j_1 : \bullet_1,\dots,j_m : \bullet_m]]}$$

$$\text{(B-Act)}\ \frac{\mu = [k_1 : a\langle P\rangle \mid k_2 : a(X)\triangleright P' \,;T[j_1 : \bullet_1,\dots,j_m : \bullet_m]]}{T[j_1 : Q_1,\dots,j_m : Q_m] \mid \mu \rightsquigarrow k_1 : a\langle P\rangle \mid k_2 : a(X)\triangleright P'}$$

Using rule schema (F-Act) a configuration can execute a forward step in which the memory $\mu = [k_1 : a\langle P\rangle \mid k_2 : a(X)\triangleright P' \,;T[j_1 : \bullet_1,\dots,j_m : \bullet_m]]$, recording the prior state of the configuration and the context with the new fresh keys, is generated. After the communication we obtain the system $P'\{P/X\}$ which we can rewrite as a term $T[Q_1,\dots,Q_m]$, where $Q_1,\dots,Q_m$ are entities. Using rule (B-Act) the configuration can undo the forward step which produced the memory μ. The prior state of the system is restored from it. ⌟

3 Properties

In this section, we show that the reversible semantics defined using the approach in the previous section satisfies a number of properties expected from a causal-consistent reversible semantics. In particular, the reversible semantics is a conservative extension of the forward semantics, and it is causally consistent [10].

Since our syntax allows for a number of ill-formed terms, as commonly done, in the following we restrict the attention to reachable configurations, defined below.

▶ **Definition 10** (Initial and reachable configuration). *A configuration R is* initial *if it does not contain memories and all keys are distinct. A configuration R is* reachable *if it can be derived from an initial configuration by applying the rules in Figures 2 and 3.*

Correspondence between reversible and original semantics

In this section we prove that the forward reversible semantics is a conservative extension of the original semantics. To this end, we first define the erasing function φ that given a configuration R, by deleting histories and keys, generates a forward-only system N.

▶ **Definition 11** (Erasing function). *The function $\varphi : \mathcal{R} \to \mathcal{N}$, where $\mathcal{R}$ and $\mathcal{N}$ denote respectively the sets of configurations and of original systems, is inductively defined as follows:*

$$\varphi(k : P) = P \quad \varphi([R\,;C]) = \boldsymbol{0} \quad \varphi(\boldsymbol{0}) = \boldsymbol{0} \quad \varphi(op_n(R_1,\dots,R_n)) = op_n(\varphi(R_1),\dots,\varphi(R_n))$$

Now, we can show that the forward semantics of a configuration R and the semantics of its projection on the forward system $\varphi(R)$ are strong bisimilar (Definition 28 in Appendix A).

▶ **Theorem 12.** *For each configuration R, its forward semantics and the semantics of $\varphi(R)$ are strong bisimilar.*

3.1 Concurrency and Causal Consistency

In order to prove that the defined reversible semantics is indeed causal-consistent we need to define a causality relation on our systems. We define it directly on reversible systems, for two reasons. First, keys and memories help in this respect. Second, in a reversible system the concurrency relation induces a causality relation (see [30, Def. 11 and Lemma 6]).

We extend the reduction semantics to a notion of transitions, which carry in the label information on the used resources, in form of the memory involved in the transition. Formally, we define transitions t of a system R as $t : R \xrightarrow{\mu} R'$, where μ is the memory created by the transition, if it is forward, or consumed by it, if it is backward. There, R is the *source* while R' is the *target* of the transitions t. Two transitions are *coinitial* (resp. *cofinal*) if they have the same source (resp. target), and *composable* if the target of the former is the source of the latter. A *derivation* d from the source R to the target R', written as $d : R \to^* R'$, is a sequence of composable transitions. A zero steps derivation is written ϵ.

The concurrency relation between transitions states that two coinitial transitions are concurrent if they do not share entities. Formally:

▶ **Definition 13** (Concurrent transitions). *Two coinitial transitions* $t' : R \xrightarrow{\mu'} R'$ *and* $t'' : R \xrightarrow{\mu''} R''$ *are* concurrent*, written* $t' \frown_c t''$*, if* $\mathtt{key}(\mu') \cap \mathtt{key}(\mu'') = \emptyset$*. Coinitial transitions which are not concurrent are in* conflict*.*

Notably, our notion of concurrency is extracted from the operational semantics (via its extension with keys and memories), hence it can be obtained also for those models where no notion of concurrency exists in the literature, like most mainstream programming languages.

Having fixed a notion of concurrency, we can proceed to show the Causal Consistency of the reversible semantics. To prove it, we use the recent axiomatic approach given in [32], which allows one to show a number of properties relevant for reversible calculi, such as the Parabolic Lemma (PL) and Causal Consistency (CC), by just proving a few basic axioms. The advantage is that proving the axioms is simpler than proving the results directly. Moreover, [32] introduces two new properties: Causal Safety (CS) stating that an action cannot be reversed until all actions caused by it have been reversed; and Causal Liveness (CL) saying that actions do not necessarily need to be reversed in the exact inverse order of the forward execution, but can be reversed in any order consistent with CS.

In the following we give the axioms and auxiliary definitions required by the framework of [32] necessary to show Causal Consistency, Safety and Liveness.

First, we re-formulate our framework as a Labelled Transition System with Independence (LTSI, see also [45]) $(\mathcal{R}, \mathcal{L}, \to, \iota)$, where $\mathcal{R}$ is a set of systems, $\mathcal{L}$ is the set of action labels, $\to \subset \mathcal{R} \times \mathcal{L} \times \mathcal{R}$ is a transition relation and ι is the independence relation, namely an irreflexive symmetric binary relation on transitions. In our case, $\mathcal{R}$ is the set of configurations and $\mathcal{L}$ the set of labels of our transitions. The latter include both forward and backward transitions. Also, the notion of independence is defined on coinitial transitions and it coincides with the notion of concurrency, namely $\iota = \frown_c$. A key property required by the framework in [32] is that each action is reversible, as shown by the following result.

▶ **Lemma 14** (Loop Lemma). *For every reachable configuration* R *and forward transition* $t : R \xrightarrow{\mu} R'$*, there exists a backward transition* $t^\bullet : R' \overset{\mu}{\rightsquigarrow} R$ *and vice versa.*

From now on we denote with $t^\bullet$ the reverse of t. The basic properties required to show causal consistency are as follows.

▶ **Definition 15** (Basic axioms).

Square Property (SP): *if* $t_1 : R \xrightarrow{\mu_1} R'$ *and* $t_2 : R \xrightarrow{\mu_2} R''$ *are two coinitial independent transitions, there exist two cofinal transitions* $t_2/t_1 : R' \xrightarrow{\mu_2} R'''$ *and* $t_1/t_2 : R'' \xrightarrow{\mu_1} R'''$*.*

Backward transitions are independent (BTI): *any two coinitial backward transitions* $t_1 : R \overset{\mu_1}{\rightsquigarrow} R_1$ *and* $t_2 : R \overset{\mu_2}{\rightsquigarrow} R_2$ *where* $t_1 \neq t_2$ *are independent.*

Well-foundedness (WF): *there is no infinite backward computation.*

SP states that independent transitions can be executed in any order. We follow the standard notation and write t_2/t_1 for the residual of t_2 after t_1. Coinitial backward transitions are always independent by BTI. WF ensures that each system has a finite past.

To state Causal Consistency, we first define Causal Equivalence [10], an equivalence relation between derivations which stipulates that independent transitions can be swapped while pairs of reverse transitions can be removed from the derivation. The definition is well-posed if the LTSI satisfies the Square Property.

▶ **Definition 16** (Causal equivalence). *Causal equivalence, $\sim$, is the least equivalence relation between derivations closed under composition satisfying*

$$t_1; t_2/t_1 \sim t_2; t_1/t_2 \qquad t; t^\bullet \sim \epsilon$$

We now define two properties needed for Causal Safety and Causal Liveness, namely Coinitial propagation of independence (CPI) and Coinitial independence respects events (CIRE).

▶ **Definition 17** (Coinitial propagation of independence (CPI)). *If whenever $t_1 : R \xrightarrow{\mu_1} R'$, $t_2 : R \xrightarrow{\mu_2} R''$, $t_2' : R' \xrightarrow{\mu_2} R'''$ and $t_1' : R'' \xrightarrow{\mu_1} R'''$ with $t_1 \smile_c t_2$, then we have $t_2' \smile_c t_1^\bullet$.*

We introduce the notion of *event*, needed to state (CIRE), and define independence (concurrency in our case) on them.

▶ **Definition 18** (Events). *Let $(\mathcal{R}, \mathcal{L}, \rightarrow, \smile_c)$ be a LTSI satisfying SP,BTI,WF and CPI. Let $\approx$ be the smallest equivalence relation satisfying: if $t_1 : R \xrightarrow{\mu_1} R'$, $t_2 : R \xrightarrow{\mu_2} R''$, $t_2' : R' \xrightarrow{\mu_2} R'''$ and $t_1' : R'' \xrightarrow{\mu_1} R'''$ and $t_1 \smile_c t_2$, then $t_1 \approx t_1'$.*
The equivalence classes of forward transitions $R \overset{\mu}{\twoheadrightarrow} R'$, written $[R, \mu, R']$, are the events. The equivalence classes of reverse transitions $R \overset{\mu}{\hookrightarrow} R'$, $[R, \mu^\bullet, R']$, are the reverse events. A labelling function l from $\rightarrow / \approx$ to $\mathcal{L}$ is defined by settings $l([R, \mu, R']) = l([R, \mu^\bullet, R']) = \mu$. Events e_1, e_2 are (coinitially) independent, written $e_1 \mathtt{ci}\ e_2$, iff there are coinitial transitions t_1 and t_2 such that $[t_1] = e_1$, $[t_2] = e_2$ and $t_1 \smile_c t_2$.

▶ **Definition 19** (Coinitial independence respects events (CIRE)). *If $[t_1]\ \mathtt{ci}\ [t_2]$ and t_1 and t_2 are coinitial, then $t_1 \smile_c t_2$.*

▶ **Proposition 20.** *Axioms SP, BTI, WF, CPI and CIRE hold for each instance of our framework.*

Given that our reversible semantics satisfies all the axioms, thanks to [32], all instances of our framework satisfy the Parabolic Lemma, Causal Consistency, Causal Safety and Causal Liveness, defined below.

▶ **Definition 21** (Parabolic Lemma (PL)). *Given a derivation $d : R \rightarrow^* R'$, there exists a configuration R'' such that $d' : R \rightsquigarrow^* R'' \twoheadrightarrow^* R'$ and $d \sim d'$. Also, d' is not longer than d.*

▶ **Definition 22** (Causal Consistency (CC)). *Given two coinitial derivations d_1 and d_2, $d_1 \sim d_2$ if and only if d_1 and d_2 are cofinal.*

Below, we state Causal Safety and Causal Liveness. We present them in a slightly rephrased and more intuitive form w.r.t. [32], whose presentation is however more formal.

▶ **Definition 23** (Causal Safety (CS) and Causal Liveness (CL)).
Let $\mathtt{L} = (\mathcal{R}, \mathcal{L}, \rightarrow, \smile_c)$ be a LTSI satisfying SP,BTI,WF and CPI. Take a derivation $R \xrightarrow{\mu} R' \xrightarrow{\rho}{}^* R''$. Transition $R \xrightarrow{\mu} R'$ can be undone in R'', that is there is a transition $R_1 \xrightarrow{\mu} R''$ with $(R, \mu, R') \approx (R_1, \mu, R'')$, if (CL) and only if (CS) $R \xrightarrow{\mu} R'$ is concurrent to all transitions $R' \xrightarrow{\rho}{}^* R''$ which are not undone in $R' \xrightarrow{\rho}{}^* R''$.

4 Case Studies

In this section we apply our approach to two relevant case studies from the literature, the Higher-Order π-calculus [44] and Core Erlang [7]. Causal-consistent reversible semantics for both of them are available in the literature [26, 29]. We show that the ones derived using our approach, albeit syntactically different, are equivalent to the ones in the literature. In the case of Core Erlang we go beyond the literature, which covers only the functional and concurrent fragment of Core Erlang, showing how to deal also with constructs for error handling based on links.

4.1 Reversible Semantics for Higher-Order π-calculus

In the previous sections, we already shown how to apply our approach to the Higher-Order π-calculus. We show here that the semantics derived using our approach is equivalent to the one of $\rho\pi$, the reversible HOπ in the literature [26]. Additionally, it is easy to see that the notion of concurrency induced by our approach (Definition 13) on HOπ matches the definition of concurrent transitions of [26, Definition 9].

Our reversible HOπ and the one in the literature are indeed quite close, but for a few differences. Our approach stores a context for the resulting term, in $\rho\pi$ only a key is kept. Actually, the context is always composed by parallel operators and restriction operators. The former are always collected during $\rho\pi$ backward steps, the latter are instead removed by $\rho\pi$ structural congruence when no more needed. Also, in $\rho\pi$ restrictions for keys are explicit, in our approach they are implicit. In addition, the single key kept in $\rho\pi$ is split into multiple complex tags, in direct correspondence with our keys, by $\rho\pi$ structural congruence, hence in $\rho\pi$ one key is enough.

For instance, starting from the system $R = k_1 : a\langle P_1 \mid P_2\rangle \mid k_2 : a(X) \triangleright X$, in our reversible HO$\pi$ semantics, by applying rule (F-Act), we have:

$$k_1 : a\langle P_1 \mid P_2\rangle \mid k_2 : a(X) \triangleright X \twoheadrightarrow j_1 : P_1 \mid j_2 : P_2 \mid [R\,; j_1 : \bullet_1 \mid j_2 : \bullet_2]$$

In $\rho\pi$, by applying rule (R.Fw) followed by structural congruence [26], we have:

$$R \twoheadrightarrow \nu k\,.k(P_1 \mid P_2) \mid [R\,; k] \equiv \nu k, \tilde{j}\,.(\langle j_1, \tilde{j}\rangle \cdot k : P_1 \mid \langle j_2, \tilde{j}\rangle \cdot k : P_2) \mid [R\,; k]$$

Structural congruence splits key k referring to the whole continuation into complex tags $\langle j_i, \tilde{j}\rangle \cdot k$, where $\tilde{j} = \{j_1, j_2\}$ and $i \in \{1, 2\}$. By using structural congruence, complex tags for single entities can be always generated, as in our example above.

Despite the differences, our reversible HOπ semantics and $\rho\pi$ semantics [26] are equivalent. To show this, we exploit the encoding function $(\!|\cdot|\!) : \mathcal{R} \to \mathcal{M}$ which translates a reversible HOπ configuration into a $\rho\pi$ configuration. Function $(\!|\cdot|\!)$ needs to extract the set of keys of all entities obtained by the split from the memory of our HOπ system and to construct the complex tags of $\rho\pi$ configuration. The encoding function together with other technical details can be found in Appendix A. Using the encoding function above we can show a bijective correspondence between transitions in our approach and $\rho\pi$ transitions.

▶ **Theorem 24.** *Let R be a reachable configuration of reversible HOπ with $(\!|R|\!) = M$. There is a transition $R \to R'$ in reversible HOπ iff there is a $\rho\pi$ transition $M \to M'$ with $(\!|R'|\!) \equiv M'$.*

4.2 Reversible Semantics for Erlang

In this section we apply our approach to Core Erlang [7], an intermediate step in the compilation of the concurrent and functional language Erlang. We also show the equivalence between the obtained reversible semantics and the one in [30]. As a forward model, we use the logging semantics of Core Erlang [30, Figure 14] (used also in [31]) with some minor changes: we use floating messages, as in [33], instead of a global mailbox Γ and we omit the labels of the relation $\hookrightarrow$. Indeed, labels are used in [30] to log the steps of the computation so to be able to replay it from logs [31, 30]. In our work, we are not interested in replaying from logs, therefore we do not need this information.

The semantics of Core Erlang is defined in a modular way as in [30], with relation $\rightarrow$ modelling the evaluation of expressions and relation $\hookrightarrow$ representing reductions of systems. Due to space constraints, we only present the application of our approach to selected rules of the evaluation of systems $\rightarrow$, referring to the Appendix A for the others. Since evaluation of expressions is not central for us, we refer to [30] for their description.

A Core Erlang system E is defined as a pool of processes and floating messages:

$$E := \langle p, \theta, e\rangle \mid (p, p', v) \mid \ (E_1 \mid E_2)$$

where

- $\langle p, \theta, e\rangle$ represents a process evaluating expression e in environment θ and uniquely identified by a pid (process identifier) p;
- (p, p', v) stands for a floating message carrying value v sent by the process with pid p to the one with pid p'. A floating message is a message in the system after it is sent and before it is received.

We show below rules (Send) and (Rec) of Core Erlang semantics, the full semantics is in Figure 7 of Appendix A.

$$\text{(Send)}\ \frac{\theta, e \xrightarrow{\texttt{send}(p', v)} \theta', e'}{\langle p, \theta, e\rangle \hookrightarrow \langle p, \theta', e'\rangle \mid (p, p', v)} \qquad \text{(Rec)}\ \frac{\theta, e \xrightarrow{\texttt{rec}(\kappa, \overline{cl_n})} \theta', e' \ \text{ and } \texttt{matchrec}(\theta, \overline{cl_n}, v) = (\theta_i, e_i)}{(p', p, v) \mid \langle p, \theta, e\rangle \hookrightarrow \langle p, \theta'\theta_i, e'\{\kappa \mapsto e_i\}\rangle}$$

Roughly speaking, rule (Send) states that if the evaluation of the expression e in the premise requires as a side effect to send value v to process p', the process evolves accordingly and a corresponding message is added to the system. Dually, rule (Rec) receives a message if the expression e requires a message matching some clauses $\overline{cl_n}$ and the message at hand indeed matches one of the clauses (second premise).

Now, we can apply our approach to the Core Erlang semantics and derive a reversible semantics for it. A reversible Core Erlang configuration, denoted with R, is defined as usual by adding keys and memories to an Erlang systems, as formalised by the following grammar:

$$R ::= k : \langle p, \theta, e\rangle \mid k : (p, p', v) \ \mid \ (R_1 \mid R_2) \mid [R; C]$$

In the following, we give the forward rules (F-Send) and (F-Rec) of the reversible semantics for Erlang. The complete set of forward rules is given in Figure 8 of Appendix A.

$$\text{(F-Send)}\ \frac{\theta, e \xrightarrow{\texttt{send}(p', v)} \theta', e' \quad k_1, k_2 \text{ are fresh keys}}{k : \langle p, \theta, e\rangle \twoheadrightarrow k_1 : \langle p, \theta', e'\rangle \mid k_2 : (p, p', v) \mid [k : \langle p, \theta, e\rangle\ ; k_1 : \bullet_1 \mid k_2 : \bullet_2]}$$

$$\text{(F-Rec)}\ \frac{\theta, e \xrightarrow{\texttt{rec}(\kappa, \overline{cl_n})} \theta', e' \quad \text{and} \quad \texttt{matchrec}(\theta, \overline{cl_n}, v) = (\theta_i, e_i) \quad k_1 \text{ is a fresh key}}{k_2 : (p', p, v) \mid k : \langle p, \theta, e\rangle \twoheadrightarrow k_1 : \langle p, \theta'\theta_i, e'\{\kappa \mapsto e_i\}\rangle \mid [k_2 : (p', p, v) \mid k : \langle p, \theta, e\rangle\ ; k_1 : \bullet_1]}$$

Actually, both rules are to be interpreted as schemas, so that premises related to the semantics of expressions and to match are used to select the allowed instances and do not occur in actual instances. E.g., an allowed instance for (F-Send) is:

$$\frac{k_1, k_2 \text{ are fresh keys}}{k : \langle p, \theta, p'!5\rangle \twoheadrightarrow k_1 : \langle p, \theta, 5\rangle \mid k_2 : (p, p', 5) \mid [k : \langle p, \theta, p'!5\rangle \,; k_1 : \bullet_1 \mid k_2 : \bullet_2]}$$

where ! is Erlang operator for sending.

Below, we give the backward rules (B-Send) and (B-Rec) of the reversible semantics for Erlang. The complete set of backward rules is given in Figure 9 of Appendix A. When the action is undone, the prior state of the process is restored from the memory μ.

$$\text{(B-Send) } k_1 : \langle p, \theta', e'\rangle \mid k_2 : (p, p', v) \mid [k : \langle p, \theta, e\rangle \,; k_1 : \bullet_1 \mid k_2 : \bullet_2] \rightsquigarrow k : \langle p, \theta, e\rangle$$

$$\text{(B-Rec) } k_1 : \langle p, \theta', e'\rangle \mid [k_2 : (p', p, v) \mid k : \langle p, \theta, e\rangle \,; k_1 : \bullet_1] \rightsquigarrow k_2 : (p', p, v) \mid k : \langle p, \theta, e\rangle$$

The reversible semantics obtained by applying our approach to Core Erlang is not exactly the same as the one of [30]. There are two important differences. First, we are not using execution logs, that we removed from the semantics we gave in input to our approach, since we do not need them.

Another difference is in how the past information of the system is stored. In [30], a history element h is kept as part of the process $\langle p, h, \theta, e\rangle$. It contains information to recover all past states of the process. In our reversible semantics, each step generates a memory with the information needed to reverse it, and the memories are connected using keys. Also, memories are not inside processes but floating in the configuration.

In the following, we prove that, despite the differences above, the two semantics capture the same behaviours. To this end, we first show that the two semantics are based on the same notion of causality (by showing that conflicting transitions are the same) and then that they are strong back and forth barbed equivalent [26]. Here we just discuss the idea, we refer to Appendix A for the technical details.

The notion of conflict for reversible Core Erlang in [30, Definition 12] (which is an instance of the happened-before relation [20] as discussed in [30]) is defined in general terms, referring to which actions (e.g., send, ...) are performed and by which processes. Hence, it is also applicable to our reversible Core Erlang. We show below that it coincides with the definition we gave, based on keys and memories.

▶ **Theorem 25** (Causal correspondence). *Two coinitial transitions t_1 and t_2 of our reversible Core Erlang semantics are in conflict according to [30, Definition 12] iff they are in conflict according to Definition 13.*

We show below that the reversible semantics of Erlang in [30] and ours are strong back and forth barbed equivalent [26]. We let E to stand for a Core Erlang system, L for a reversible Erlang system as in [30] and R for one of our reversible Erlang configurations.

Following [33], we write $E \downarrow p$ if the system E contains a floating message targeting a process with pid p (i.e., if $(p', p, v) \mid E' \equiv E$ for some p', v and E'). We use the same notation for systems L and configurations R, writing $L \downarrow p$ and $R \downarrow p$.

We now adapt the definition of back and forth barbed bisimulation [26] to reversible Erlang. In words, two reversible semantics are back and forth barbed bisimilar if they have the same barbs and they can match each other execution steps. Formally:

▶ **Definition 26.** *Relation $\mathcal{R}$ is a strong back and forth barbed simulation if $(L, R) \in \mathcal{R}$ implies:*

- *$L \downarrow p$ implies $R \downarrow p$*
- *$L \rightharpoonup L'$ implies $R \twoheadrightarrow R'$ with $(L', R') \in \mathcal{R}$*
- *$L \leftharpoondown L'$ implies $R \rightsquigarrow R'$ with $(L', R') \in \mathcal{R}$*

Relation $\mathcal{R}$ is a strong back and forth barbed bisimulation if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are strong back and forth barbed simulations. Strong back and forth barbed bisimilarity is the largest strong back and forth barbed bisimulation.

Now we can state the equivalence result between the two semantics.

▶ **Theorem 27.** *The reversible semantics of Erlang in [30] and our reversible semantics of Erlang are strong back and forth barbed bisimilar.*

4.3 Reversible Link Semantics for Erlang

Here, we apply our approach to the remote error handling mechanism of Core Erlang, based on links. No reversible semantics for it exists in the literature as far as we know. Defining it correctly does not present specific technical challenges, but it requires care, hence its definition is an interesting result on its own.

We start by giving some general idea about links and their role in Erlang (see [15] for more details). A link can be seen as a bidirectional path between two processes along which error signals travel. This can be used, e.g., to signal normal or abnormal termination. A process terminates normally when its code is completely executed, or it can terminate abnormally with a "reason", meaning that some faulty behaviour occurred during the execution. In both the cases, the process signals its termination to linked processes. This gives to the receiver the role of a controller in charge of handling the termination. There are two possibilities, depending on the nature of the receiver process: it can terminate too, or, if it is a system process, it can trap the termination signal and "resolve" the faulty behaviour. For instance, it could ignore it and continue with its execution, or start a copy of the terminated process, etc. Thanks to this feature, Erlang is particularly suited to build fault-tolerant systems [1].

In Erlang, links between two processes can be created by calling either function `link`(), linking any two processes (provided they are not terminated yet) or function `spawn_link`(), which spawns a new process and links it with the parent process atomically. In this work, we concentrate on function `spawn_link`(). Function `link`() can be dealt with similarly.

We start from the Core Erlang semantics discussed in the previous section and extend it to support the functions `spawn_link`() and `process_flag`(). The latter allows one to set the state of a process to system process, i.e. a process which will trap the error signal, or non-system process. More precisely, we add to Core Erlang syntax (see [30, Section 2.1, Figure 1]) expressions `spawn_link`$(expr, [expr_1, \ldots, expr_1])$ and `process_flag`$(expr_1, expr_2)$. In our case, function `process_flag`() is always called as `process_flag`$(trap\_exit, flag)$, where the process becomes a system process if `flag` = `true`, a non-system process otherwise.

We show now a sample Erlang program to clarify the error handling mechanism described above. It calculates the sum of a given list of elements and returns *invalid* if the list contains some non-numeric element.

The execution starts by calling function `total`(), which first sets the process flag to `true`. In this way, the process will be able to trap termination signals from any process linked with it. The execution proceeds by calling function `spawn_link`(), which atomically spawns and links a new process, executing function `sumProcess`(), in charge of calculating the sum via auxiliary function `sum`(). Because of the link, when the linked process terminates, its parent process will receive an exit notification message.

Finally, function `receiveValue`() is invoked. It checks whether the computation finished without misbehaviours: if this is the case message $\{'\texttt{EXIT}', Pid, \texttt{normal}\}$ is received and the function will read the result of the computation. If an error occurred during the computation message $\{'EXIT', Pid, \{\textbf{badarith}, Stack\}\}$ is received and the function returns atom *invalid*.

$\texttt{total}(List) \rightarrow$
  $\textbf{process\_flag}(trap\_exit, true)$,
  $\texttt{SumPid} = \textbf{spawn\_link}(?MODULE, \texttt{sumProcess}, [\textbf{self()}, List])$,
  $\texttt{receiveValue}(\texttt{SumPid})$.
$\texttt{sumProcess}(Pid, List) \rightarrow Pid \,!\, \texttt{sum}(List)$.
$\texttt{sum}([\,]) \rightarrow 0$;
$\texttt{sum}([H|T]) \rightarrow H + \texttt{sum}(T)$.
$\texttt{receiveValue}(Pid) \rightarrow$
  receive
    $\{'EXIT', Pid, normal\} \rightarrow$
      receive $Value \rightarrow Value$ **end**;
    $\{'EXIT', Pid, \{\textbf{badarith}, Stack\}\} \rightarrow invalid$
  end.

To integrate the functions `spawn_link`() and `process_flag`(), we add to processes two pieces of information, the set of links l and the flag f. The link set l is updated when a link is created, adding the pid of the other process, or destroyed, removing it. The flag f is a Boolean, tracking whether a process is a system process or not. Also, we say that the process $\langle p, \theta, e, l, f\rangle$ is *terminated* if $e = v$ for some value v (normal termination) or $e = r$ for some reason r (faulty termination).

Formally, a Core Erlang system supporting links is defined as a pool of floating messages, live and terminated processes, with the following grammar:

$$E := \langle p, \theta, e, l, f\rangle \mid (p, p', v) \mid \ (E_1 \mid E_2)$$

By applying our approach we obtain the syntax for reversible Core Erlang supporting links below. As usual, keys and memories are added to the system.

$$R := k : \langle p, \theta, e, l, f\rangle \mid k : (p, p', v) \mid \ R_1 \mid R_2 \ \mid [R; C]$$

The new rules of the reversible semantics of Erlang supporting links are in Figure 4, but for rule (F-Nrm) which is very similar to rule (F-Err) and is given in Appendix A. We do not show the original semantics, which can however be easily deduced by removing keys and memories from the one in Figure 4. The reversible semantics includes also all the rules in Figure 8, with the only addition of the set of links l and flag f in each process, which are not affected by those rules, but for the fact that when a process is spawned its link set is initialised to empty and its flag to `false`.

Rule (F-SpLink) above is similar to rule (F-Spawn) in Figure 8, with the addition that the link between the two processes is created, by inserting the pid of the other process in the link set. Rules (F-Err) and (F-Nrm) are similar: they both model the signalling of the termination of a process p to all the processes it is linked with. In both of them links are broken, by removing pids from link sets. The effect of termination depends on whether it is a normal termination, as in rule (F-Nrm), or an error termination, as in rule (F-Err). Also,

$$\text{(F-SpLink)}\ \frac{\theta, e \xrightarrow{\texttt{spawn\_link}(\kappa, f/n, [\overline{v_n}])} \theta', e' \quad p' \text{ is a fresh pid} \quad k_1, k_2 \text{ are fresh keys}}{k : \langle p, \theta, e, l, f\rangle \twoheadrightarrow k_1 : \langle p, \theta', e'\{\kappa \mapsto p'\}, l \cup \{p'\}, f\rangle \mid k_2 : \langle p', id, \texttt{apply}\ f/n\ (\overline{v_n}), \{p\}, false\rangle \mid [k : \langle p, \theta, e, l, f\rangle\ ; k_1 : \bullet_1 \mid k_2 : \bullet_2]}$$

$$\text{(F-Flag)}\ \frac{\theta, e \xrightarrow{\texttt{process\_flag}(\kappa, trap\_exit, f')} \theta', e' \quad k_1 \text{ is a fresh key}}{k : \langle p, \theta, e, l, f\rangle \twoheadrightarrow k_1 : \langle p, \theta', e'\{\kappa \mapsto f\}, l, f'\rangle \mid [k : \langle p, \theta, e, l, f\rangle\ ; k_1 : \bullet_1]}$$

$$\text{(F-Err)}\ \frac{l = \{p_1, \ldots, p_m\} \quad 1 \le i \le n \Rightarrow f_i = \texttt{true} \ \wedge\ n+1 \le i \le m \Rightarrow f_i = \texttt{false} \quad h, h_i, j_i \text{ are fresh keys}}{k : \langle p, \theta, r, l, f\rangle \mid \prod_{1\le i\le m} k_i : \langle p_i, \theta_i, e_i, l_i, f_i\rangle \twoheadrightarrow h : \langle p, \theta, r, \emptyset, f\rangle \mid \prod_{1\le i\le n} h_i : \langle p_i, \theta_i, e_i, l_i \setminus \{p\}, f_i\rangle \mid \prod_{1\le i\le n} j_i : (p, p_i, \{'\texttt{EXIT}', p, r\}) \mid \prod_{n+1\le i\le m} h_i : \langle p_i, \theta_i, r, l_i \setminus \{p\}, f_i\rangle \mid [k : \langle p, \theta, r, l, f\rangle \mid \prod_{1\le i\le m} k_i : \langle p_i, \theta_i, e_i, l_i, f_i\rangle\ ; h : \bullet_h \mid \prod_{1\le i\le m} h_i : \bullet_{h_i} \mid \prod_{1\le i\le n} j_i : \bullet_{j_i}]}$$

Figure 4 Forward rules of the reversible link semantics for Erlang.

a termination signal affects system processes and non-system processes differently, and this is why in the two rules we split the processes in two groups according to the value of the flag. It can be set to the desired value using rule (F-Flag).

In rule (F-Err), the process terminates for some reason r. In this case messages $\{'\texttt{EXIT}', p, r\}$ where p is the pid of the terminated process are sent to all system processes while non-system processes are forced to terminate. We can see the latter, e.g., in non-system process $\langle p_m, \theta_m, r, l_m \setminus \{p\}, f_m\rangle$ where expressions e_m is replaced by reason r, denoting abnormal termination. A memory is generated as usual, recording the past state of the system and the configuration of the resulting processes.

In rule (F-Nrm), the process terminates normally. The only differences w.r.t. rule (F-Err) is that non-system processes are unaffected and the messages are of the form $\{'\texttt{EXIT}', p, \texttt{normal}\}$.

Backward rules are as usual.

We need to couple the rules in Figure 4, describing the semantics of Erlang configurations, with rules describing the evaluation of expressions, as the ones in [30, Figure 11]. Two main changes are needed. First, evaluation of operators may produce either a value or an error:

$$\text{(Call3)}\ \frac{\texttt{eval}(op, v_1, \ldots, v_n) = x \text{ with } x = v \vee x = r}{\theta, \texttt{call}\ op(v_1, \ldots, v_n) \xrightarrow{\tau} \theta, x}$$

Then, one also needs to add rules to evaluate the functions `spawn_link`() and `process_flag`(). They are quite standard and can be found in Appendix A.

We are working to integrate the error mechanisms above into Erlang reversible debugger CauDEr [28]. CauDEr follows the reversible semantics of Core Erlang in [30], however our results can be rephrased in that setting, as hinted at by Theorems 25 and 27.

5 Conclusion, Related and Future Work

We presented a fully automatic method to extend a given forward model to a reversible one. Notably, our approach only needs a syntax and a reduction semantics of the forward model fitting our constraints. A causal semantics is produced as a by-product of our approach (see Definition 13). We exploited our method to obtain reversible extensions of Higher-Order π and Core Erlang. We showed that the obtained reversible semantics are equivalent to the ones in the literature [26, 30]. As an illustration that our approach can go beyond the literature, we tackled Core Erlang constructs for remote error handling based on links.

Sequential systems would correspond in our framework to single entities which evolve using instances of schemas having a single entity both on the left- and on the right-hand side. While our approach would create a reversible semantics for them, undoing actions in reverse order of execution, many of our results would become trivial.

In the concurrency literature, one can find many approaches defining a single reversible formalism or studying its properties, all using techniques tailored to the chosen model (e.g., [10, 9, 26, 39, 17, 38, 3, 18, 29, 37]). Indeed, our work can be seen as a generalisation of [26, 29], which we also used as case studies. A few works present general approaches able to cope with a number of formalisms. Our work fits in this class, hence we compare it below with the other approaches of this kind we are aware of. Also, since we deal with concurrent models, we focus on approaches targeting them as well.

Beyond ours, the only work that we are aware of providing a general and fully automatic method to derive a reversible semantics is [42], which considers calculi defined in a specific SOS format. Their approach allows to deal with open systems since their semantics is SOS, while our approach based on a reduction semantics considers only close systems. On the other hand, the higher degree of abstraction provided by reduction semantics simplifies the approach and makes it applicable to a wider range of formalisms. Indeed, the approach in [42] cannot cope with our two case studies, Higher-Order π-calculus and Core Erlang, since they do not fit their SOS format.

Also [4], which presents a modular framework to define reversible extensions of models such as CCS and concurrent X-machines, can deal with open systems. Its main limitation is that it is not fully automatic. Indeed, it requires to manually refine the labels of a given LTS to ensure properties such as determinism and codeterminism. This is far from trivial.

Two abstract approaches to reversibility are [13] and [32]. The former focuses on the interplay between reversible and irreversible actions, hence its results become trivial if, like in our case, there are no irreversible actions. We exploited the latter to prove properties of reversible models built using our approach. It concentrates on deriving properties from a set of axioms but gives no indication about how to render an irreversible system reversible.

Uncontrolled reversible semantics as obtained by our approach are the foundation of a reversible model on which one can build on, by adding control mechanisms [25] such as irreversible actions [11], rollback operators [17] or energy potentials [2]. An interesting line for future work is to integrate such approaches in our framework. For rollback, we could follow the ideas in [22], which leave however open the issue of how to manage rollback targets.

Another direction for future work is to adapt our approach so to handle further forward models. For instance, we currently cannot cope with the semantics of muKlaim defined in [17], since its concurrency model includes read dependencies. In particular, our approach is based on consumed and produced resources, while in [17] resources can also be read without being consumed. More in general, our approach can cope well with message-based concurrency modelled by some form of happened-before relation [20] (e.g., beyond our case studies, CCS, π-calculus and place-transition Petri nets) but not with read-write concurrency (e.g., beyond muKlaim, imperative languages). In order to extend our method to cope with read-write concurrency, we need to identify resources which are read but not consumed.

A last direction for future work concerns reducing the memory overhead of our approach. While it is difficult to find optimisations sound for every instance, many optimisations can work on specific classes of instances. E.g., in models where the context T in the instances of schema (Scm-Act) is always composed by parallel operators only, as in Core Erlang, there is no need to store T, but it is enough to store the set of fresh keys.

References

1 Joe Armstrong. Erlang – software for a concurrent world. In Erik Ernst, editor, *ECOOP 2007 – Object-Oriented Programming, 21st European Conference, Berlin, Germany, July 30 - August 3, Proceedings*, volume 4609 of *Lecture Notes in Computer Science*, page 1. Springer, 2007. doi:10.1007/978-3-540-73589-2_1.

2 Giorgio Bacci, Vincent Danos, and Ohad Kammar. On the statistical thermodynamics of reversible communicating processes. In *Algebra and Coalgebra in Computer Science – 4th International Conference, CALCO, Winchester, UK, August 30 – September 2, 2011. Proceedings*, volume 6859 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011. doi:10.1007/978-3-642-22944-2_1.

3 Kamila Barylska, Evgeny Erofeev, Maciej Koutny, Lukasz Mikulski, and Marcin Piatkowski. Reversing transitions in bounded Petri nets. *Fundam. Inform.*, 157(4):341–357, 2018. doi:10.3233/FI-2018-1631.

4 Alexis Bernadet and Ivan Lanese. A modular formalization of reversibility for concurrent models and languages. In *Proceedings 9th Interaction and Concurrency Experience, ICE 2016, Heraklion, Greece, 8-9*, pages 98–112, 2016. doi:10.4204/EPTCS.223.7.

5 Bob Boothe. Efficient algorithms for bidirectional debugging. In *Proceedings of the 2000 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), Vancouver, British Columbia, Canada, June 18-21*, PLDI '00, pages 299–310, New York, NY, USA, 2000. ACM. doi:10.1145/349299.349339.

6 Luca Cardelli and Cosimo Laneve. Reversible structures. In *Computational Methods in Systems Biology, 9th International Conference, CMSB 2011, Paris, France, September 21–23. Proceedings*, pages 131–140, 2011. doi:10.1145/2037509.2037529.

7 Richard Carlsson, Björn Gustavsson, Erik Johansson, Thomas Lindgren, Sven-Olof Nyström, Mikael Pettersson, and Robert Virding. *Core Erlang 1.0.3. Language specification*, 2004. URL: https://www.it.uu.se/research/group/hipe/cerl/doc/core_erlang-1.0.3.pdf.

8 Christopher D. Carothers, Kalyan S. Perumalla, and Richard Fujimoto. Efficient optimistic parallel simulations using reverse computation. *ACM TOMACS*, 9(3):224–253, 1999. doi:10.1145/347823.347828.

9 Ioana Cristescu, Jean Krivine, and Daniele Varacca. A compositional semantics for the reversible pi-calculus. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28*, pages 388–397. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.45.

10 Vincent Danos and Jean Krivine. Reversible communicating systems. In *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, Proceedings*, volume 3170 of *Lecture Notes in Computer Science*, pages 292–307. Springer, 2004. doi:10.1007/978-3-540-28644-8_19.

11 Vincent Danos and Jean Krivine. Transactions in RCCS. In *CONCUR 2005 - Concurrency Theory, 16th International Conference, San Francisco, CA, USA, August 23-26, Proceedings*, volume 3653 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 2005. doi:10.1007/11539452_31.

12 Vincent Danos and Jean Krivine. Formal molecular biology done in CCS-R. *Electr. Notes Theor. Comput. Sci.*, 180(3):31–49, 2007. doi:10.1016/j.entcs.2004.01.040.

13 Vincent Danos, Jean Krivine, and Paweł Sobociński. General reversibility. In *Proceedings of the 13th International Workshop on Expressiveness in Concurrency, EXPRESS 2006, Bonn, Germany, August 26*, pages 75–86, 2006. doi:10.1016/j.entcs.2006.07.036.

14 Rolf Drechsler and Robert Wille. From truth tables to programming languages: Progress in the design of reversible circuits. In *IEEE International Symposium on Multiple-Valued Logic, ISMVL*, pages 78–85, 2011. doi:10.1109/ISMVL.2011.40.

15 Erlang website. https://www.erlang.org/.

16 Elena Giachino, Ivan Lanese, and Claudio Antares Mezzina. Causal-consistent reversible debugging. In *Fundamental Approaches to Software Engineering - 17th International Conference, FASE 2014, Proceedings*, pages 370–384, 2014. doi:10.1007/978-3-642-54804-8_26.

17 Elena Giachino, Ivan Lanese, Claudio Antares Mezzina, and Francesco Tiezzi. Causal-consistent rollback in a tuple-based language. *J. Log. Algebraic Methods Program.*, 88:99–120, 2017. doi:10.1016/j.jlamp.2016.09.003.

18 Eva Graversen, Iain Phillips, and Nobuko Yoshida. Event structure semantics of (controlled) reversible CCS. In *Reversible Computation - 10th International Conference, RC 2018, Leicester, UK, September 12-14. Proceedings*, volume 11106 of *Lecture Notes in Computer Science*, pages 102–122. Springer, 2018. doi:10.1007/978-3-319-99498-7_7.

19 Stefan Kuhn and Irek Ulidowski. Local reversibility in a calculus of covalent bonding. *Sci. Comput. Program.*, 151:18–47, 2018. doi:10.1016/j.scico.2017.09.008.

20 Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978. doi:10.1145/359545.359563.

21 Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5:183–191, 1961. doi:10.1147/rd.53.0183.

22 Ivan Lanese. From reversible semantics to reversible debugging. In *Reversible Computation - 10th International Conference, RC 2018, Leicester, UK, September 12-14, Proceedings*, volume 11106 of *Lecture Notes in Computer Science*, pages 34–46. Springer, 2018. doi:10.1007/978-3-319-99498-7_2.

23 Ivan Lanese and Doriana Medić. A general approach to derive uncontrolled reversible semantics (TR). Available at https://hal.archives-ouvertes.fr/hal-02902204.

24 Ivan Lanese, Claudio Antares Mezzina, Alan Schmitt, and Jean-Bernard Stefani. Controlling reversibility in higher-order pi. In *CONCUR 2011 - Concurrency Theory - 22nd International Conference, Aachen, Germany, September 6-9. Proceedings*, volume 6901 of *Lecture Notes in Computer Science*, pages 297–311. Springer, 2011. doi:10.1007/978-3-642-23217-6_20.

25 Ivan Lanese, Claudio Antares Mezzina, and Jean-Bernard Stefani. Controlled reversibility and compensations. In *Reversible Computation, 4th International Workshop, RC 2012, Copenhagen, Denmark, July 2-3. Revised Papers*, volume 7581 of *Lecture Notes in Computer Science*, pages 233–240. Springer, 2012. doi:10.1007/978-3-642-36315-3_19.

26 Ivan Lanese, Claudio Antares Mezzina, and Jean-Bernard Stefani. Reversibility in the higher-order π-calculus. *Theor. Comput. Sci.*, 625:25–84, 2016. doi:10.1016/j.tcs.2016.02.019.

27 Ivan Lanese, Claudio Antares Mezzina, and Francesco Tiezzi. Causal-consistent reversibility. *Bulletin of the EATCS*, 114, 2014. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/305.

28 Ivan Lanese, Naoki Nishida, Adrián Palacios, and Germán Vidal. Cauder: A causal-consistent reversible debugger for Erlang. In *Functional and Logic Programming - 14th International Symposium, FLOPS 2018, Nagoya, Japan, May 9-11, Proceedings*, volume 10818 of *Lecture Notes in Computer Science*, pages 247–263. Springer, 2018. doi:10.1007/978-3-319-90686-7_16.

29 Ivan Lanese, Naoki Nishida, Adrián Palacios, and Germán Vidal. A theory of reversibility for Erlang. *J. Log. Algebraic Methods Program.*, 100:71–97, 2018. doi:10.1016/j.jlamp.2018.06.004.

30 Ivan Lanese, Adrián Palacios, and Germán Vidal. Causal-consistent replay debugging for message passing programs. In *Technical report, DSIC, Universitat Politecnica de Valencia*, 2019. URL: http://personales.upv.es/gvidal/german/forte19tr/paper.pdf.

31 Ivan Lanese, Adrián Palacios, and Germán Vidal. Causal-consistent replay debugging for message passing programs. In *Formal Techniques for Distributed Objects, Components, and Systems - 39th IFIP WG 6.1 International Conference, FORTE 2019, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17-21, Proceedings*, pages 167–184, 2019. doi:10.1007/978-3-030-21759-4_10.

32 Ivan Lanese, Iain C. C. Phillips, and Irek Ulidowski. An axiomatic approach to reversible computation. In *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, Proceedings*, pages 442–461, 2020. doi:10.1007/978-3-030-45231-5_23.

33 Ivan Lanese, Davide Sangiorgi, and Gianluigi Zavattaro. Playing with bisimulation in Erlang. In *Models, Languages, and Tools for Concurrent and Distributed Programming - Essays Dedicated to Rocco De Nicola on the Occasion of His 65th Birthday*, pages 71–91, 2019. doi:10.1007/978-3-030-21485-2_6.

34 Johan Sund Laursen, Ulrik Pagh Schultz, and Lars-Peter Ellekilde. Automatic error recovery in robot assembly operations using reverse execution. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2*, pages 1785–1792. IEEE, 2015. doi:10.1109/IROS.2015.7353609.

35 Christopher Lutz. Janus: a time-reversible language. *Letter to R. Landauer.*, 1986. URL: http://tetsuo.jp/ref/janus.html.

36 James McNellis, Jordi Mola, and Ken Sykes. Time travel debugging: Root causing bugs in commercial scale software. CppCon talk, https://www.youtube.com/watch?v=l1YJTg_A914, 2017.

37 Hernán C. Melgratti, Claudio Antares Mezzina, and Irek Ulidowski. Reversing P/T nets. In Hanne Riis Nielson and Emilio Tuosto, editors, *Coordination Models and Languages - 21st IFIP WG 6.1 International Conference, COORDINATION 2019, Held as Part of the 14th International Federated Conference on Distributed Computing Techniques, DisCoTec 2019, Kongens Lyngby, Denmark, June 17-21, Proceedings*, volume 11533 of *Lecture Notes in Computer Science*, pages 19–36. Springer, 2019. doi:10.1007/978-3-030-22397-7_2.

38 Claudio Antares Mezzina. On reversibility and broadcast. In *Reversible Computation - 10th International Conference, RC 2018, Leicester, UK, September 12-14. Proceedings*, volume 11106 of *Lecture Notes in Computer Science*, pages 67–83. Springer, 2018. doi:10.1007/978-3-319-99498-7_5.

39 Claudio Antares Mezzina and Jorge A. Pérez. Causally consistent reversible choreographies: a monitors-as-memories approach. In *Proceedings of the 19th International Symposium on Principles and Practice of Declarative Programming, Namur, Belgium, October 09 - 11, 2017*, pages 127–138. ACM, 2017. doi:10.1145/3131851.3131864.

40 Anna Philippou and Kyriaki Psara. Reversible computation in Petri nets. In *Reversible Computation - 10th International Conference, RC 2018, Leicester, UK, September 12-14, Proceedings*, pages 84–101, 2018. doi:10.1007/978-3-319-99498-7_6.

41 Iain Phillips, Irek Ulidowski, and Shoji Yuen. A reversible process calculus and the modelling of the ERK signalling pathway. In *Reversible Computation, 4th International Workshop, RC 2012, Copenhagen, Denmark, July 2-3. Revised Papers*, pages 218–232, 2012. doi:10.1007/978-3-642-36315-3_18.

42 Iain C. C. Phillips and Irek Ulidowski. Reversing algebraic process calculi. *J. Log. Algebraic Methods Program.*, 73(1-2):70–96, 2007. doi:10.1016/j.jlap.2006.11.002.

43 Gordon D. Plotkin. A structural approach to operational semantics. *J. Log. Algebraic Methods Program.*, 60-61:17–139, 2004.

44 Davide Sangiorgi. Bisimulation for higher-order process calculi. *Inf. Comput.*, 131(2):141–178, 1996. doi:10.1006/inco.1996.0096.

45 Vladimiro Sassone, Mogens Nielsen, and Glynn Winskel. Models of concurrency: Towards a classification. *Theoretical Computer Science*, 170(1-2):297–348, 1996. doi:10.1016/S0304-3975(96)80710-9.

46 Ulrik Pagh Schultz. Reversible object-oriented programming with region-based memory management - work-in-progress report. In *RC*, volume 11106 of *Lecture Notes in Computer Science*, pages 322–328. Springer, 2018. doi:10.1007/978-3-319-99498-7_22.

47 Irek Ulidowski, Iain Phillips, and Shoji Yuen. Reversing event structures. *New Generation Comput.*, 36(3):281–306, 2018. doi:10.1007/s00354-018-0040-8.

A Further Technical Material

Higher-Order π-calculus and reversible HOπ-calculus

This section recalls the semantics of the HOπ-calculus [44], discusses how it fits our framework and details the reversible semantics derived for it using our approach.

The standard rules of the structural congruence for the HOπ-calculus are:

$$\text{(ParC) } P \mid Q \equiv Q \mid P \qquad \text{(ParA) } P \mid (Q \mid S) \equiv (P \mid Q) \mid S \qquad \text{(Nil) } P \mid \mathbf{0} \equiv P$$

$$\text{(Alpha) } \nu a\, P \equiv \nu b\, P\{b/a\} \quad \text{if } b \notin \mathtt{fn}(P) \qquad \text{(ResF) } (\nu a\, P) \mid Q \equiv \nu a\, (P \mid Q) \quad \text{if } a \notin \mathtt{fn}(Q)$$

Rules (ParC) and (ParA) ensure that parallel composition is commutative and associative, while rule (Nil) defines $\mathbf{0}$ as neutral element as required by our framework. Rule (Alpha) is α-conversion while rule (ResF) deals with scope extrusion.

The semantics of HOπ is given by the reduction relation $\rightarrowtail$ below:

$$\text{(Act) } \frac{}{a\langle Q\rangle \mid a(X) \triangleright P \rightarrowtail P\{Q/X\}} \qquad \text{(Par) } \frac{P \rightarrowtail P'}{P \mid Q \rightarrowtail P' \mid Q}$$

$$\text{(Res) } \frac{P \rightarrowtail P'}{\nu a\, P \rightarrowtail \nu a\, P'} \qquad \text{(Eqv) } \frac{P \equiv P' \quad P \rightarrowtail Q \quad Q \equiv Q'}{P' \rightarrowtail Q'}$$

Rule (Act) is the communication rule where process Q is received and bound to variable X. Process P can execute inside a parallel or a restriction operator thanks to rules (Par) and (Res), respectively. Rules (Par) and (Eqv) are as required by our framework. Rules (Act) and (Res) can be seen as infinite families of rules fitting schemas (Scm-Act) and (Scm-Opn), respectively.

In Figure 5, we give forward and backward rules of the reversible semantics for the HOπ-calculus derived using our approach.

$$\text{(F-Act) } \frac{a\langle P\rangle \mid a(X) \triangleright P' \rightarrowtail P'\{P/X\} \qquad j_1, \dots j_m \text{ are fresh keys} \quad \text{and} \quad P'\{P/X\} = T[Q_1, \dots, Q_m]}{\begin{array}{l} k_1 : a\langle P\rangle \mid k_2 : a(X) \triangleright P' \twoheadrightarrow T[j_1 : Q_1, \dots, j_m : Q_m] \mid \\ \qquad [k_1 : a\langle P\rangle \mid k_2 : a(X) \triangleright P' \,;\, T[j_1 : \bullet_1, \dots, j_m : \bullet_m]] \end{array}}$$

$$\text{(F-Par) } \frac{R \twoheadrightarrow R' \quad (\mathtt{key}(R') \setminus \mathtt{key}(R)) \cap \mathtt{key}(R_1) = \emptyset}{R \mid R_1 \twoheadrightarrow R' \mid R_1} \qquad \text{(F-Res) } \frac{R \twoheadrightarrow R'}{\nu a\,(R) \twoheadrightarrow \nu a\,(R')}$$

$$\text{(F-Eqv) } \frac{R \equiv R' \quad R \twoheadrightarrow R_1 \quad R_1 \equiv R_1'}{R' \twoheadrightarrow R_1'}$$

$$\text{(B-Act) } \frac{\mu = [k_1 : a\langle P\rangle \mid k_2 : a(X) \triangleright P' \,;\, T[j_1 : \bullet_1, \dots, j_m : \bullet_m]]}{T[j_1 : Q_1, \dots, j_m : Q_m] \mid \mu \rightsquigarrow k_1 : a\langle P\rangle \mid k_2 : a(X) \triangleright P'} \qquad \text{(B-Par) } \frac{R' \rightsquigarrow R}{R' \mid R_1 \rightsquigarrow R \mid R_1}$$

$$\text{(B-Res) } \frac{R' \rightsquigarrow R}{\nu a\,(R') \rightsquigarrow \nu a\,(R)} \qquad \text{(B-Eqv) } \frac{R \equiv R' \quad R \rightsquigarrow R_1 \quad R_1 \equiv R_1'}{R' \rightsquigarrow R_1'}$$

Figure 5 Forward and backward rules of the reversible semantics for the Higher-Order π-calculus.

Properties

This section contains further technical material related to Section 3. We start by defining strong bisimilarity between the forward semantics of a reversible configuration R and the semantics of its projection on the original model $\varphi(R)$.

▶ **Definition 28.** *A relation $\mathcal{R}$ between reversible configurations R and forward-only systems N is a* strong bisimulation *whenever for each $(R, N) \in \mathcal{R}$:*
- *if $R \twoheadrightarrow R'$, then $N \rightarrowtail N'$ with $(R', N') \in \mathcal{R}$;*
- *if $N \rightarrowtail N'$, then $R \twoheadrightarrow R'$ with $(R', N') \in \mathcal{R}$.*

Strong bisimilarity is the largest strong bisimulation.

Notably, only forward actions of the reversible systems need to be matched.

Concurrency and Causal Consistency. To fit the framework of [32], we re-formulate our framework as a labelled transition system enriched with an independence relation (LTSI) [45, Definition 3.7].

▶ **Definition 29.** *A* labelled transition system *(LTS) is a structure $(\mathcal{R}, \mathcal{L}, \rightarrow)$, where $\mathcal{R}$ is a set of systems, $\mathcal{L}$ is the set of action labels and $\rightarrow \subset \mathcal{R} \times \mathcal{L} \times \mathcal{R}$ is a transition relation.*

▶ **Definition 30.** *A* labelled transition system with independence *(LTSI) is a structure $(\mathcal{R}, \mathcal{L}, \rightarrow, \iota)$, where $(\mathcal{R}, \mathcal{L}, \rightarrow)$ is an LTS and ι is the independence relation (an irreflexive symmetric binary relation on transitions).*

In our case, $\mathcal{R}$ is the set of configurations and $\mathcal{L}$ the set of labels of our transitions. The latter include both forward and backward transitions. Also, the notion of independence is defined on coinitial transitions and it coincides with the notion of concurrency, namely $\iota = \smile_c$.

Given that our reversible semantics satisfies all the required axioms (Proposition 20), thanks to [32], all instances of our framework satisfy the Parabolic Lemma, Causal Consistency, Causal Safety and Causal Liveness.

▶ **Proposition 31.** *For every instance of our framework, the LTSI $(\mathcal{R}, \mathcal{L}, \rightarrow, \smile_c)$ satisfies the Parabolic Lemma, Causal Consistency, Causal Safety and Causal Liveness.*

Correspondence between our reversible HOπ and $\rho\pi$

This section contains technical material necessary to define the correspondence between our reversible HOπ and $\rho\pi$ (Section 4.1).

In the following, we recall the definition of thread normal form from [26, Lemma 1] using which, by exploiting structural congruence, unique keys are generated for each primitive thread process in a configuration (primitive thread processes are entities in our terminology).

▶ **Definition 32** (Thread normal form). *For any closed configuration M in $\rho\pi$, we have*

$$M \equiv \nu\tilde{u} \prod_{i \in I} (\kappa_i : \rho_i) \prod_{j \in J} [\mu_j \,;k_j] \qquad \textit{with} \quad \rho_i = a_i\langle P_i \rangle \quad \textit{or} \quad \rho_i = a_i(X_i) \triangleright P_i$$

We now present an encoding from our reversible HOπ to $\rho\pi$. The encoding works in two steps: a first step explores memories in our configurations to find related sets of keys, the second step uses the gathered information to actually perform the translation. The first step is done by function $\texttt{col}(R)$, which computes (possibly annotated) sets of keys, one for each memory $[R' \,;C]$ in R. If C contains more than one key, the extracted set of keys is annotated with a fresh key k, what is denoted with $\texttt{key}(C)_k$. The formal definition of function $\texttt{col}(R)$ is in Figure 6. Note that the result of function $\texttt{col}(R)$ is a set of possibly annotated sets of keys. We denote with $\tilde{h}_k$ and $\tilde{h}$ sets $\texttt{key}(C)_k$ and $\texttt{key}(C)$, respectively. We also assume to have a fresh key generator, giving us fresh keys k as needed.

$$
\begin{aligned}
\mathtt{col}(\nu a\,(R)) &= \mathtt{col}(R)\\
\mathtt{col}(R_1 \mid R_2) &= \mathtt{col}(R_1) \cup \mathtt{col}(R_2)\\
\mathtt{col}([R\,;C]) &= \{\mathtt{key}(C)_k\} \quad \text{where } k \text{ is fresh} \quad \text{if } |\mathtt{key}(C)| > 1\\
\mathtt{col}([R\,;C]) &= \{\mathtt{key}(C)\} \qquad\qquad \text{if } |\mathtt{key}(C)| = 1\\
\mathtt{col}(k : P) &= \emptyset\\
\mathtt{col}(\mathbf{0}) &= \emptyset
\end{aligned}
$$

Figure 6 Function $\mathtt{col}()$.

The second step performs the translation of a given configuration R and uses as a parameter a set of sets of keys S as above, which is initialised as $S = \mathtt{col}(R)$. Let us denote with $\mathcal{M}$ the set of all $\rho\pi$ [26] configurations in normal form and with $\mathcal{R}$ the set of all configurations obtained by applying our approach to the HOπ-calculus. The encoding function $(\!|\cdot|\!) : \mathcal{R} \to \mathcal{M}$, is defined as:

$$
\begin{aligned}
(\!|R|\!) &= \nu K\,(\!|R|\!)_{\mathtt{col}(R)} \qquad \text{where } K = (\bigcup_{\tilde{h}_k \in \mathtt{col}(R)} \tilde{h} \cup \{k\}) \cup \bigcup_{\{h\} \in \mathtt{col}(R)} \{h\}\\
(\!|\nu a\, R|\!)_S &= \nu a\,(\!|R|\!)_S\\
(\!|R_1 \mid R_2|\!)_S &= (\!|R_1|\!)_S \mid (\!|R_2|\!)_S\\
(\!|[R\,;C]|\!)_S &= [(\!|R|\!)_S\,;(\!|C|\!)_S]\\
(\!|h : P|\!)_S &= \langle h, \tilde{h}\rangle \cdot k : P \qquad \text{if} \quad h \in \tilde{h} \text{ for some } \tilde{h}_k \in S\\
(\!|h : P|\!)_S &= h : P \qquad \text{if} \quad h \notin \tilde{h} \text{ for all } \tilde{h}_k \in S\\
(\!|C|\!)_S &= k \qquad \text{if } \mathtt{key}(C)_k \in S\\
(\!|C|\!)_S &= h \qquad \text{if } \mathtt{key}(C) = \{h\}
\end{aligned}
$$

Let us comment on it. The first rule computes the parameter $\mathtt{col}(R)$ containing information on keys, to be used in the rest of the translation, and creates restrictions for all the keys occurring in it. The other rules just propagate the set S, till one of the last 4 rules applies. The first two deal with keys labelling processes: it the key belongs to a non-singleton set, then it is replaced by a complex tag, otherwise it is left unchanged. The two last rules remove the context C in the memory, which is not needed in $\rho\pi$, replacing it with a key. If C contains only one key, this is the key used. If it contains more than one key instead the fresh key k generated for the set of keys is used. The keys in the set will become complex tags, carrying k so to make the connection between the memory and all the processes created by the corresponding transition.

Let us show a simple example to clarify how the translation works.

▶ **Example 33.** Let us consider the system produced by the sample transition in Section 4.1:

$$R' = j_1 : P_1 \mid j_2 : P_2 \mid [R\,; j_1 : \bullet_1 \mid j_2 : \bullet_2]$$

$$\text{(Seq)}\ \frac{\theta, e \xrightarrow{\tau} \theta', e'}{\langle p, \theta, e\rangle \hookrightarrow \langle p, \theta', e'\rangle} \qquad \text{(Rec)}\ \frac{\theta, e \xrightarrow{\mathtt{rec}(\kappa, \overline{cl_n})} \theta', e' \quad \text{and} \quad \mathtt{matchrec}(\theta, \overline{cl_n}, v) = (\theta_i, e_i)}{(p', p, v) \mid \langle p, \theta, e\rangle \hookrightarrow \langle p, \theta'\theta_i, e'\{\kappa \mapsto e_i\}\rangle}$$

$$\text{(Send)}\ \frac{\theta, e \xrightarrow{\mathtt{send}(p', v)} \theta', e'}{\langle p, \theta, e\rangle \hookrightarrow \langle p, \theta', e'\rangle \mid (p, p', v)} \qquad \text{(Self)}\ \frac{\theta, e \xrightarrow{\mathtt{self}(\kappa)} \theta', e'}{\langle p, \theta, e\rangle \hookrightarrow \langle p, \theta', e'\{\kappa \mapsto p\}\rangle}$$

$$\text{(Spawn)}\ \frac{\theta, e \xrightarrow{\mathtt{spawn}(\kappa, f/n, [\overline{v_n}])} \theta', e' \quad p' \text{ is a fresh pid}}{\langle p, \theta, e\rangle \hookrightarrow \langle p, \theta', e'\{\kappa \mapsto p'\}\rangle \mid \langle p', id, \mathtt{apply}\ f/n\ (\overline{v_n})\rangle} \qquad \text{(Par)}\ \frac{E \hookrightarrow E' \quad \mathtt{pid}(E') \cap \mathtt{pid}(E_1) = \emptyset}{E \mid E_1 \hookrightarrow E' \mid E_1}$$

Figure 7 System rules of standard Core Erlang.

We have $\mathtt{col}(R') = \{\{j_1, j_2\}_k\}$ where k is a fresh key. We now have:

$$\begin{aligned}
(\!|R'|\!) =& \nu j_1, j_2, k\, (\!|R'|\!)_{\mathtt{col}(R')} = \\
=& \nu j_1, j_2, k\, (\!|j_1 : P_1|\!)_{\mathtt{col}(R')} \mid (\!|j_2 : P_2|\!)_{\mathtt{col}(R')} \mid (\!|[R\,; j_1 : \bullet_1 \mid j_2 : \bullet_2]|\!)_{\mathtt{col}(R')} \\
=& \nu j_1, j_2, k\, \langle j_1, \{j_1, j_2\}\rangle \cdot k : P_1 \mid \langle j_2, \{j_1, j_2\}\rangle \cdot k : P_2 \mid \\
& \qquad [(\!|R|\!)_{\mathtt{col}(R')}\,; (\!|j_1 : \bullet_1 \mid j_2 : \bullet_2|\!)_{\mathtt{col}(R')}] \\
=& \nu j_1, j_2, k\, \langle j_1, \{j_1, j_2\}\rangle \cdot k : P_1 \mid \langle j_2, \{j_1, j_2\}\rangle \cdot k : P_2 \mid [R\,; k]
\end{aligned}$$

where R is unchanged since it only contains keys not occurring in $\mathtt{col}(R')$.

Classic and reversible semantics for Core Erlang

This section recalls a reduction semantics for Core Erlang [30] and presents forward and backward rules of the reversible semantics for Core Erlang obtained using approach (Section 4.2).

In Figure 7 we give the reduction semantics for Core Erlang. Rule (Spawn) adds a new process with a fresh pid p', initialised with an empty environment id, into the system. The function $\mathtt{pid}(\cdot)$ used in rule (Par) extracts the set of pids of processes in a given system. It is used to ensure that the pid of the newly spawned process is fresh. We refer to [30] for a detailed description of the rules.

The forward rules of the reversible semantics are given in Figure 8. Rule (F-Par) allows configurations to execute as part of a larger configuration with the additional condition that keys generated by the execution are not part of the parallel configuration.

Backward rules of the reversible semantics are given in Figure 9. Notably, to capture exactly the instances produced by our approach some side conditions would be needed. E.g., in rule B-Par one would need condition $\mathtt{pid}(R') \cap \mathtt{pid}(R'') = \emptyset$. However, such conditions are always satisfied in reachable configurations.

Reversible link semantics for Erlang

In this section we give the additional rules of the reversible link semantics for Erlang (Section 4.3), namely system rule (F-Nrm) as well as rules describing the evaluation of functions `spawn_link`() and `process_flag`(). In rule (Flag), f is a Boolean value.

$$\text{(F-Seq)}\ \frac{\theta, e \xrightarrow{\tau} \theta', e' \quad k_1 \text{ is a fresh key}}{k : \langle p, \theta, e\rangle \twoheadrightarrow k_1 : \langle p, \theta', e'\rangle \mid [k : \langle p, \theta, e\rangle \,;\, k_1 : \bullet_1]}$$

$$\text{(F-Send)}\ \frac{\theta, e \xrightarrow{\texttt{send}(p', v)} \theta', e' \quad k_1, k_2 \text{ are fresh keys}}{k : \langle p, \theta, e\rangle \twoheadrightarrow k_1 : \langle p, \theta', e'\rangle \mid k_2 : (p, p', v) \mid [k : \langle p, \theta, e\rangle \,;\, k_1 : \bullet_1 \mid k_2 : \bullet_2]}$$

$$\text{(F-Rec)}\ \frac{\theta, e \xrightarrow{\texttt{rec}(\kappa, \overline{cl_n})} \theta', e' \quad \text{and} \quad \texttt{matchrec}(\theta, \overline{cl_n}, v) = (\theta_i, e_i) \quad k_1 \text{ is a fresh key}}{k_2 : (p', p, v) \mid k : \langle p, \theta, e\rangle \twoheadrightarrow k_1 : \langle p, \theta' \theta_i, e'\{\kappa \mapsto e_i\}\rangle \mid [k_2 : (p', p, v) \mid k : \langle p, \theta, e\rangle \,;\, k_1 : \bullet_1]}$$

$$\text{(F-Spawn)}\ \frac{\theta, e \xrightarrow{\texttt{spawn}(\kappa, f/n, [\overline{v_n}])} \theta', e' \quad p' \text{ is a fresh pid} \quad \text{and} \quad k_1, k_2 \text{ are fresh keys}}{k : \langle p, \theta, e\rangle \twoheadrightarrow k_1 : \langle p, \theta', e'\{\kappa \mapsto p'\}\rangle \mid k_2 : \langle p', id, \texttt{apply}\ f/n\ (\overline{v_n})\rangle \mid [k : \langle p, \theta, e\rangle \,;\, k_1 : \bullet_1 \mid k_2 : \bullet_2]}$$

$$\text{(F-Self)}\ \frac{\theta, e \xrightarrow{\texttt{self}(\kappa)} \theta', e' \quad k_1 \text{ is a fresh key}}{k : \langle p, \theta, e\rangle \twoheadrightarrow k_1 : \langle p, \theta', e'\{\kappa \mapsto p\}\rangle \mid [k : \langle p, \theta, e\rangle \,;\, k_1 : \bullet_1]}$$

$$\text{(F-Par)}\ \frac{R \twoheadrightarrow R' \quad \texttt{pid}(R') \cap \texttt{pid}(R'') = \emptyset \quad \text{and} \quad (\texttt{key}(R') \setminus \texttt{key}(R)) \cap \texttt{key}(R'') = \emptyset}{R \mid R'' \twoheadrightarrow R' \mid R''}$$

Figure 8 Forward rules of the reversible semantics for Erlang.

$$\text{(B-Seq)}\ k_1 : \langle p, \theta', e'\rangle \mid [k : \langle p, \theta, e\rangle \,;\, k_1 : \bullet_1] \rightsquigarrow k : \langle p, \theta, e\rangle$$

$$\text{(B-Send)}\ k_1 : \langle p, \theta', e'\rangle \mid k_2 : (p, p', v) \mid [k : \langle p, \theta, e\rangle \,;\, k_1 : \bullet_1 \mid k_2 : \bullet_2] \rightsquigarrow k : \langle p, \theta, e\rangle$$

$$\text{(B-Rec)}\ k_1 : \langle p, \theta', e'\rangle \mid [k_2 : (p', p, v) \mid k : \langle p, \theta, e\rangle \,;\, k_1 : \bullet_1] \rightsquigarrow k_2 : (p', p, v) \mid k : \langle p, \theta, e\rangle$$

$$\text{(B-Spawn)}\ k_1 : \langle p, \theta', e'\rangle \mid k_2 : \langle p', id, e''\rangle \mid [k : \langle p, \theta, e\rangle \,;\, k_1 : \bullet_1 \mid k_2 : \bullet_2] \rightsquigarrow k : \langle p, \theta, e\rangle$$

$$\text{(B-Self)}\ k_1 : \langle p, \theta', e'\rangle \mid [k : \langle p, \theta, e\rangle \,;\, k_1 : \bullet_1] \rightsquigarrow k : \langle p, \theta, e\rangle \qquad \text{(B-Par)}\ \frac{R' \rightsquigarrow R}{R' \mid R'' \rightsquigarrow R \mid R''}$$

Figure 9 Backward rules of the reversible semantics for Erlang.

$$\text{(F-Nrm)}\ \frac{l = \{p_1, \ldots, p_m\} \quad 1 \le i \le n \Rightarrow f_i = \texttt{true} \ \wedge\ n+1 \le i \le m \Rightarrow f_i = \texttt{false} \quad h, h_i, j_i \text{ are fresh keys}}{k : \langle p, \theta, v, l, f\rangle \mid \prod_{1\le i\le m} k_i : \langle p_i, \theta_i, e_i, l_i, f_i\rangle \twoheadrightarrow h : \langle p, \theta, v, \emptyset, f\rangle \mid \prod_{1\le i\le n} h_i : \langle p_i, \theta_i, e_i, l_i \setminus \{p\}, f_i\rangle \mid \prod_{1\le i\le n} j_i : (p, p_i, \{'\texttt{EXIT}', p, \texttt{normal}\}) \mid \prod_{n+1\le i\le m} h_i : \langle p_i, \theta_i, e_i, l_i \setminus \{p\}, f_i\rangle \mid [k : \langle p, \theta, v, l, f\rangle \mid \prod_{1\le i\le m} k_i : \langle p_i, \theta_i, e_i, l_i, f_i\rangle \,;\, h : \bullet_h \mid \prod_{1\le i\le m} h_i : \bullet_{h_i} \mid \prod_{1\le i\le n} j_i : \bullet_{j_i}]}$$

$$\text{(Spawn\_Link1)}\ \frac{\theta, e_i \xrightarrow{l} \theta', e_i' \quad i \in \{1, \ldots, n\}}{\theta, \texttt{spawn\_link}(a/n, [\overline{v_{1,i-1}}, e_i, \overline{e_{i+1,n}}]) \xrightarrow{l} \theta', \texttt{spawn\_link}(a/n, [\overline{v_{1,i-1}}, e_i', \overline{e_{i+1,n}}])}$$

$$\text{(Spawn\_Link2)}\ \theta, \texttt{spawn\_link}(a/n, [\overline{v_n}]) \xrightarrow{\texttt{spawn\_link}(\kappa, a/n, [\overline{v_n}])} \theta, \kappa$$

$$\text{(Flag)}\ \theta, \texttt{process\_flag}(trap\_exit, f) \xrightarrow{\texttt{process\_flag}(\kappa, trap\_exit, f)} \theta, \kappa$$

On the Representation of References in the Pi-Calculus

Daniel Hirschkoff
ENS de Lyon, France

Enguerrand Prebet
ENS de Lyon, France

Davide Sangiorgi
Università di Bologna, Italy
INRIA, Sophia Antipolis, France

Abstract

The π-calculus has been advocated as a model to interpret, and give semantics to, languages with higher-order features. Often these languages make use of forms of references (and hence viewing a store as set of references). While translations of references in π-calculi (and CCS) have appeared, the precision of such translations has not been fully investigated. In this paper we address this issue.

We focus on the asynchronous π-calculus ($A\pi$), where translations of references are simpler. We first define $\pi^{\mathtt{ref}}$, an extension of $A\pi$ with references and operators to manipulate them, and illustrate examples of the subtleties of behavioural equivalence in $\pi^{\mathtt{ref}}$. We then consider a translation of $\pi^{\mathtt{ref}}$ into $A\pi$. References of $\pi^{\mathtt{ref}}$ are mapped onto names of $A\pi$ belonging to a dedicated "reference" type. We show how the presence of reference names affects the definition of barbed congruence. We establish full abstraction of the translation w.r.t. barbed congruence and barbed equivalence in the two calculi. We investigate proof techniques for barbed equivalence in $A\pi$, based on two forms of labelled bisimilarities. For one bisimilarity we derive both soundness and completeness; for another, more efficient and involving an inductive "game" on reference names, we derive soundness, leaving completeness open. Finally, we discuss examples of uses of the bisimilarities.

2012 ACM Subject Classification Theory of computation → Semantics and reasoning

Keywords and phrases Process calculus, Bisimulation, Asynchrony, Imperative programming

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.34

Related Version The proofs of most of the results in this paper are presented in a full version of this paper, available at `https://hal.archives-ouvertes.fr/hal-02895654`.

Funding Hirschkoff and Prebet acknowledge support from the European Research Council (ERC) under the European Union's Horizon 2020 programme (CoVeCe, grant agreement No 678157), and from LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" ANR-11-IDEX-0007. Sangiorgi acknowledges support from the MIUR-PRIN project "Analysis of Program Analyses" (ASPRA, ID: `201784YSZ5_004`), and from the European Research Council (ERC) Grant DLV-818616 DIAPASoN.

1 Introduction

The π-calculus has been advocated as a model to interpret, and give semantics to, languages with higher-order features. Often these languages make use of forms of references (and hence viewing a store as set of references). This therefore requires representations of references using the names of the π-calculus. There are strong similarities between the names of the π-calculus and the references of imperative languages. This is evident in the denotational semantics of these languages: the mathematical techniques employed in modelling the π-calculus (e.g., [25, 6]) were originally developed for the semantic description of references. Yet names and

references behave rather differently: receiving from a name is destructive – it consumes a value – whereas reading from a reference is not; a reference has a unique location, whereas a name may be used by several processes both in input and in output; etc. These differences make it unclear if and how interesting properties of imperative languages can be proved via a translation into the π-calculus.

A subset of the π-calculus that often appears in the literature, for its expressive power and elegant theory, is the Asynchronous π-calculus ($A\pi$). $A\pi$ allows one to provide a simpler representation of references, where a reference ℓ storing a value n is just an output message $\overline{\ell}\langle n\rangle$ (in $A\pi$ output is not a prefix, hence it has no process continuation). A process that wishes to access the reference is supposed to make an input at ℓ and then immediately emit a message at ℓ with the new content of the reference. For instance a process reading on the reference and binding its content to x in the continuation P is

$$\ell(x).\,(\overline{\ell}\langle x\rangle \mid P) \ .$$

Another reason that makes this representation of references in $A\pi$ interesting is the bisimilarity of $A\pi$, called *asynchronous bisimilarity*. It differs from standard bisimilarity in the input clause, in which a transition $P \xrightarrow{n\langle m\rangle} P'$ (where P is receiving m on n) can be answered by a bisimilar process Q thus:

$$\overline{n}\langle m\rangle \mid Q \Rightarrow Q' \qquad (*)$$

(provided P' and Q' are bisimilar), where $\Rightarrow$ stands for zero or several internal communication steps. Intuitively, Q does not necessarily perform an input on n in response to the transition done by P. To see why this clause could be interesting with references, consider a process that performs a *useless read* on a reference ℓ and then continues as P_2; in a language with references this would be equivalent to P_2 itself. When written in $A\pi$, the process with the useless read becomes $P_1 \stackrel{\text{def}}{=} \ell(x).\,(\overline{\ell}\langle x\rangle \mid P_2)$ where x does not appear in P_2. In ordinary bisimilarity, P_1 is immediately distinguished from P_2, as the latter cannot answer the input transition $P_1 \xrightarrow{\ell\langle n\rangle} \overline{\ell}\langle n\rangle \mid P_2$. However, the answer is possible using the clause $(*)$, as we have

$$\overline{\ell}\langle n\rangle \mid P_2 \Rightarrow \overline{\ell}\langle n\rangle \mid P_2 \ .$$

We are not aware of studies that investigate the faithfulness of the above representation of references in $A\pi$. In this paper we address this issue. For this, we first define $\pi^{\mathtt{ref}}$, an extension of $A\pi$ with references and operators to manipulate them. We then consider a translation of $\pi^{\mathtt{ref}}$ into $A\pi$ and:

- we study the properties of this translation;
- we establish proof techniques on $A\pi$ to reason about references.

The calculus with references, $\pi^{\mathtt{ref}}$, has constructs for reading from a reference, writing on a reference, and a swap operation for atomically reading on a reference and placing a new value onto it. Modern computer architectures offer hardware instructions similar to swap, e.g., test-and-set, or control-and-swap constructs to atomically check and modify the content of a register. These constructs are important to tame the access to shared resources. In distributed systems, swap can be used to solve the consensus problem with two parallel processes, whereas simple registers cannot [8].

The swap construct is also suggested by the translation of references into $A\pi$. The pattern for accessing a reference ℓ is $\ell(x).\,(\overline{\ell}\langle n\rangle \mid P)$. This yields four cases, depending on whether x is used in P and whether x is equal to n:

	$n \neq x$	$n = x$
x free in P	swap	read
x not free in P	write	useless read

We define a type system in $A\pi$ to capture the intended pattern of usage of names that represent references, called *reference names*, in particular the property that there is always a unique output message available at these names. The type system has linearity features similar to π-calculus type systems for locks [13] or for receptiveness [22].

Imposing a type system has consequences on behavioural equivalences. Since the set of legal contexts becomes smaller, the behavioural equivalence itself becomes coarser. For instance, in the case of reference names, a process P is supposed to be tested only in a context that guarantees that all references mentioned in P are "allocated" (thus, an input at a reference name ℓ is never "stuck", as an output message at ℓ must always exist). A consequence of these is a read in which the value read is not used is irrelevant (see formally law (1)).

In both calculi, as behavioural equivalence we use *barbed congruence* and *barbed equivalence*. These equivalences equate processes which, roughly, in all contexts give rise to "matching reductions".

We establish an operational correspondence between the behaviour of a process in $\pi^{\mathtt{ref}}$ and its encoding in $A\pi$, and from this we establish full abstraction of the translation of $\pi^{\mathtt{ref}}$ into $A\pi$ with respect to both barbed equivalence and barbed congruence in the two calculi. We then investigate proof techniques for barbed equivalence in $A\pi$, based on two forms of labelled bisimilarities. For one bisimilarity we derive both soundness and completeness. This bisimilarity is similar to, but not the same as, asynchronous bisimilarity. For instance, it is defined on "reference-closed" processes (intuitively, processes in which all references are allocated); therefore inputs on reference names from the tested processes are not visible (because such inputs are supposed to consume the unique output message at that reference that is present in the tested processes). The output clause of bisimilarity on reference names is also different, as we have to make sure that the observer respects the pattern of usage for reference names; thus the observer consuming the output message on a reference name ℓ should immediately re-install an output on ℓ.

The second bisimilarity is more efficient because it does not require processes to be "reference-closed". Thus output messages on reference names consumed by the observer need not be immediately re-installed. However sometimes access to a certain reference is needed by a process in order to answer the bisimulation challenge from the other process. And depending on the content of such references, further accesses to other references may be needed. Since we wish to add only the needed references, this introduces an inductive game, in which a player requires a reference and the other player specifies the content of such reference, within the coinductive game of bisimulation. We show that the resulting bisimilarity is sound, and leave completeness as an open problem. Finally, we discuss examples of uses of the bisimilarities.

Related Work. The classic encoding of references in the π-calculus [16] follows their encoding into CCS [15]: a reference is a stateful recursive process, which may be interrogated using two names, one for read operations, the other for write operations. Properties of this encoding have been explored [20], comparing the π-calculus to *Concurrent Idealised Algol* [3], an extension of Idealised Algol [19] with shared variables concurrency. The encoding has been shown to be sound but not complete.

Many works have studied the effect of type systems on behavioural equivalence, formalised using both barbed congruence and labelled bisimilarity. (See the references in the books [24, 7]). To our knowledge, no such study has been done regarding the discipline for reference names which we use in this work. This discipline bears similarities with receptiveness [22], which is also related to the results in [23, 14]. We can also remark that our notion of complete processes is reminiscent of the notion of catalysers used by Dezani et al. [5] in session types to enforce progress.

Section 5 discusses further related work.

Paper outline. In Section 2, we introduce $\pi^{\mathtt{ref}}$ and discuss examples of behavioural equivalences between $\pi^{\mathtt{ref}}$ processes. In Section 3 we present $A\pi$ with reference names, using a type system that captures the usage of such names. We show the encoding of $\pi^{\mathtt{ref}}$ into such $A\pi$ and prove its full abstraction for barbed equivalence and congruence. In Section 4 we introduce the two new labelled bisimilarities for $A\pi$, we establish soundness and completeness for one and soundness for the other (we conjecture that also completeness holds), and present a useful "up-to" technique for the second one. Finally we illustrate the benefits of using the proof techniques based on the labelled bisimilarities of $A\pi$ on some examples.

The proofs of most of the results in this work are presented in a full version of this paper [9].

2 Asynchronous Processes Accessing References: $\pi^{\mathtt{ref}}$

In this section, we introduce $\pi^{\mathtt{ref}}$, the asynchronous π-calculus extended with primitives to interact with memory locations.

2.1 Syntax and Semantics

We assume an infinite set $\mathtt{Names}$ of *names* and a distinct infinite set $\mathtt{Refs}$ of *references*. These sets do not contain the special symbol $\star$, that stands for the constant "unit". We use $a, b, c, \ldots, p, q, \ldots$ to range over $\mathtt{Names}$; $\ell, \ldots$ to range over $\mathtt{Refs}$; and $n, m, \ldots, x, y, \ldots$ to range over $\mathtt{All} \stackrel{\text{def}}{=} \mathtt{Names} \cup \mathtt{Refs} \cup \{\star\}$. The grammar for the calculus $\pi^{\mathtt{ref}}$ is the following; for simplicity, we develop our theory on the monadic calculus (one value at a time is handled).

$$\begin{aligned} P \quad ::= \quad & \mathbf{0} \mid a(x).\,P \mid \overline{a}\langle n\rangle \mid !P \mid P_1 \mid P_2 \mid (\boldsymbol{\nu} a)P \mid [n = m]P \\ & \mid (\boldsymbol{\nu}\ell = n)P \mid \ell \triangleleft n.\,P \mid \ell \triangleright (x).\,P \mid \ell \bowtie n(x).\,P \end{aligned}$$

The operators in the first line are the standard π-calculus constructs for the inactive process, input, asynchronous output, replication, parallel composition, name restriction, and matching (however matching here is defined on both names and references). In the second line, we find the operators to handle references: reference restriction, or allocation (creating a new reference ℓ with initial value n), write (setting the content of ℓ to n), read (reading in x the value of ℓ), swap (atomically reading on x and replacing the content of the reference with n).

As usual, we often omit $\mathbf{0}$, and abbreviate $\overline{a}\langle\star\rangle$ as $\overline{a}$ (and similarly for inputs $a.\,P$). We use a tilde, $\widetilde{\cdot}$, for (possibly empty) finite tuples; then $(\boldsymbol{\nu}\widetilde{a})$ is a sequence of restrictions; and $(\boldsymbol{\nu}\widetilde{L})$ a sequence of reference allocations (i.e., a piece of store), using L to represent a single allocation such as $\ell = n$. Given the binders $(\boldsymbol{\nu} a)P$ and $(\boldsymbol{\nu}\ell = n)P$ (for a and ℓ, respectively), $a(x).\,P$, $\ell \triangleright (x).\,P$ and $\ell \bowtie n(x)$ (for x), we define $\mathrm{bn}(O)$, $\mathrm{fn}(O)$ (resp. $\mathrm{fr}(O)$, $\mathrm{br}(O)$), for the *bound* and *free names* (resp. *references*) of some object O (process, action, etc.). The set of *names* of O is defined as the union of its free and bound names; and analogously for *references*. In $a(x).\,P$ or $\overline{a}\langle x\rangle$, name a is the *subject* whereas x is the *object*.

$$\texttt{R-Equiv:}\ \frac{P \equiv P' \qquad P' \longrightarrow Q' \qquad Q' \equiv Q}{P \longrightarrow Q} \qquad \texttt{R-Ctxt:}\ \frac{P \longrightarrow P'}{E[P] \longrightarrow E[Q]}$$

$$\texttt{R-Comm:}\ \frac{}{a(x).\,P \mid \overline{a}\langle n\rangle \longrightarrow P\{n/x\}}$$

$$\texttt{R-Read:}\ \frac{\ell, n \notin \mathrm{br}(\boldsymbol{\nu}\widetilde{L})}{(\boldsymbol{\nu}\ell = n)(\boldsymbol{\nu}\widetilde{L})(\ell \triangleright (x).\,P \mid Q) \longrightarrow (\boldsymbol{\nu}\ell = n)(\boldsymbol{\nu}\widetilde{L})(P\{n/x\} \mid Q)}$$

$$\texttt{R-Write:}\ \frac{\ell, n \notin \mathrm{br}(\boldsymbol{\nu}\widetilde{L})}{(\boldsymbol{\nu}\ell = m)(\boldsymbol{\nu}\widetilde{L})(\ell \triangleleft n.\,P \mid Q) \longrightarrow (\boldsymbol{\nu}\ell = n)(\boldsymbol{\nu}\widetilde{L})(P \mid Q)}$$

$$\texttt{R-Swap:}\ \frac{\ell, n, m \notin \mathrm{br}(\boldsymbol{\nu}\widetilde{L})}{(\boldsymbol{\nu}\ell = m)(\boldsymbol{\nu}\widetilde{L})(\ell \bowtie n(x).\,P \mid Q) \longrightarrow (\boldsymbol{\nu}\ell = n)(\boldsymbol{\nu}\widetilde{L})(P\{m/x\} \mid Q)}$$

Figure 1 $\pi^{\mathtt{ref}}$, reduction relation.

We assume the calculus is simply-typed. Any basic type system for the π-calculus would do. In this paper, we assume Milner's *sorting*: names and references are partitioned into a collection of *types* (or sorts). Name types contain names, and reference types contain references. Then a sorting function maps types onto types. If a name type s is mapped onto a type t, this means that names in s may only carry, or contain, objects in t; if s is a reference type then only objects of type t may be stored in s. We shall assume that there is a sorting system under which all processes we manipulate are well-typed. For simplicity we use simple types; e.g., the sorting is non-recursive (meaning that the graph that represents the sorting function, in which the nodes are the types, does not contain cycles). In the remainder we assume that all objects (processes, contexts, actions, etc.) respect a given sorting.

The definition of structural congruence, $\equiv$, is the expected one from the π-calculus, treating the $(\boldsymbol{\nu}\ell = n)$ operator like a restriction (see Appendix B).

Contexts, ranged over by C, are process expressions with a hole $[\,]$ in it. We write $C[P]$ for the process obtained by replacing the hole in C with P. *Active* (or *evaluation*) *contexts*, ranged over by E, are given by:

$$E \quad ::= \quad [\,] \mid E \mid P \mid (\boldsymbol{\nu}a)E \mid (\boldsymbol{\nu}\ell = n)E \ .$$

The reduction relation $\longrightarrow$ is presented in Figure 1. It uses *active contexts* to isolate the subpart of the term that is active in a reduction. We write $\Longrightarrow$ for the "multistep" version of $\longrightarrow$, whereby $P \Longrightarrow P'$ if P may become P' after a (possibly empty) sequence of reductions. Rules `R-Read`, `R-Write` and `R-Swap` in Figure 1 describe an interaction between the process and a reference ℓ. These rules make use of a store $(\boldsymbol{\nu}\widetilde{L})$; this is necessary because there might be references that depend on ℓ, and as such cannot be moved past the restriction on ℓ. An example is $(\boldsymbol{\nu}\ell = a)(\boldsymbol{\nu}\ell' = \ell)\ell \triangleleft b.\,P$: the write operation is executed by applying rule `R-Write`, with $(\boldsymbol{\nu}\widetilde{L}) = (\boldsymbol{\nu}\ell' = \ell)$, as the restriction on ℓ' cannot be brought above the restriction on ℓ. We recall that $\mathrm{br}(\boldsymbol{\nu}\widetilde{L})$ are the references bound by the $\boldsymbol{\nu}$.

As usual in concurrent calculi, the reference behavioural equivalence will be barbed congruence (in its variant sometimes called *reduction-closed barbed congruence*), a form of bisimulation on reduction that uses closure under contexts and simple observables. In the

context closure, however, we make sure that all references mentioned in the tested process have been allocated. As often in π-calculi, we also consider *barbed equivalence*, that uses only active contexts.

P exhibits a barb at a (so a is in `Names`), written $P \downarrow_{\overline{a}}$, if $P \equiv (\boldsymbol{\nu}\widetilde{b})(\boldsymbol{\nu}\widetilde{L})(\overline{a}\langle m\rangle \mid P')$ with $a \notin \widetilde{b}$. We write $P \Downarrow_{\overline{a}}$ if $P \Longrightarrow P_1$ and $P_1 \downarrow_{\overline{a}}$ for some P_1.

▶ **Definition 1.** *Given a relation $\mathcal{R}$ on processes, and $P\ \mathcal{R}\ Q$, we say that P, Q (in $\mathcal{R}$) are*
- **closed under reductions** *if $P \longrightarrow P'$ implies there is Q' s.t. $Q \Longrightarrow Q'$ and $P'\ \mathcal{R}\ Q'$;*
- **preserved by a set $\mathcal{C}$ of contexts** *if $C[P]\ \mathcal{R}\ C[Q]$ for all $C \in \mathcal{C}$;*
- **compatible on barbs** *if $P \downarrow_{\overline{a}}$ implies $Q \Downarrow_{\overline{a}}$, for all a.*

A process P is *reference-closed* if $\mathrm{fr}(P) = \emptyset$. A context C is *closing on the references* of a process P if $C[P]$ is reference-closed; similarly, C is closing on the references of P, Q if it closing on the references of both P and Q. Since reductions may only decrease the set of free names of a process, the property of being reference-closed is preserved by reductions.

▶ **Definition 2** (Barbed congruence and equivalence in $\pi^{\mathtt{ref}}$)**.** Barbed congruence *is the largest symmetric relation $\cong_{\mathrm{ref}}$ in $\pi^{\mathtt{ref}}$ such that whenever $P\ \mathcal{R}\ Q$ then P, Q are: closed under reductions if P, Q are reference-closed; preserved by the contexts that are closing on references for P, Q; compatible on barbs if P, Q are reference-closed.* Barbed equivalence, $\cong^{\mathrm{e}}_{\mathrm{ref}}$, *is defined in the same way, but using active contexts in place of all contexts.*

The restriction to closing contexts (as opposed to arbitrary contexts) yields laws such as

$$\ell \triangleright (x).\mathrm{P} \cong_{\mathrm{ref}} \mathrm{P}, \tag{1}$$

whenever $x \notin \mathrm{fn}(P)$. Closing contexts ensure that the reading on ℓ is not blocking, and therefore possible observables in P are visible on both sides.

As the quantification on contexts refers to the free references of the tested processes, transitivity of barbed congruence and equivalence requires some care. As usual in the π-calculus, barbed equivalence is not preserved by the input construct, and the closure of barbed equivalence under all (well-typed) substitutions coincides with barbed congruence.

2.2 Behavioural Equivalence in $\pi^{\mathtt{ref}}$: Examples

We present a few examples that illustrate some subtleties of behavioural equivalence in $\pi^{\mathtt{ref}}$. These examples will be formally treated in Section 4.2 for Examples 3 and 4, and in Appendix A for Examples 5 and 6.

The first example shows that processes may be equivalent even though the store is public and holds different values. (In the example, the reference ℓ is actually restricted, but the process P underneath the restriction, representing an observer, is arbitrary).

▶ **Example 3.** For any P, we have $P_1 \cong_{\mathrm{ref}} P_2$, for

$$P_1 \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu}\ell = a)(P \mid !\ell \triangleleft a \mid !\ell \triangleleft b) \qquad P_2 \stackrel{\mathrm{def}}{=} (\boldsymbol{\nu}\ell = b)(P \mid !\ell \triangleleft a \mid !\ell \triangleleft b)$$

In the second example, the write on top of P is not blocking, provided that the same writing is anyhow possible, and provided that the current value of the store can be recorded.

▶ **Example 4.** We have $P_1 \cong_{\mathrm{ref}} P_2$, for

$$P_1 \stackrel{\mathrm{def}}{=} \ell \triangleleft b.\, P \mid !\ell \triangleleft b \mid !\ell \triangleright (x).\, \ell \triangleleft x \qquad P_2 \stackrel{\mathrm{def}}{=} P \mid !\ell \triangleleft b \mid !\ell \triangleright (x).\, \ell \triangleleft x$$

On the left, it would seem that P runs under a store in which ℓ contains b; whereas on the right, P could also run under the initial store, where ℓ could contain a different value, say a. However the component $!\ell \triangleright (x).\, \ell \triangleleft x$ allows us to store a in x and then write it back later, thus overwriting b.

▶ **Example 5.** We have $P_s \not\cong^{\mathrm{e}}_{\mathrm{ref}} Q_s$, where

$$P_s \stackrel{\text{def}}{=} (\boldsymbol{\nu} t)\ell \triangleleft b.\, (\bar{t} \mid !t.\, \ell \triangleleft a.\, (\bar{c} \mid \ell \triangleleft b.\, (\bar{t} \mid c))) \qquad Q_s \stackrel{\text{def}}{=} (\boldsymbol{\nu} t)\ell \triangleleft a.\, (\bar{t} \mid !t.\, \ell \triangleleft b.\, (\bar{c} \mid \ell \triangleleft a.\, (\bar{t} \mid c)))$$

The discriminating context being large, the formal discussion is moved in Appendix A. Intuitively, P_s and Q_s are refinements of the processes in Example 3, in that their initial writes store different values on the reference ℓ, but both processes maintain the capability of writing both values in ℓ. The difference with Example 3 are the additional inputs and outputs on name c, which are generated along the transitions. These allow an observer to distinguish P_s from Q_s by exploiting the swap construct. We informally explain the reason. If the two processes have written the same value, say a, in ℓ, then Q_s has generated the same number of inputs and outputs on c, while P_s must have generated an extra output. An observer can use swap to read the content of ℓ, so to check that the value is indeed a, and write back a fresh name, say e. Now the observer can tell that P_s has an extra output on c: process Q_s cannot add a further output, because this would require overwriting e in ℓ, which can be tested by the observer at the end.

We have seen in Example 3 two equivalent processes whose initial store (a single reference) is different. The equivalence holds intuitively because the values that the two processes can store are the same. Using two references, it is possible to complicate the example. In Example 6, the processes are equivalent and yet the pairs of values that may be simultaneously stored in the two references are different for the two processes. For each reference separately, the set of possible values is the same. But setting a reference to a certain value implies first having set the other reference to some specific values. (The processes could be distinguished if an observer had the possibility to simultaneously read the two references.)

▶ **Example 6.** Consider two references ℓ_1, ℓ_2 where booleans (represented as 0,1 below) can be stored. Then for any P, we have $P_1 \cong_{\mathrm{ref}} P_2$, where

$$P_1 \stackrel{def}{=} (\boldsymbol{\nu} \ell_1 = 0, \ell_2 = 0)(P \mid (\boldsymbol{\nu} t)(\bar{t} \mid !t.\, \ell_1 \triangleleft 1.\, \ell_1 \triangleleft 0.\, \ell_2 \triangleleft 1.\, \ell_2 \triangleleft 0.\, \bar{t}))$$
$$P_2 \stackrel{def}{=} (\boldsymbol{\nu} \ell_1 = 0, \ell_2 = 0)(P \mid (\boldsymbol{\nu} t)(\bar{t} \mid !t.\, \ell_1 \triangleleft 1.\, \ell_2 \triangleleft 1.\, \ell_1 \triangleleft 0.\, \ell_2 \triangleleft 0.\, \bar{t}))$$

P_1 and P_2 can write 0 and 1 in references ℓ_1 and ℓ_2, but not in the same order. By doing so, we see that if P_1 loops, the content of ℓ_1 and ℓ_2 will evolve thus: $(0,0) \to (1,0) \to (0,0) \to (0,1) \to (0,0)$, while for P_2 the loop is different: $(0,0) \to (1,0) \to (1,1) \to (0,1) \to (0,0)$.

In particular, P_2 can always go through the state $(1,1)$, independently of the transitions of P, while P_1 cannot, in general, reach this state.

The example above relies on the fact that the domain of possible values for ℓ_1 and ℓ_2 is finite. A more sophisticated example, without such assumption, is given in the Appendix A.

3 Mapping $\pi^{\mathtt{ref}}$ onto the Asynchronous π-calculus

We present the encoding of $\pi^{\mathtt{ref}}$ into $A\pi$, which follows the folklore encoding of references into $A\pi$.

3.1 The Asynchronous π-calculus

Below is the grammar of the asynchronous π-calculus, $A\pi$; we reuse all notations from $\pi^{\mathtt{ref}}$.

$$P \quad ::= \quad \mathbf{0} \mid n(x).\,P \mid !P \mid \overline{n}\langle m\rangle \mid P_1 \mid P_2 \mid (\boldsymbol{\nu} n)P \mid [n=m]P$$

The reduction semantics, as well as barbed equivalence and congruence (written $\cong_{\mathrm{a}}^{\mathrm{e}}$ and $\cong_{\mathrm{a}}$, respectively), are standard (defined as in $\pi^{\mathtt{ref}}$, and recalled in Appendix B). We recall the standard definition of asynchronous bisimilarity, $\approx_{\mathrm{a}}$, from [1]. To define $\approx_{\mathrm{a}}$, as well as the other forms of bisimilarity we introduce in Section 4, we rely on the early transition system for $A\pi$. In this LTS, which is presented in Appendix B labels are either free inputs of the form $n\langle m\rangle$ (reception of name m on n), output ($\overline{n}\langle m\rangle$), bound output ($(\boldsymbol{\nu} m)\overline{n}\langle m\rangle$) or internal communication (τ).

▶ **Definition 7.** *A symmetric relation $\mathcal{R}$ between processes is an asynchronous bisimulation if whenever $P\,\mathcal{R}\,Q$ and $P \xrightarrow{\mu} P'$, one of these two clauses hold:*

- *there is Q' such that $Q \stackrel{\hat{\mu}}{\Rightarrow} Q'$ and $P'\,\mathcal{R}\,Q'$;*
- *$\mu = n\langle m\rangle$ and there is Q' such that $Q \mid \overline{n}\langle m\rangle \Rightarrow Q'$ and $P'\,\mathcal{R}\,Q'$.*

Asynchronous bisimilarity, $\approx_{\mathrm{a}}$, is the largest asynchronous bisimulation.

▶ **Theorem 8** ([1])**.** *Relations $\cong_{\mathrm{a}}^{\mathrm{e}}$ and $\approx_{\mathrm{a}}$ coincide.*

3.2 Encoding $\pi^{\mathtt{ref}}$

In π-calculi such as $A\pi$, there are no references, only names. To make the encoding easier to read, we assume however that the set of names contains the set of references $\{\ell, \cdots\}$ of $\pi^{\mathtt{ref}}$. We call such names *reference names*, and call *plain names* the remaining names. Reference names will be used to represent the references of $\pi^{\mathtt{ref}}$.

The encoding $\mathcal{E}[\![\cdot]\!]$, from $\pi^{\mathtt{ref}}$ to $A\pi$, is a homomorphism on all operators (thus, e.g., $\mathcal{E}[\![P_1 \mid P_2]\!] \stackrel{\text{def}}{=} \mathcal{E}[\![P_1]\!] \mid \mathcal{E}[\![P_2]\!]$, and $\mathcal{E}[\![a(m).\,P]\!] \stackrel{\text{def}}{=} a(m).\,\mathcal{E}[\![P]\!]$), except for reference constructs for which we have:

$$\mathcal{E}[\![(\boldsymbol{\nu}\ell = m).\,P]\!] \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell)(\overline{\ell}\langle m\rangle \mid \mathcal{E}[\![P]\!]) \qquad \mathcal{E}[\![\ell \triangleleft v.\,P]\!] \stackrel{\text{def}}{=} \ell(\_).\,(\overline{\ell}\langle v\rangle \mid \mathcal{E}[\![P]\!])$$

$$\mathcal{E}[\![\ell \triangleright (x).\,P]\!] \stackrel{\text{def}}{=} \ell(x).\,(\overline{\ell}\langle x\rangle \mid \mathcal{E}[\![P]\!]) \qquad \mathcal{E}[\![\ell \bowtie n(x).\,P]\!] \stackrel{\text{def}}{=} \ell(x).\,(\overline{\ell}\langle n\rangle \mid \mathcal{E}[\![P]\!])$$

(We write $\ell(\_).\,Q$ for an input whose bound name does not appear in Q.) In the encoding, an object m stored at reference ℓ is represented as a message $\overline{\ell}\langle m\rangle$. Accordingly, the encoding of a write $\ell \triangleleft v.\,P$ is $\ell(\_).\,(\overline{\ell}\langle v\rangle \mid \mathcal{E}[\![P]\!])$, meaning that the process acquires the current message at ℓ (which is thus not available anymore) and replaces it with an output with the new value. The encoding of a read $\ell \triangleright (x).\,P$ follows a similar pattern, this time however the same value is received and emitted: $\ell(x).\,(\overline{\ell}\langle x\rangle \mid P)$. The encoding of swap combines the two patterns.

3.3 Types and Behavioural Equivalences with Reference Names

To prove a full abstraction property for the encoding, we use types to formalise the behavioural difference between reference names and plain names in the asynchronous π-calculus. The typing discipline can be added onto any basic type system for the π-calculus. As for $\pi^{\mathtt{ref}}$, we follow Milner's sorting. The types of the sorting impose a partition on the two sets of names (reference names and plain names). Thus we assume such a sorting, under which all processes are well-typed. We separate the base type system (Milner's sorting) from the typing rules for reference names so as to show the essence of the latter rules. Accordingly, we only present the additional typing constraints for reference names.

$$\texttt{TNil}\ \frac{}{\emptyset \vdash \mathbf{0}} \qquad \texttt{TOut}\ \frac{}{\emptyset \vdash \overline{a}\langle m\rangle} \qquad \texttt{TInp}\ \frac{\emptyset \vdash P}{\emptyset \vdash a(x).\,P} \qquad \texttt{TRep}\ \frac{\emptyset \vdash P}{\emptyset \vdash\, !P}$$

$$\texttt{TPar}\ \frac{\Delta_1 \vdash P \qquad \Delta_2 \vdash Q}{\Delta_1 \uplus \Delta_2 \vdash P \mid Q} \qquad \texttt{TResN}\ \frac{\Delta \vdash P}{\Delta \vdash (\boldsymbol{\nu} a)P} \qquad \texttt{TResR}\ \frac{\Delta, \ell \vdash P}{\Delta \vdash (\boldsymbol{\nu} \ell)P}$$

$$\texttt{TRefO}\ \frac{}{\ell \vdash \overline{\ell}\langle m\rangle} \qquad \texttt{TRefI}\ \frac{\ell \vdash P}{\emptyset \vdash \ell(x).\,P}$$

Figure 2 Typing conditions for reference names in $A\pi$ processes.

We write: `RefTypes` for the the set of reference types (i.e., types that contain reference names); $\texttt{Type}(n)$ is the type of name n; $\texttt{ObType}(n)$ is the type of the objects of n (i.e., the type of the names that may be carried at n). For example in well-typed processes such as $\overline{n}\langle m\rangle$ and $n(m).\,P$, name m will be of type $\texttt{ObType}(n)$.

Notations. *We use $\ell, \ldots$ to range over reference names, $a, b, \ldots$ over plain names, $n, m, \ldots$ over the set of all names. Δ ranges over finite sets of reference names. We sometimes write $\Delta - x$ as abbreviation for $\Delta - \{x\}$. Moreover $\Delta_1 \uplus \Delta_2$ is defined only when $\Delta_1 \cap \Delta_2 = \emptyset$, in which case it is $\Delta_1 \cup \Delta_2$; we write Δ, x for $\Delta \uplus \{x\}$.*

The type system is presented in Figure 2. Judgements have the form $\Delta \vdash P$, where P is an $A\pi$ process. Rule `TRefO` along with Rule `TPar` ensures that every reference names in Δ appears in subject of exactly one unguarded output. Rule `TResR` ensures that new reference names are always in Δ while Rule `TRefI` ensures that Δ is constant after a communication between references (by re-emitting an output after one has been consumed).

Intuitively, if $\Delta \vdash P$, then P must make available the names in Δ *immediately* and *exactly once* in output subject position. We say that ℓ is *output receptive* in P if there is exactly one unguarded output at ℓ, and moreover this output is not underneath a replication. Then $\Delta \vdash P$ holds if

- any $\ell \in \Delta$ is output receptive in P;
- in any subterm of P of the form $(\boldsymbol{\nu}\ell')Q$ or $\ell'(m).\,Q$, name ℓ' is output receptive in Q.

This intuition is formalised in Lemma 9, and in Proposition 10 that relates types and operational semantics.

Typing is important because it allows us to derive the required behavioural equivalences. For instance, allowing parallel composition with the ill-typed process $\ell(x).\,\mathbf{0}$ would invalidate barbed equivalence between the (translations of the) terms in law (1).

In the remainder of the paper, it is assumed that all processes are *well typed*, meaning that each process P obeys the underlying sorting system and that there is Δ s.t. $\Delta \vdash P$ holds. Two processes P, Q are *type-compatible* if both $\Delta \vdash P$ and $\Delta \vdash Q$, for some Δ; we write $\Delta \vdash P, Q$ in this case. *In the remainder of the paper, all relations are on pairs of type-compatible processes. Similarly, all compositions (i.e., of a context with processes) and actions are well-typed.*

The type system satisfies standard properties, like uniqueness of typing ($\Delta \vdash P$ and $\Delta' \vdash P$ imply $\Delta = \Delta'$), and preservation by structural congruence ($P \equiv Q$ and $\Delta \vdash P$ imply $\Delta \vdash Q$). As claimed above, if $\Delta \vdash P$, then names in Δ are output receptive:

▶ **Lemma 9.** *If $\Delta, \ell \vdash P$ then $P \equiv (\boldsymbol{\nu}\tilde{n})(\overline{\ell}\langle m\rangle \mid Q)$, with $\ell \notin \tilde{n}$, and there is no unguarded output at ℓ in Q.*

The following standard property relies on the standard LTS for $A\pi$, which is given in Appendix B.

▶ **Proposition 10** (Subject reduction). *If $\Delta \vdash P$ and $P \xrightarrow{\mu} P'$, then*
1. *if $\mu = \tau$, $\mu = \overline{a}\langle m\rangle$, $\mu = a\langle m\rangle$ or $\mu = (\boldsymbol{\nu}b)\overline{a}\langle b\rangle$, then $\Delta \vdash P'$.*
2. *if $\mu = (\boldsymbol{\nu}\ell)\overline{a}\langle\ell\rangle$ then $\Delta, \ell \vdash P'$.*
3. *if $\mu = \ell\langle m\rangle$ and $\ell \notin \Delta$, then $\Delta, \ell \vdash P'$*
4. *if $\ell \notin \Delta$, then $\Delta, \ell \vdash P \mid \overline{\ell}\langle m\rangle$.*
5. *if $\mu = \overline{\ell}\langle m\rangle$ or $\mu = (\boldsymbol{\nu}b)\overline{\ell}\langle b\rangle$, then $\Delta - \ell \vdash P'$.*
6. *if $\mu = (\boldsymbol{\nu}\ell')\overline{\ell}\langle\ell'\rangle$, then $(\Delta - \ell), \ell' \vdash P'$.*

We can remark that in case 3, we have $\ell \notin \Delta$, as otherwise the context would not be able to trigger an input (since, by typing, it could not generate an output on ℓ).

Barbed congruence. As usual in typed calculi, the definitions of the barbed relations take typing into account, so that the composition of a context and a process be well-typed. In the case of reference names, an additional ingredient has to be taken into account, namely the accessibility of reference names. If a process has the possibility of accessing a reference, then a context in which the process is tested should guarantee the availability of that reference. For this, we define the notion of *completing context* and *complete* process. Then, roughly, barbed congruence becomes "barbed congruence under all completing contexts".

A process P is *complete* if each reference name that appears free in P is "allocated" in P. We write $\mathtt{frn}(P)$ for the set of free reference names in P.

▶ **Definition 11** (Open references and complete processes). *The* open references *of P such that $\Delta \vdash P$ are the names in $\mathtt{frn}(P) \setminus \Delta$; similarly the open references of processes $P_1, \ldots, P_n$ is the union of the open references of the P_i's. P is* complete *if it contains no open reference. $\mathtt{frn}(P) \subseteq \Delta$ and $\Delta \vdash P$, for some Δ.*

A context C is completing for *P if $C[P]$ is complete.*

(Note that an *$A\pi$ complete process* might have free reference names, if these are not open references; in contrast, a *$\pi^{\mathtt{ref}}$ reference-closed process* does not have free references.)

▶ **Lemma 12.** *P is complete iff $\emptyset \vdash (\boldsymbol{\nu}\widetilde{n})P$ where $\widetilde{n} \stackrel{def}{=} \mathtt{frn}(P)$.*

Completing contexts are the only contexts in which processes should be tested. We constrain the definitions of typed barbed congruence and equivalence accordingly. The grammar for the active contexts in $A\pi$ is as expected:

$$E \quad ::= \quad [\,] \;\big|\; E \mid P \;\big|\; (\boldsymbol{\nu}n)E \ .$$

▶ **Definition 13** (Barbed congruence and equivalence in $A\pi$ with reference names). Barbed congruence *is the largest symmetric relation $\cong_{\mathrm{Arn}}$ in $A\pi$ such that whenever $P\,\mathcal{R}\,Q$ then P, Q are: closed under reductions whenever they are complete; closed under the contexts that are completing for P, Q; compatible on barbs whenever they are complete.* Barbed equivalence, *$\cong^{\mathrm{e}}_{\mathrm{Arn}}$, is defined analogously except that one uses active contexts in place of all contexts.*

This typed barbed equivalence is the behavioural equivalence we are mainly interested in. The reference name discipline weakens the requirements on names (by limiting the number of legal contexts), hence the corresponding typed barbed relation is coarser. We are not aware of existing works in the literature that study the impact of the reference name discipline on behavioural equivalence.

▶ **Lemma 14.** *For all compatible P, Q, $P \cong^{\mathrm{e}}_{\mathrm{a}} Q$ (and hence also $P \approx_{\mathrm{a}} Q$) implies $P \cong^{\mathrm{e}}_{\mathrm{Arn}} Q$.*

We show in Section 4 that the inclusion is strict.

3.4 Validating the Encoding

We now show that the two notions of barbed congruence coincide via the encoding.

▶ **Theorem 15** (Operational correspondence). *If $P \longrightarrow P'$, then $\mathcal{E}[\![P]\!] \longrightarrow \mathcal{E}[\![P']\!]$. Conversely, if $\mathcal{E}[\![P]\!] \longrightarrow Q$, then $P \longrightarrow P'$, with $\mathcal{E}[\![P']\!] \equiv Q$.*

The next lemma shows that, up to asynchronous bisimilarity, we can "read back" well-typed processes in $A\pi$, via the encoding, as processes in $\pi^{\mathtt{ref}}$. And similarly for contexts.

▶ **Lemma 16.** *If $\emptyset \vdash P$, then there exists R in $\pi^{\mathtt{ref}}$ such that $\mathcal{E}[\![R]\!] \approx_{\mathrm{a}} P$.*

Theorem 15 and Lemma 16 are the main ingredients to derive the following theorem:

▶ **Theorem 17** (Full abstraction). *For any P, Q in $\pi^{\mathtt{ref}}$: $P \cong_{\mathrm{ref}} Q$ iff $\mathcal{E}[\![P]\!] \cong_{\mathrm{Arn}} \mathcal{E}[\![Q]\!]$; and similarly $P \cong^{\mathrm{e}}_{\mathrm{ref}} Q$ iff $\mathcal{E}[\![P]\!] \cong^{\mathrm{e}}_{\mathrm{Arn}} \mathcal{E}[\![Q]\!]$.*

4 Bisimulation with Reference Names

4.1 Two Labelled Bisimilarities

In this section we present proof techniques for barbed equivalence based on the labelled transition semantics of $A\pi$. For this we introduce two labelled bisimilarities.

The first form of bisimulation, *reference bisimilarity*, only relates complete processes; processes that are not complete have to be made so. Intuitively, in this bisimilarity processes are made complete by requiring a closure of the relation with respect to the (well-typed) addition of output messages at reference names (the "closure under allocation" below). Moreover, when an observer consumes an output at a reference name, say $\overline{\ell}\langle n\rangle$, then, following the discipline on reference names, he/she has to immediately provide another such output message, say $\overline{\ell}\langle m\rangle$. This is formalised using transition notations such as $P \xrightarrow{\overline{\ell}\langle n\rangle[m]} P'$, which makes a swap on ℓ (reading its original content n and replacing it with m). As a consequence of the appearance of such swap transitions, ordinary outputs at reference names are not observed in the bisimulation. Similarly for inputs at reference names: an input $P \xrightarrow{\ell\langle m\rangle} P'$ from a complete process P is not observed, since it is supposed to interact with unique output at ℓ contained in P (which exists as P is complete). Finally, an observer should respect the completeness condition by the processes and should not communicate a fresh reference name – to communicate such a reference, say ℓ, an allocation for ℓ (an output message at ℓ) has first to be added.

A relation $\mathcal{R}$ is *closed under allocation* if $P\ \mathcal{R}\ Q$ implies $P \mid \overline{\ell}\langle n\rangle\ \mathcal{R}\ Q \mid \overline{\ell}\langle n\rangle$ for any $\overline{\ell}\langle n\rangle$ such that $P \mid \overline{\ell}\langle n\rangle$ and $Q \mid \overline{\ell}\langle n\rangle$ are well-typed. We write $P \xrightarrow{\overline{\ell}\langle n\rangle[m]} P'$ if $P \xrightarrow{\overline{\ell}\langle n\rangle} P''$ and $P' = \overline{\ell}\langle m\rangle \mid P''$, for some P''; similarly for $P \xrightarrow{(\boldsymbol{\nu} n)\overline{\ell}\langle n\rangle[m]} P'$. Then, as usual, $P \xRightarrow{\overline{\ell}\langle n\rangle[m]} P'$ holds if $P \Rightarrow P'' \xrightarrow{\overline{\ell}\langle n\rangle[m]} P''' \Rightarrow P'$ for some P'', P''', and similarly for $P \xRightarrow{(\boldsymbol{\nu} n)\overline{\ell}\langle n\rangle[m]} P'$.

We let α range over the actions μ plus the aforementioned "update actions" $\overline{\ell}\langle n\rangle[m]$ and $(\boldsymbol{\nu} n)\overline{\ell}\langle n\rangle[m]$.

Setting m to be the object of an update actions, we write $\Delta \vdash \alpha$ when: (i) if the object of α is a free reference name then it is in Δ, and (ii) α is not an input or an output at a reference name.

▶ **Definition 18** (Reference bisimilarity). *A symmetric relation $\mathcal{R}$ closed under allocation is a* reference bisimulation *if whenever $P\ \mathcal{R}\ Q$ with P, Q complete, $\Delta \vdash P, Q$ and $P \xrightarrow{\alpha} P'$ with $\Delta \vdash \alpha$, then*

1. *either there exists Q' such that $Q \stackrel{\hat{\alpha}}{\Rightarrow} Q'$ and $P'\ \mathcal{R}\ Q'$ for some Q'*
2. *or α is an input $a\langle m\rangle$ and $Q \mid \overline{a}\langle m\rangle \Rightarrow Q'$ with $P'\ \mathcal{R}\ Q'$ for some Q'.*

Reference bisimilarity, written $\approx$, is the largest reference bisimulation.

We now show that $\approx$ coincides with barbed equivalence. The structure of the proof is standard, however some care has to be taken to deal with closure under parallel composition.

▶ **Lemma 19.** *If $P \approx Q$, and $\emptyset \vdash R$, then $P \mid R \approx Q \mid R$.*

▶ **Proposition 20** (Substitutivity for active contexts). *If $P \approx Q$, then $E[P] \approx E[Q]$ for any active context E.*

▶ **Theorem 21** (Labelled characterisation). *$P \approx Q$ iff $P \cong^{\mathrm{e}}_{\mathrm{Arn}} Q$.*

In reference bisimilarity, the tested processes are complete: hence all their references must explicitly appear as allocated, and when a reference is accessed, an extension of the store is made so to remain with complete processes (and if such an extension introduces other new references, a further extension is needed). The goal of the bisimilarity $\approx_{\mathrm{ip}}$ below is to allow one to work on processes with open references, and make the extension of the store only when necessary. The definition of the bisimulation exploits an inductive predicate to accommodate finite extensions of the store, one step at a time. This predicate can be thought of as an inductive game, in which the "verifier" can choose rule `Base` and close the game, or choose rule `Ext` and a reference ℓ; in the latter case the "refuter" chooses the value stored in ℓ.

▶ **Definition 22** (Inductive predicate). *The predicate $\mathtt{ok}(\Delta, \mathcal{R}, P, Q, \mu)$ (where Δ is a set of names, $\mathcal{R}$ a process relation, P, Q processes, and μ an action) holds if it can be proved inductively from the following two rules:*

$$\textit{Base}\ \frac{\begin{cases} Q \mid \overline{n}\langle m\rangle \Rightarrow Q' & \text{for } \mu = n\langle m\rangle \\ Q \stackrel{\mu}{\Rightarrow} Q' & \text{otherwise} \end{cases} \qquad P'\ \mathcal{R}\ Q'}{\mathtt{ok}(\Delta, \mathcal{R}, P', Q, \mu)}$$

$$\textit{Ext}\ \frac{\ell \notin \Delta \qquad \forall\ m\ :\ \ \mathtt{ok}((\Delta, \ell), \mathcal{R}, P' \mid \overline{\ell}\langle m\rangle, Q \mid \overline{\ell}\langle m\rangle, \mu)}{\mathtt{ok}(\Delta, \mathcal{R}, P', Q, \mu)}$$

▶ **Definition 23** (Bisimilarity with inductive predicate, $\approx_{\mathrm{ip}}$). *A symmetric relation $\mathcal{R}$ is a* $\approx_{\mathrm{ip}}$-bisimulation *if whenever $P\ \mathcal{R}\ Q$ with $\Delta \vdash P, Q$, and $P \xrightarrow{\mu} P'$ with $\Delta' \vdash P'$, we can derive $\mathtt{ok}(\Delta \cup \Delta', \mathcal{R}, P', Q, \mu)$. We write $\approx_{\mathrm{ip}}$ for the largest $\approx_{\mathrm{ip}}$-bisimulation.*

The names in $\Delta \cup \Delta'$ are the reference names that appear in output subject position in P' or Q. Therefore, when using rule `Ext` of the inductive predicate, the condition $\ell \notin \Delta$ ensures us that the message at ℓ can be added without breaking typability.

The following up-to technique allows us to erase common messages on reference names along the bisimulation game.

For this, we use the notation M_s, where s is a finite list of pairs (ℓ,m), to describe parallel compositions of outputs on reference names (i.e., $M_s \stackrel{\text{def}}{=} \prod_{(\ell,m)\in s} \overline{\ell}\langle m\rangle$), and $\Delta_s \vdash M_s$ where Δ_s contains all first components of pairs of s. Intuitively, M_s represents a chunk of store.

▶ **Definition 24** ($\approx_{\text{ip}}$-bisimulation up to store). *An $\approx_{\text{ip}}$-bisimulation* up to store *is defined like $\approx_{\text{ip}}$-bisimulation (Definition 23), using a predicate* `ok'`$(\Delta\cup\Delta',\mathcal{R},P',Q,\mu)$. *This predicate is defined by a modified version of rule* `Ext` *where ok′ is used instead of ok, both in the premise and in the conclusion, and the following modified version of the* `Base` *rule:*

$$\textit{Base-Up}\ \frac{P' \equiv P'' \mid M_s \qquad \begin{cases} Q \mid \overline{n}\langle m\rangle \Rightarrow\equiv Q'' \mid M_s & \text{for } \mu = n\langle m\rangle \\ Q \stackrel{\mu}{\Rightarrow}\equiv Q'' \mid M_s & \text{otherwise} \end{cases} \qquad P''\ \mathcal{R}\ Q''}{\texttt{ok'}(\Delta,\mathcal{R},P',Q,\mu)}$$

Rule `Base-Up` makes it possible to erase common store components before checking that the processes are related by $\mathcal{R}$.

▶ **Proposition 25.** *If $\mathcal{R}$ is a $\approx_{\text{ip}}$-bisimulation up to store, then $\mathcal{R}\subseteq\approx_{\text{ip}}$.*

▶ **Proposition 26** (Soundness of $\approx_{\text{ip}}$). $\approx_{\text{ip}} \subseteq \approx$.

Intuitively, the inclusion holds because a $\approx_{\text{ip}}$-bisimulation is closed by parallel composition with M_s processes. We leave the opposite direction, completeness, as an open issue.

4.2 Examples

We now give examples of uses of the various forms of labelled bisimulation ($\approx_{\text{a}}$, $\approx$, $\approx_{\text{ip}}$, $\approx_{\text{ip}}$ up to store) for $A\pi$ to establish equivalences between processes with references. In some cases, we use the "up-to structural congruence" ($\equiv$) version of the bisimulations – a standard "up-to" technique. In the examples we consider barbed equivalence; the results can be lifted to barbed congruence using closure under substitutions.

The first example is about a form of commutativity for the write construct.

▶ **Example 27.** We wish to establish $!\ell \triangleleft a.\, \ell \triangleleft b \cong^{\text{e}}_{\text{ref}} !\ell \triangleleft b.\, \ell \triangleleft a$. For this, we prove the law $!\ell \triangleleft a.\, \ell \triangleleft b \cong^{\text{e}}_{\text{ref}} !\ell \triangleleft a \mid !\ell \triangleleft b$, which will be enough to conclude, by commutativity of parallel composition. The two given processes are mapped into $A\pi$ as

$$P_1 \stackrel{\text{def}}{=} !\ell(\_).\,(\overline{\ell}\langle a\rangle \mid \ell(\_).\,\overline{\ell}\langle b\rangle) \quad \text{and} \quad P_2 \stackrel{\text{def}}{=} (!\ell(\_).\,\overline{\ell}\langle a\rangle) \mid (!\ell(\_).\,\overline{\ell}\langle b\rangle).$$

We can derive $P_1 \approx_{\text{a}} P_2$, using the singleton relation $\mathcal{R} \stackrel{def}{=} \{(P_1,P_2)\}$, and showing that $\mathcal{R}$ is an asynchronous bisimilarity up-to context and structural congruence [18] (this known 'up-to' technique allows one to remove additional processes created from the replications after a transition). We can then conclude by Lemma 14.

We now consider Examples 3 and 4 from Section 2.

Proof of Example 3. Let R_1, R_2 be the encodings of P_1, P_2 in the example:

$$R_1 \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell)\big(\overline{\ell}\langle a\rangle \mid \mathcal{E}[\![P]\!] \mid !\ell(\_).\,\overline{\ell}\langle a\rangle \mid !\ell(\_).\,\overline{\ell}\langle b\rangle\big)$$
$$R_2 \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell)\big(\overline{\ell}\langle b\rangle \mid \mathcal{E}[\![P]\!] \mid !\ell(\_).\,\overline{\ell}\langle a\rangle \mid !\ell(\_).\,\overline{\ell}\langle b\rangle\big)$$

We then have $R_1 \Longrightarrow\equiv R_2$ and $R_2 \Longrightarrow\equiv R_1$, which implies $R_1 \approx_{\text{a}} R_2$ (where $\approx_{\text{a}}$ is asynchronous bisimilarity), as $\{(R_1,R_2)\}\cup\mathcal{I}$, where $\mathcal{I} = \{(P,P)\}$ is the identity relation, is an asynchronous bisimulation up to $\equiv$. We can then conclude by Theorems 8 and 17. ◀

Proof of Example 4. Let R_1, R_2 be the encodings of P_1, P_2 in the example:

$$R_1 \stackrel{\text{def}}{=} \ell(\_).(\overline{\ell}\langle b\rangle \mid \mathcal{E}[\![P]\!]) \mid !\ell(\_).\overline{\ell}\langle b\rangle \mid !\ell(x).(\overline{\ell}\langle x\rangle \mid \ell(\_).\overline{\ell}\langle x\rangle)$$
$$R_2 \stackrel{\text{def}}{=} \mathcal{E}[\![P]\!] \mid !\ell(\_).\overline{\ell}\langle b\rangle \mid !\ell(x).(\overline{\ell}\langle x\rangle \mid \ell(\_).\overline{\ell}\langle x\rangle)$$

Then for all m, processes $\overline{\ell}\langle m\rangle \mid R_1$ and $\overline{\ell}\langle m\rangle \mid R_2$ are complete. We define

$$\mathcal{R} \stackrel{def}{=} \{\big(R_1 \mid \overline{\ell}\langle m\rangle \mid B_X,\, R_2 \mid \overline{\ell}\langle m\rangle \mid B_X\big)\}\ ,$$

where $X \stackrel{\text{def}}{=} \{x_1, \ldots, x_n\}$ is a possibly empty finite set of names, and

$$B_X \stackrel{\text{def}}{=} \ell(\_).\overline{\ell}\langle x_1\rangle \mid \ldots \mid \ell(\_).\overline{\ell}\langle x_n\rangle$$

Then $\mathcal{R} \cup \mathcal{I}$ is a $\approx_{\text{ip}}$-bisimulation.

Reusing the same notations, $\mathcal{R}' \stackrel{def}{=} \{\big(R_1 \mid B_X,\, R_2 \mid B_X\big)\}$ is an $\approx_{\text{ip}}$-bisimulation up to store: this up-to technique allows us to remove the $\overline{\ell}\langle m\rangle$ particles. ◀

The following example shows some benefits of using $\approx_{\text{ip}}$ and $\approx_{\text{ip}}$ up to store in the proof of a property that generalises (the $A\pi$ version of) law (1), which involves a "useless read".

▶ **Example 28.** Consider $\emptyset \vdash P_0\ \mathcal{R}\ Q_0$, where $\mathcal{R}$ is an asynchronous bisimulation, $\texttt{ObType}(\ell) \in \texttt{RefTypes}$, and x is a fresh name. Then $\emptyset \vdash \ell(x).(P_0 \mid \overline{\ell}\langle x\rangle) \approx Q_0$.

In general, $\ell(x).(P_0 \mid \overline{\ell}\langle x\rangle)$ and Q_0 are not related by $\approx_{\text{a}}$ (take $P_0 = Q_0 = \overline{a}\langle n\rangle$), thus the inclusion in Lemma 14 is strict.

To prove $\ell(x).(P_0 \mid \overline{\ell}\langle x\rangle) \approx Q_0$ using a $\approx$-bisimulation, we need a relation such as

$$\begin{aligned}\mathcal{R}_1 \stackrel{\text{def}}{=}\ & \{(\ell(x).(P_0 \mid \overline{\ell}\langle x\rangle), Q_0)\}\\ & \cup \{(\ell(x).(P_0 \mid \overline{\ell}\langle x\rangle) \mid \overline{\ell}\langle\ell'\rangle \mid \overline{\ell'}\langle m\rangle, Q_0 \mid \overline{\ell}\langle\ell'\rangle \mid \overline{\ell'}\langle m\rangle) \;\big|\; \text{for any } m\}\\ & \cup \{(\ell(x).(P_0 \mid \overline{\ell}\langle x\rangle) \mid \overline{\ell}\langle\ell'\rangle \mid \overline{\ell'}\langle m\rangle \mid M_s, Q_0 \mid \overline{\ell}\langle\ell'\rangle \mid \overline{\ell'}\langle m\rangle \mid M_s) \;\big|\; \text{for any } m, M_s\}\\ & \cup \{P \mid \overline{\ell}\langle\ell'\rangle \mid \overline{\ell'}\langle m\rangle \mid M_s, Q \mid \overline{\ell}\langle\ell'\rangle \mid \overline{\ell'}\langle m\rangle \mid M_s) \;\big|\; \text{for any } m, M_s, \text{with } P\ \mathcal{R}\ Q\}\end{aligned}$$

and prove that $\mathcal{R}_1 \cup \mathcal{R}_1^{-1}$ (where $\mathcal{R}_1^{-1}$ is the inverse of $\mathcal{R}_1$) is a $\approx$-bisimulation.

We can simplify the proof and avoid the several quantifications in $\mathcal{R}_1$ (in particular on M_s, whose size is arbitrary), and prove that $\mathcal{R}_2$ is an $\approx_{\text{ip}}$-bisimulation, for

$$\begin{aligned}\mathcal{R}_2 \stackrel{\text{def}}{=}\ & \mathcal{R} \cup \{(P \mid \overline{\ell}\langle m\rangle, Q \mid \overline{\ell}\langle m\rangle),\ \text{for any } m, \text{with } P\ \mathcal{R}\ Q\}\\ & \cup \{(\ell(x).(P_0 \mid \overline{\ell}\langle x\rangle), Q_0),\ (Q_0, \ell(x).(P_0 \mid \overline{\ell}\langle x\rangle))\}.\end{aligned}$$

The last component of $\mathcal{R}_2$ is dealt with using rule `Ext` of the inductive predicate (Definition 22), and this brings in the second component (the closure of $\mathcal{R}$ under messages on ℓ).

We can simplify the proof further, by removing such second component, and show that $\mathcal{R}_3$ is an $\approx_{\text{ip}}$-bisimulation up to store, for

$$\mathcal{R}_3 \stackrel{\text{def}}{=} \mathcal{R} \cup \{(\ell(x).(P_0 \mid \overline{\ell}\langle x\rangle), Q_0),\ (Q_0, \ell(x).(P_0 \mid \overline{\ell}\langle x\rangle))\}.$$

5 Future work

In languages with store, which are usually sequential languages, bisimulation is commonly defined on *configurations*. In $\pi^{\mathtt{ref}}$, a configuration would be written $(\boldsymbol{\nu}\widetilde{n})\langle P, s\rangle$, where s is an explicit store and $\widetilde{n}$ is a set of private names shared between process P and store s. We could in principle read back $\approx$ onto $\pi^{\mathtt{ref}}$, and define a behavioural equivalence between $\pi^{\mathtt{ref}}$ configurations. The LTS on configurations would then have specific actions to describe how an observer may act on the visible part of the store. The labelled transition semantics for $\pi^{\mathtt{ref}}$ and $\pi^{\mathtt{ref}}$ configurations would however be more complex than those for $A\pi$; for instance the forms of actions, expressing external observations, would be much broader.

The swap operation arises naturally in the encoding into $A\pi$. We do not know if and how swap increases the discriminating power of external observers. We believe that, without swap, the two processes in Example 5 could not be distinguished. This point deserves further investigation, which we leave for future work. Similarly we leave for future work proving or disproving the completeness of the bisimilarity with an inductive predicate (Definition 23).

It would be interesting to see if the labelled bisimilarities we have considered, whose bisimulation clauses are different from those of ordinary bisimilarity, can be recovered in an abstract setting, e.g., using coalgebras [12, 2, 21]. This would be particularly interesting for $\approx_{\mathrm{ip}}$-bisimulation, whose definition involves a mixture of induction and coinduction.

Equivalences for higher-order languages with state are known to be hard to establish. Various approaches exist, from Kripke logical relations to trace semantics and game semantics [10, 11, 17, 4]. It would be interesting to compare the proof techniques offered by these approaches with those shown in this paper, and developments of them. More generally, more experimentation is needed to test the bisimilarities proposed in this paper and the associated proof techniques, on examples from high-level languages that include higher-order features, mutable state, and concurrency.

References

1 R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous pi-calculus. *Theor. Comput. Sci.*, 195(2):291–324, 1998.

2 F. Bonchi, D. Petrişan, D. Pous, and J. Rot. A general account of coinduction up-to. *Acta Informatica*, pages 1–64, 2016.

3 S. D. Brookes. The Essence of Parallel Algol. *Inf. Comput.*, 179(1):118–149, 2002.

4 S. Castellan, P. Clairambault, J. Hayman, and G. Winskel. Non-angelic concurrent game semantics. In *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018*, pages 3–19, 2018.

5 M. Coppo, M. Dezani-Ciancaglini, N. Yoshida, and L. Padovani. Global progress for dynamically interleaved multiparty sessions. *Math. Struct. Comput. Sci.*, 26(2):238–302, 2016.

6 M. P. Fiore, E. Moggi, and D. Sangiorgi. A fully abstract model for the π-calculus. *Inf. Comput.*, 179(1):76–117, 2002.

7 M. Hennessy. *A distributed Pi-calculus*. Cambridge University Press, 2007.

8 M. P. Herlihy. Impossibility and universality results for wait-free synchronization. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, PODC '88, pages 276–290, 1988.

9 D. Hirschkoff, E. Prebet, and D. Sangiorgi. Online appendix to this paper. available from `https://hal.archives-ouvertes.fr/hal-02895654`, 2020.

10 C.-K. Hur, D. Dreyer, G. Neis, and V. Vafeiadis. The marriage of bisimulations and kripke logical relations. In *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL*, pages 59–72, 2012.

11 G. Jaber and N. Tzevelekos. Trace semantics for polymorphic references. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16*, pages 585–594, 2016.

12 B. Jacobs. Introduction to coalgebra. towards mathematics of states and observations. Draft, 2014.

13 N. Kobayashi. A partially deadlock-free typed process calculus. *Transactions on Programming Languages and Systems*, 20(2):436–482, 1998. A preliminary version in *12th Lics Conf.* IEEE Computer Society Press 128–139, 1997.

14 M. Merro, J. Kleist, and U. Nestmann. Mobile objects as mobile processes. *Information and Computation*, 177(2):195–241, 2002.

15 R. Milner. *Communication and concurrency.* PHI Series in computer science. Prentice Hall, 1989.

16 R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (Parts I and II). *Inf. Comput.*, 100:1–77, 1992.

17 A. S. Murawski and N. Tzevelekos. Full abstraction for reduced ML. *Ann. Pure Appl. Logic*, 164(11):1118–1143, 2013.

18 D. Pous and D. Sangiorgi. *Advanced Topics in Bisimulation and Coinduction (D. Sangiorgi and J. Rutten editors)*, chapter Enhancements of the coinductive proof method. Cambridge University Press, 2011.

19 J. C. Reynolds. The essence of ALGOL. In *Algorithmic Languages*, pages 345–372. North-Holland, 1981.

20 C. Röckl and D. Sangiorgi. A pi-calculus process semantics of concurrent idealised ALGOL. In *Foundations of Software Science and Computation Structure, Second International Conference, FoSSaCS'99*, volume 1578 of *Lecture Notes in Computer Science*, pages 306–321. Springer, 1999.

21 J. Rot, F. Bonchi, M. M. Bonsangue, D. Pous, J. Rutten, and A. Silva. Enhanced coalgebraic bisimulation. *Math. Struct. Comput. Sci.*, 27(7):1236–1264, 2017.

22 D. Sangiorgi. The name discipline of uniform receptiveness. *Theor. Comput. Sci.*, 221(1-2):457–493, 1999.

23 D. Sangiorgi. Typed pi-calculus at work: A Correctness Proof of Jones's Parallelisation Transformation on Concurrent Objects. *TAPOS*, 5(1):25–33, 1999.

24 D. Sangiorgi and D. Walker. *The pi-calculus: a Theory of Mobile Processes*. Cambridge university press, 2003.

25 I. Stark. A fully abstract domain model for the pi-calculus. In *Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science*, pages 36–42. IEEE Computer Society, 1996.

A Additional Material for the Examples in Section 2.2

Proof of Example 5. To get a idea of how P_s and Q_s evolve, let us consider first $E \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell = z)[\,]$. Then $E[Q_s]$ can reduce to one of the following:

1. $(\boldsymbol{\nu}\ell = z)(\boldsymbol{\nu}t)\ell \triangleleft a.\,(\overline{t} \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c)))$
2. $(\boldsymbol{\nu}\ell = a)(\boldsymbol{\nu}t)(\overline{t} \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c))) \mid c^n \mid \overline{c}^n$
3. $(\boldsymbol{\nu}\ell = a)(\boldsymbol{\nu}t)(\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c)) \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c))) \mid c^n \mid \overline{c}^n$
4. $(\boldsymbol{\nu}\ell = b)(\boldsymbol{\nu}t)(\ell \triangleleft a.\,(\overline{t} \mid c) \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c))) \mid c^n \mid \overline{c}^{n+1}$.

Similarly, $E[P_s]$ can reduce to those four processes but with the role of a and b swapped. Notice that when $E[Q_s] \Longrightarrow Q'$, then there is a correspondence between the value stored in ℓ (i.e a or b) and the presence of more $\overline{c}$ processes than c processes (or the same number).

We now consider the following context:

$$E_0 \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell = z)([\,] \mid \ell \bowtie z(x).\,[x = b]s_0.\,s_1.\,(P_{11} \mid P_{12}) \mid \overline{s_0} \mid \overline{s_1})$$

$$P_{11} \stackrel{\text{def}}{=} \ell \triangleright (x).\,[x = z]s_{11} \mid \overline{s_{11}} \qquad P_{12} \stackrel{\text{def}}{=} c.\,\ell \triangleright (x).\,[x = z]s_{12} \mid \overline{s_{12}}$$

with s_0, s_{11}, s_{12} fresh names.

At first $\overline{s_0}$ and $\overline{s_1}$ are the only observables, meaning $E_0[P_s] \downarrow_{\overline{s_0}}$ and $E_0[P_s] \downarrow_{\overline{s_1}}$, but then $E_0[P_s] \longrightarrow\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow (\boldsymbol{\nu}\ell = z)((\boldsymbol{\nu}t)(\overline{t} \mid !t.\,\ell \triangleleft a.\,(\overline{c} \mid \ell \triangleleft b.\,(\overline{t} \mid c))) \mid s_1.\,(P_{11} \mid P_{12}) \mid \overline{s_1}) \stackrel{\text{def}}{=} P'$ where the three reductions have been derived using rules `R-Write`, `R-Swap`, and `R-Comm` respectively. Finally, we have $P' \not\downarrow_{\overline{s_0}}$, whereas $P' \downarrow_{\overline{s_1}}$.

Thus, to avoid the observable $\overline{s_0}$, process $E_0[Q_s]$ must reduce to a process with b stored in ℓ before doing the swap in E_0. This implies that the swap is executed in a state that corresponds to case 4 above. So for any Q' with $E[Q_s] \Longrightarrow Q'$ and $Q' \not\downarrow_{\overline{s_0}}$ and $Q' \Downarrow_{\overline{s_1}}$, such process Q' has one of the following forms:

1. $Q'_1 \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell = a)((\boldsymbol{\nu}t)(\overline{t} \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c)) \mid c^n \mid \overline{c}^n) \mid s_1.\,(P_{11} \mid P_{12}) \mid \overline{s_1})$
2. $Q'_2 \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell = a)((\boldsymbol{\nu}t)(\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c)) \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c)) \mid c^n \mid \overline{c}^n) \mid s_1.\,(P_{11} \mid P_{12}) \mid \overline{s_1})$
3. $Q'_3 \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell = b)((\boldsymbol{\nu}t)(\ell \triangleleft a.\,(\overline{t} \mid c) \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c)) \mid c^n \mid \overline{c}^{n+1}) \mid s_1.\,(P_{11} \mid P_{12}) \mid \overline{s_1})$
4. $Q'_4 \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell = z)((\boldsymbol{\nu}t)(\ell \triangleleft a.\,(\overline{t} \mid c) \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c)) \mid c^n \mid \overline{c}^{n+1}) \mid s_1.\,(P_{11} \mid P_{12}) \mid \overline{s_1})$

Then we use either P_{11} or P_{12} depending on the form of Q'. If Q' is of the first three forms, then we use P_{11}.

Indeed, $P' \longrightarrow\!\!\!\!\!\rightarrow (\boldsymbol{\nu}\ell = z)((\boldsymbol{\nu}t)(\overline{t} \mid !t.\,\ell \triangleleft a.\,(\overline{c} \mid \ell \triangleleft b.\,(\overline{t} \mid c))) \mid P_{12}) \stackrel{\text{def}}{=} P''$ using rules `R-Read` and `R-Comm` respectively. Notice that $P'' \not\downarrow_{\overline{s_{11}}}$. On the other hand, z does not appear anywhere else than in a matching in Q', thus there is no reduction $Q' \Longrightarrow Q''$ with $Q'' \not\downarrow_{\overline{s_{11}}}$ for any Q''.

In the other case, it holds that $Q'_4 \longrightarrow\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow (\boldsymbol{\nu}\ell = z)((\boldsymbol{\nu}t)(\ell \triangleleft a.\,(\overline{t} \mid c) \mid !t.\,\ell \triangleleft b.\,(\overline{c} \mid \ell \triangleleft a.\,(\overline{t} \mid c)) \mid c^n \mid \overline{c}^n) \mid P_{11}) \stackrel{\text{def}}{=} Q''$ using rules `R-Comm`, `R-Read`, and `R-Comm` respectively. Then we have $Q'' \not\downarrow_{\overline{s_{12}}}$. However, the only output $\overline{c}$ is behind a write $\ell \triangleleft a$ in P'. Thus, there is no $P' \Longrightarrow P''$ with $P'' \not\downarrow_{\overline{s_{12}}}$.

We can finally conclude $P_s \not\cong_{\text{ref}} Q_s$. ◀

Proof of Example 6. Recall the definitions of the two processes (we rename the processes that are given in the main text, to ease readability):

$$P \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell_1 = 0, \ell_2 = 0)(R \mid (\boldsymbol{\nu}t)(\overline{t} \mid !t.\,\ell_1 \triangleleft 1.\,\ell_1 \triangleleft 0.\,\ell_2 \triangleleft 1.\,\ell_2 \triangleleft 0.\,\overline{t}))$$

$$Q \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell_1 = 0, \ell_2 = 0)(R \mid (\boldsymbol{\nu}t)(\overline{t} \mid !t.\,\ell_1 \triangleleft 1.\,\ell_2 \triangleleft 1.\,\ell_1 \triangleleft 0.\,\ell_2 \triangleleft 0.\,\overline{t}))$$

To prove their equivalence, we introduce the following processes:

$$P' \stackrel{\text{def}}{=} !t.\,\ell_1(\_).\,(\overline{\ell_1}\langle 1\rangle \mid \ell_1(\_).\,(\overline{\ell_1}\langle 0\rangle \mid \ell_2(\_).\,(\overline{\ell_2}\langle 1\rangle \mid \ell_2(\_).\,(\overline{\ell_2}\langle 0\rangle \mid \overline{t}))))$$

$$Q' \stackrel{\text{def}}{=} !t.\,\ell_1(\_).\,(\overline{\ell_1}\langle 1\rangle \mid \ell_2(\_).\,(\overline{\ell_2}\langle 1\rangle \mid \ell_1(\_).\,(\overline{\ell_1}\langle 0\rangle \mid \ell_2(\_).\,(\overline{\ell_2}\langle 0\rangle \mid \overline{t}))))$$

$$
\begin{aligned}
P_1 = Q_1 &\stackrel{\text{def}}{=} \bar{t} \\
P_2 &\stackrel{\text{def}}{=} \ell_1(\_).(\overline{\ell_1}\langle 1\rangle \mid \ell_1(\_).(\overline{\ell_1}\langle 0\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 1\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 0\rangle \mid \bar{t})))) \\
Q_2 &\stackrel{\text{def}}{=} \ell_1(\_).(\overline{\ell_1}\langle 1\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 1\rangle \mid \ell_1(\_).(\overline{\ell_1}\langle 0\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 0\rangle \mid \bar{t})))) \\
P_3 &\stackrel{\text{def}}{=} \ell_1(\_).(\overline{\ell_1}\langle 0\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 1\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 0\rangle \mid \bar{t}))) \\
Q_3 &\stackrel{\text{def}}{=} \ell_2(\_).(\overline{\ell_2}\langle 1\rangle \mid \ell_1(\_).(\overline{\ell_1}\langle 0\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 0\rangle \mid \bar{t}))) \\
P_4 &\stackrel{\text{def}}{=} \ell_2(\_).(\overline{\ell_2}\langle 1\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 0\rangle \mid \bar{t})) \\
Q_4 &\stackrel{\text{def}}{=} \ell_1(\_).(\overline{\ell_1}\langle 0\rangle \mid \ell_2(\_).(\overline{\ell_2}\langle 0\rangle \mid \bar{t})) \\
P_5 = Q_5 &\stackrel{\text{def}}{=} \ell_2(\_).(\overline{\ell_2}\langle 0\rangle \mid \bar{t})
\end{aligned}
$$

P' and Q' are the encodings of the replicated part of P and Q. Then P_i and Q_i are the processes that can be reached from P' and Q'.

We now show that the relation $\mathcal{R} \cup \mathcal{R}^{-1}$ is an $\approx_{\text{ip}}$-bisimulation where we have:

$$
\begin{aligned}
\mathcal{R} \stackrel{\text{def}}{=} &\left\{ \begin{array}{c} (\overline{\ell_1}\langle n_1\rangle \mid (\boldsymbol{\nu} t)(P' \mid P_i),\ \overline{\ell_1}\langle n_1'\rangle \mid (\boldsymbol{\nu} t)(Q' \mid Q_j)) \\ \text{for any } n_1, n_1' \in \{0,1\}, i, j \end{array} \right\} \\
\cup &\left\{ \begin{array}{c} (\overline{\ell_2}\langle n_2\rangle \mid (\boldsymbol{\nu} t)(P' \mid P_i),\ \overline{\ell_2}\langle n_2'\rangle \mid (\boldsymbol{\nu} t)(Q' \mid Q_j)) \\ \text{for any } n_2, n_2' \in \{0,1\}, i, j \end{array} \right\} \\
\cup &\left\{ \begin{array}{c} (\overline{\ell_1}\langle n_1\rangle \mid \overline{\ell_2}\langle n_2\rangle \mid (\boldsymbol{\nu} t)(P' \mid P_i),\ \overline{\ell_1}\langle n_1'\rangle \mid \overline{\ell_2}\langle n_2'\rangle \mid (\boldsymbol{\nu} t)(Q' \mid Q_j)) \\ \text{for any } n_1, n_1', n_2, n_2' \in \{0,1\}, i, j \end{array} \right\}
\end{aligned}
$$

First, note that the only free names appearing in those processes are ℓ_1 and ℓ_2. Thus for any $P \mathcal{R} Q$, the only actions to consider are $\tau, \ell_i\langle n\rangle$ and $\overline{\ell_i}\langle n\rangle$, for $i = 1, 2$.

For any $P \mathcal{R} Q$, we have:

- If $P \xrightarrow{\tau} P_0$, then $P_0 \mathcal{R} Q$
- If $P \xrightarrow{\ell_i\langle n\rangle} P_0$, then $P_0 \mathcal{R} Q \mid \overline{\ell_i}\langle n\rangle$
- If $P \xrightarrow{\overline{\ell_i}\langle n\rangle} P_0$, then either $Q \xrightarrow{\overline{\ell_i}\langle n\rangle} Q_0$ and $P_0 \mathcal{R} Q_0$, or $Q \xrightarrow{\overline{\ell_i}\langle 1-n\rangle} Q_0$. In this case, we use rule `Ext` (from Definition 22) to add the other location if $\Delta \neq \ell_1, \ell_2$. Then after at most 5 internal transitions (by cycling around the P_i or Q_j), we obtain a process Q_0 that can make the required transition $Q_0 \xrightarrow{\overline{\ell_i}\langle n\rangle} Q_0'$ with $P_0 \mathcal{R} Q_0'$.

As $\mathcal{R} \cup \mathcal{R}^{-1}$ is an $\approx_{\text{ip}}$-bisimulation, we have $\mathcal{R} \subseteq \approx$. Moreover, $(\boldsymbol{\nu}\ell_1, \ell_2)(\mathcal{E}[\![R]\!] \mid [\,])$ is an active context, so this implies $\mathcal{E}[\![P]\!] \approx \mathcal{E}[\![Q]\!]$. By Theorems 21 and 17, we can conclude $P \cong^{\text{e}}_{\text{ref}} Q$.

To extend this result to barbed congruence, we notice that for all σ,

1. either $P\sigma = (\boldsymbol{\nu}\ell_1 = 0, \ell_2 = 0)(R\sigma \mid (\boldsymbol{\nu} t)(\bar{t} \mid !t.\,\ell_1 \triangleleft 1.\,\ell_1 \triangleleft 0.\,\ell_2 \triangleleft 1.\,\ell_2 \triangleleft 0.\,\bar{t})$
2. or $P\sigma = (\boldsymbol{\nu}\ell_1 = 0, \ell_2 = 0)(R\sigma \mid (\boldsymbol{\nu} t)(\bar{t} \mid !t.\,\ell_1 \triangleleft 0.\,\ell_1 \triangleleft 0.\,\ell_2 \triangleleft 0.\,\ell_2 \triangleleft 0.\,\bar{t})$
3. or $P\sigma = (\boldsymbol{\nu}\ell_1 = 1, \ell_2 = 1)(R\sigma \mid (\boldsymbol{\nu} t)(\bar{t} \mid !t.\,\ell_1 \triangleleft 1.\,\ell_1 \triangleleft 1.\,\ell_2 \triangleleft 1.\,\ell_2 \triangleleft 1.\,\bar{t})$

As $P \cong^{\text{e}}_{\text{ref}} Q$ holds for any R, it also holds for any $R\sigma$, which prove the first case. Moreover, the proof never uses the fact that 0 and 1 are distinct, so we can prove in the same way that cases 2 and 3 hold.

We conclude $P \cong_{\text{ref}} Q$. ◀

We now present an additional example, which corresponds to a generalisation of Example 6.

▶ **Example 29.** Here we remove the assumption that the two references can only hold values 0 and 1. This enables the context to store fresh names in references. If used with the original processes, these are distinguished by using those fresh values to block transition along the lines of Example 5. To make these processes equivalent again, we could add in parallel a buffer as in Example 4. However, by making these additions, we would also enable P_1 to desynchronise the content in ℓ_1 and ℓ_2 and have $(1,1)$. The solution is to prevent those buffers from writing at a different "time" than the "time" they have read. For this we introduce a more complex buffer B_i^j. Consider the following processes:

$$B_i^j \stackrel{\text{def}}{=} r(x^j).\,\mathbf{0} \mid !r(x^j).\,t_i.\,\ell^j \bowtie x^j(y^j).\,(\overline{r}\langle y^j\rangle \mid \overline{t_i})$$
$$S_i^j \stackrel{\text{def}}{=} !t_i.\,\ell^j \triangleright (x^j).\,(\overline{t_i} \mid (\boldsymbol{\nu} r)(\overline{r}\langle x^j\rangle \mid B_i^j))$$

$$P \stackrel{\text{def}}{=} (\boldsymbol{\nu} t_1,t_2,t_3,t_4)\Big(\overline{t_1} \mid !t_1.\,\ell^1 \triangleleft 1.\,\overline{t_2} \mid S_1^1 \mid S_1^2 \mid !t_2.\,\ell^1 \triangleleft 0.\,\overline{t_3} \mid S_2^1 \mid S_2^2 \mid !t_3.\,\ell^2 \triangleleft 1.\,\overline{t_4} \mid S_3^1 \mid S_3^2 \mid !t_4.\,\ell^2 \triangleleft 0.\,\overline{t_1} \mid S_4^1 \mid S_4^2\Big)$$
$$Q \stackrel{\text{def}}{=} (\boldsymbol{\nu} t_1,t_2,t_3,t_4)\Big(\overline{t_1} \mid !t_1.\,\ell^1 \triangleleft 1.\,\overline{t_2} \mid S_1^1 \mid S_1^2 \mid !t_2.\,\ell^2 \triangleleft 1.\,\overline{t_3} \mid S_2^1 \mid S_2^2 \mid !t_3.\,\ell^1 \triangleleft 0.\,\overline{t_4} \mid S_3^1 \mid S_3^2 \mid !t_4.\,\ell^2 \triangleleft 0.\,\overline{t_1} \mid S_4^1 \mid S_4^2\Big)$$

We have $P \cong_{\text{ref}} Q$. If we take $E \stackrel{\text{def}}{=} (\boldsymbol{\nu}\ell^1 = 0)(\boldsymbol{\nu}\ell_2 = 0)[\,]$, we have $E[Q] \longrightarrow\!\!\!\!\longrightarrow (\boldsymbol{\nu}\ell^1 = 1)(\boldsymbol{\nu}\ell^2 = 1)Q'$ for some Q'. However, there is no sequence of reductions such that $E[P] \Longrightarrow (\boldsymbol{\nu}\ell^1 = 1)(\boldsymbol{\nu}\ell^2 = 1)P'$ for any P'.

If we forget all S_i^j's, then these processes are similar to the 'loop' used in the previous example but split into multiple replications. Those S_i^j's help to equate the two processes even if the context can write any value in ℓ_1, ℓ_2.

Process S_i^j can only be activated when $\overline{t_i}$ is available. It then reads the content of ℓ_j to initialise a new buffer B_i^j.

Process B_i^j contains value x_i^j that is the object of $\overline{r}\langle x_i^j\rangle$. Process B_i^j can be stopped by making the communication with the first input on r, or can be used to swap its content with the content of ℓ^j. Note that this swap can only be done when $\overline{t_i}$ is available, so it cannot be used to desynchronise the content in ℓ_1, and ℓ_2.

B Operational Semantics of $A\pi$: Reduction and Labelled Transitions

Reduction

Structural congruence is defined as the smallest congruence that satisfies the following axioms:

$$P \mid \mathbf{0} \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad !P \equiv P$$
$$P \mid (\boldsymbol{\nu} n)Q \equiv (\boldsymbol{\nu} n)P \mid Q \text{ if } n \notin \text{fn}(P) \qquad (\boldsymbol{\nu} n)(\boldsymbol{\nu} m)P \equiv (\boldsymbol{\nu} m)(\boldsymbol{\nu} n)P \qquad (\boldsymbol{\nu} n)\mathbf{0} \equiv \mathbf{0}$$
$$[x = x]P \equiv P$$

Active contexts in $A\pi$ are defined by:

$$E \quad ::= \quad [\,] \mid E \mid P \mid (\boldsymbol{\nu} n)E \ .$$

Reduction is defined by the following rules:

$$\frac{P \equiv P' \qquad P' \longrightarrow Q' \qquad Q' \equiv Q}{P \longrightarrow Q} \qquad \frac{P \longrightarrow P'}{E[P] \longrightarrow E[Q]} \qquad \frac{}{n(x).\,P \mid \overline{n}\langle m\rangle \longrightarrow P\{m/x\}}$$

$$\texttt{Inp:}\ \frac{}{n(x).\,P \xrightarrow{n\langle m\rangle} P\{m/x\}} \qquad \texttt{Out:}\ \frac{}{\overline{n}\langle m\rangle \xrightarrow{\overline{n}\langle m\rangle} \mathbf{0}}$$

$$\texttt{Open:}\ \frac{P \xrightarrow{\overline{n}\langle m\rangle} P'}{(\boldsymbol{\nu} m)P \xrightarrow{(\boldsymbol{\nu} m)\overline{n}\langle m\rangle} P'}\ \text{if } m \neq n \qquad \texttt{Rep:}\ \frac{P \mid !P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P'}$$

$$\texttt{Res:}\ \frac{P \xrightarrow{\mu} P'}{(\boldsymbol{\nu} n)P \xrightarrow{\mu} (\boldsymbol{\nu} n)P'}\ \text{if } n \notin \mu \qquad \texttt{Par:}\ \frac{P \xrightarrow{\mu} P'}{P \mid Q \xrightarrow{\mu} P' \mid Q}\ \text{if } \mathrm{bn}(\mu) \cap \mathrm{fn}(Q) = \emptyset$$

$$\texttt{Comm:}\ \frac{P \xrightarrow{n\langle m\rangle} P' \qquad Q \xrightarrow{\overline{n}\langle m\rangle} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\texttt{Close:}\ \frac{P \xrightarrow{n\langle m\rangle} P' \qquad Q \xrightarrow{(\boldsymbol{\nu} m)\overline{n}\langle m\rangle} Q'}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu} m)(P' \mid Q')}\ \text{if } m \notin \mathrm{fn}(P) \qquad \texttt{Match:}\ \frac{P \xrightarrow{\mu} P'}{[n = n]P \xrightarrow{\mu} P'}$$

Figure 3 Labelled Transition Semantics for $A\pi$.

Labelled Transition Semantics

Actions of the LTS are defined as follows:

$$\mu \quad ::= \quad n(m) \mid \overline{n}\langle m\rangle \mid (\boldsymbol{\nu} m)\overline{n}\langle m\rangle \mid \tau \ .$$

Transitions are defined in Figure 3. The symmetric versions of rules PAR, COM and CLOSE are omitted. Weak transitions are defined by $\Rightarrow \stackrel{\text{def}}{=} \xrightarrow{\tau}{}^{*}$, $\stackrel{\mu}{\Rightarrow} \stackrel{\text{def}}{=} \Rightarrow\xrightarrow{\mu}\Rightarrow$, and $\stackrel{\hat{\mu}}{\Rightarrow} \stackrel{\text{def}}{=} \stackrel{\mu}{\Rightarrow}$ if $\mu \neq \tau$ and $\Rightarrow$ otherwise.

Canonical Solutions to Recursive Equations and Completeness of Equational Axiomatisations

Xinxin Liu
State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, China
University of Chinese Academy of Sciences, China
xinxin@ios.ac.cn

TingTing Yu
Beijing Sunwise Information Technology Ltd, China
Beijing Institute of Control Engneering, China
yutingting@sunwiseinfo.com

Abstract

In this paper we prove completeness of four axiomatisations for finite-state behaviours with respect to behavioural equivalences at various τ-abstract levels: branching congruence, delay congruence, η-congruence, and weak congruence. Instead of merging guarded recursive equations, which was the approach originally used by Robin Milner and has since become the standard strategy for proving completeness results of this kind, in this work we take a new approach by solving guarded recursive equations with canonical solutions which are those with the fewest reachable states. The new strategy allows uniform treatment of the axiomatisations with respect to different behavioural equivalences.

2012 ACM Subject Classification Theory of computation → Process calculi

Keywords and phrases Bisimulation, Congruence, Axiomatisation, Soundness and Completeness

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.35

Funding *Xinxin Liu*: The work has been partially supported by the CAS-INRIA major project No. GJHZ1844, and by NSFC under grants No. 61836006 and No. 61672540.

Acknowledgements We thank the reviewers' suggestions which greatly improved the presentation.

1 Introduction

The notion of bisimulation which originated from the early ideas of Park and van Benthem and coined by Milner is the foundation of many popular equivalence and congruence relations in concurrency theory. Weak bisimilarity introduced by Milner [6] (originally called observational equivalence) and branching bisimilarity introduced by van Glabbeek and Weijland [9] are two widely studied equivalence relations for processes, with the former identifying more processes than the latter because of the difference in treating internal actions (τ-moves) to achieve observational abstractness. Delay bisimilarity and η-bisimilarity [9] are two other interesting equivalence relations with abstractness in between weak and branching bisimilarities. Although these relations are not congruences on process constructs, all can be made into congruences, called weak congruence and branching congruence etc., which are very close to the respective bisimilarities. For finite processes where every process will eventually become inactive, complete inference systems for all four congruences mentioned above can be found in the literature. For example, a complete axiomatisation of finite processes with respect to weak congruence was devised by Hennessy and Milner [3], and complete axiomatisations of finite processes with respect to branching, delay, and η-congruences were devised by van Glabbeek and Weijland [9]. For finite-state processes where recursion is allowed in the process constructs to introduce infinite behaviours, complete inference systems for weak congruence and branching congruence were devised by Milner [7] and van Glabbeek [8] respectively,

while such inference systems for the other two congruences are not found in the literature. In this paper we present a hierarchy of complete inference systems for finite-state processes with respect to the four congruences. For proving completeness, we use a new approach by constructing canonical solutions to guarded recursive equations which is different from the common strategy of merging guarded recursive equations, a strategy originally proposed by Milner [5] and later followed in many other completeness proofs [7][10][8][4]. Our new strategy allows more uniform treatment of the axiomatisations with respect to different behavioural equivalences. In the following we roughly explain the ideas of the old and new approaches.

Both the old and new approaches rely on the following two fundamental facts:

Fact 1. Each expression provably solves a set of guarded recursive equations.

Fact 2. A set of guarded recursive equations has unique solution in the following sense: if two expressions both provably solve the same set of guarded recursive equations, then the two expressions are provably equal.

In [5], after setting up the foundation by proving the two facts above, Milner then devised the following strategy to establish the provable equality of two semantically equivalent expressions E_1 and E_2:

Step 1. By Fact 1. above, let S_1, S_2 be two sets of guarded recursive equations such that E_1 and E_2 provably solve S_1 and S_2 respectively.

Step 2. Then, with the help of the semantic equality of E_1 and E_2, by merging S_1 and S_2 in a systematic way to form a single set of guarded recursive equations S which is provably solved by both E_1 and E_2.

Step 3. Then by Fact 2. above, it follows that E_1 and E_2 are provably equal.

For Step 2. Milner provided a procedure which guarantees that the wanted guarded recursive equation set S can be constructed. To see a tiny but nevertheless illustrative example, consider the expressions $\mu X.(a.a.X)$ and $\mu Y.(a.a.a.Y)$, they both perform the visible action a forever. If we write $\vdash E = F$ to mean that the expressions E and F are provably equal, then following Milner's strategy, first note that by unfolding recursion we obtain

$$\vdash \mu X.(a.a.X) = a.a.\mu X.(a.a.X), \quad \vdash \mu Y.(a.a.a.Y) = a.a.a.\mu Y.(a.a.a.Y).$$

That is, $\mu X.(a.a.X)$ and $\mu Y.(a.a.a.Y)$ provably solve $\{X = a.a.X\}$ and $\{Y = a.a.a.Y\}$ respectively. Now by equational reasoning, if $\vdash E = a.a.E$, then $\vdash a.a.E = a.a.a.a.E$, and $\vdash a.a.a.a.E = a.a.a.a.a.a.E$, thus $\vdash E = a.a.a.a.a.a.E$. That is to say, a solution to $X = a.a.X$ also provably solves $Z = a.a.a.a.a.a.Z$. And for the same reason a solution to $Y = a.a.a.Y$ also provably solves $Z = a.a.a.a.a.a.Z$. Thus both $\mu X.(a.a.X)$ and $\mu Y.(a.a.a.Y)$ provably solve $\{Z = a.a.a.a.a.a.Z\}$, hence by Fact 2. $\vdash \mu X.(a.a.X) = \mu Y.(a.a.a.Y)$. Note that here we have simplified Milner's procedure by allowing sets of guarded equations which may not be in the *standard form* required in his procedure. That is sufficient to make our points. (The sets of guarded equations in standard form which provably solved by $\mu X.(a.a.X)$ and $\mu Y.(a.a.a.Y)$ are $\{X_0 = a.X_1, X_1 = a.X_0\}$ and $\{Y_0 = a.Y_1, Y_1 = a.Y_2, Y_2 = a.Y_0\}$ respectively, which can be merged to obtain the following set of standard equations, provably solved by both $\mu X.(a.a.X)$ and $\mu Y.(a.a.a.Y)$ in Z_{00}: $\{Z_{00} = a.Z_{11}, Z_{11} = a.Z_{02}, Z_{02} = a.Z_{10}, Z_{10} = a.Z_{01}, Z_{01} = a.Z_{12}, Z_{12} = a.Z_{00}\}$.)

Now by examining Milner's approach, we can find an interesting alternative which has never been explored. The idea, which is very simple, is as follows. After Step 1., instead of merging S_1 and S_2 to obtain S in Step 2. (which is the most involving and complex step in

this approach), is it possible to find *an expression* E which provably solves both S_1 and S_2? If the answer is yes, then by Fact 2. E is provably equal to both E_1 and E_2, thus by transitivity E_1 is provably equal to E_2. Let us apply this alternative approach to the above example, and consider the expression $\mu Z.(a.Z)$. By unfolding we obtain $\vdash \mu Z.(a.Z) = a.\mu Z.(a.Z)$, and then by equational reasoning we obtain $\vdash a.\mu Z.(a.Z) = a.a.\mu Z.(a.Z)$ and $\vdash a.a.\mu Z.(a.Z) = a.a.a.\mu Z.(a.Z)$, thus $\vdash \mu Z.(a.Z) = a.\mu Z.(a.Z) = a.a.\mu Z.(a.Z) = a.a.a.\mu Z.(a.Z)$. This shows that, $\mu Z.(a.Z)$ provably solves both $\{X = a.a.X\}$ (which is also probably solved by $\mu X.(a.a.X)$) and $\{Y = a.a.a.Y\}$ (which is also provably solved by $\mu Y.(a.a.a.Y)$). Then we can use Fact 2. to obtain $\vdash \mu Z.(a.Z) = \mu X.(a.a.X)$ and $\vdash \mu Z.(a.Z) = \mu Y.(a.a.a.Y)$, hence $\vdash \mu X.(a.a.X) = \mu Y.(a.a.a.Y)$.

The work of this paper is to show that the alternative approach works in general case. We show that the common provable solution E to S_1 and S_2 can always be constructed out of canonical solutions to some set of guarded recursive equations, where canonical solutions are those with the smallest number of reachable states amongst the solutions.

The paper is organized as follows. In the next section we settle the preliminaries. In section 3 we present a hierarchy of axiomatisations for branching congruence, delay congruence, η-congruence, and weak congruence, and state some basic properties. In section 4 we carry out in detail the construction of canonical solutions to recursive equations to prove the completeness result for branching congruence. In section 5 we prove some saturation results, and with which to obtain the completeness with respect to other three congruences. Then, after discussing related work in section 6, we conclude in section 7.

2 Expressions, Equivalences, and Congruences

Let $\mathcal{V}$ be an infinite set of *variables*, $\mathcal{A}$ be an infinite set of *visible actions*, τ be the *invisible action* or *silent move* ($\tau \notin \mathcal{A}$). We write $\mathcal{A}_\tau$ for $\mathcal{A} \cup \{\tau\}$. The set $\mathcal{E}$ of *regular process expressions* is given by the following BNF grammar:

$$E \quad ::= \quad \mathbf{0} \;\Big|\; X \;\Big|\; a.E \;\Big|\; E + E \;\Big|\; \mu X.E$$

where $a \in \mathcal{A}_\tau$, $X \in \mathcal{V}$. Here $\mathbf{0}$ is the null expression which is not capable of any action; $a.E$ is a prefix expression which first performs the action a and then proceeds as E; $E + F$ is a summation which will behave as either E or F; $\mu X.E$ stands for recursion, with μ binding the variable X in E. We assume the usual notion of free and bound occurrences of variables with respect to the variable binder μ, and write $E\{F/X\}$ for the (capture free) substitution of F for (free occurrences of) X in E. More generally, for a finite set of variables $\{X_1, \ldots, X_n\}$, we write $E\{E_1, \ldots, E_n/X_1, \ldots, X_n\}$ or $E\{E_i/X_i \mid i = 1, \ldots, n\}$ for the expression obtained by simultaneous (capture free) substitution of F_1 for X_1, …, F_n for X_n in E. We write $E \equiv F$ when E, F are syntactically identical.

For an expression $E \in \mathcal{E}$, the set of free variables of E, written $fv(E)$, is defined on the structure of E such that

$$fv(\mathbf{0}) = \emptyset, \quad fv(X) = \{X\}, \quad fv(a.E) = fv(E),$$
$$fv(E + F) = fv(E) \cup fv(F), \quad fv(\mu X.E) = fv(E) - \{X\}.$$

Then it is easy to prove that $fv(E)$ is a finite set for all expression $E \in \mathcal{E}$.

The operational semantics of expressions is given by a transition relation and two binary relations between expressions and variables defined as follows.

▶ **Definition 1.** *The transition relation* $\longrightarrow \subseteq \mathcal{E} \times \mathcal{A}_\tau \times \mathcal{E}$ *is the smallest relation such that (as usual we write* $E \xrightarrow{a} E'$ *for* $(E, a, E') \in \longrightarrow$*):*

1. $a.E \xrightarrow{a} E$;
2. *If* $E_1 \xrightarrow{a} E'$ *then* $E_1 + E_2 \xrightarrow{a} E'$;
3. *If* $E_2 \xrightarrow{a} E'$ *then* $E_1 + E_2 \xrightarrow{a} E'$;
4. *If* $E\{\mu X.E/X\} \xrightarrow{a} E'$ *then* $\mu X.E \xrightarrow{a} E'$.

Define $\triangleright \subseteq \mathcal{E} \times \mathcal{V}$ *as the smallest relation such that*

5. $X \triangleright X$;
6. *If* $E_1 \triangleright X$ *then* $E_1 + E_2 \triangleright X$;
7. *If* $E_2 \triangleright X$ *then* $E_1 + E_2 \triangleright X$;
8. *If* $E\{\mu Y.E/Y\} \triangleright X$ *then* $\mu Y.E \triangleright X$.

Intuitively, $E \triangleright X$ means that X has an occurrence in E which is not proceeded by any action, not even a τ.

We follow the CCS (Milner's Calculus of Communicating Systems [6]) tradition to write $\Longrightarrow$ for $(\xrightarrow{\tau})^*$, i.e. the reflexive and transitive closure of $\xrightarrow{\tau}$. We also freely write $\Longrightarrow\xrightarrow{a}$ and $\Longrightarrow\xrightarrow{a}\Longrightarrow$ etc. for various composition of transition relations. We write $E \mathrel{\triangleright\!\!\!\!\!\cdot} X$ if $E \Longrightarrow E'$ for some E' with $E' \triangleright X$.

▶ **Definition 2.** *A free occurrence of a variable* X *in an expression* $E \in \mathcal{E}$ *is* guarded *if it occurs in a subexpression of the form* $a.F$ *where* $a \neq \tau$. X *is* (un)guarded in E *if (not) every free occurrence of* X *in* E *is guarded. An expression* $E \in \mathcal{E}$ *is* guarded *if for every subexpression* $\mu X.F$, X *is guarded in* F.

▶ **Lemma 3.** *Let* $E \in \mathcal{E}, X \in \mathcal{V}$. X *is unguarded in* E *iff* $E \mathrel{\triangleright\!\!\!\!\!\cdot} X$.

Proof. Straightforward. ◀

▶ **Theorem 4.** *For* $E \in \mathcal{E}$, *let* $\mathcal{E}_E$ *be the set of expressions which are* reachable *from* E, *i.e.* $\mathcal{E}_E$ *is the smallest subset of* $\mathcal{E}$ *such that* $E \in \mathcal{E}_E$ *and* $\mathcal{E}_E$ *is closed for transitions (if* $F \xrightarrow{a} G$ *with* $F \in \mathcal{E}_E$ *and* $a \in \mathcal{A}_\tau$ *then* $G \in \mathcal{E}_E$*). Then* $\mathcal{E}_E$ *is a finite set.*

Proof. It was proved in [8] (Proposition 1). ◀

▶ **Definition 5.** *Let* $R \subseteq \mathcal{E} \times \mathcal{E}$ *be a symmetric binary relation between expressions. If for all* $(E, F) \in R$ *the following hold:*

1. *whenever* $E \xrightarrow{a} E'$, *then*
   a. *either* $a = \tau$ *and* $(E', F) \in R$,
   b. *or there exist* F_1, F_2, F' *such that* $F \Longrightarrow F_1, F_1 \xrightarrow{a} F_2, F_2 \Longrightarrow F'$, *and* $(E, F_1), (E', F_2), (E', F') \in R$;
2. *whenever* $E \triangleright X$, *then there exists* F' *such that* $F \Longrightarrow F'$, $F' \triangleright X$, *and* $(E, F') \in R$;

then R *is called a branching bisimulation.*

If for all $(E, F) \in R$ *the above hold except that without requiring* $(E', F_2) \in R$ *in 1. then* R *is called an* η*-bisimulation.*

If for all $(E, F) \in R$ *the above hold except that without requiring* $(E, F_1) \in R$ *in 1. and* $(E, F') \in R$ *in 2., then* R *is called a delay bisimulation.*

If for all $(E, F) \in R$ *the above hold except that without requiring* $(E, F_1), (E', F_2) \in R$ *in 1. and* $(E, F') \in R$ *in 2., then* R *is called a weak bisimulation.*

Define branching bisimilarity, delay bisimilarity, η*-bisimilarity, weak bisimilarity, written* $\approx_b, \approx_d, \approx_\eta, \approx_w$ *respectively as follows*

$$\approx_b = \bigcup\{R \mid R \text{ is a branching bisimulation}\}, \quad \approx_d = \bigcup\{R \mid R \text{ is a delay bisimulation}\},$$
$$\approx_\eta = \bigcup\{R \mid R \text{ is an } \eta\text{-bisimulation}\}, \quad \approx_w = \bigcup\{R \mid R \text{ is a weak bisimulation}\}.$$

▶ Remark. Normally the definition of branching bisimulation does not require the existence of F' in 1., that is because with the existence of F_2 the requirement of F' is trivially satisfied by taking F_2 as F'. For this reason the variations of definition result in the same relation.

With the above definition, it is well-known that $\approx_b, \approx_d, \approx_\eta, \approx_w$ are all equivalence relations. Moreover, it is standard to prove that each of the equivalences is the largest one amongst the corresponding bisimulation relations. Also, about the distinguishing power of these equivalences we have $\approx_b \subseteq \approx_d \subseteq \approx_w$, and $\approx_b \subseteq \approx_\eta \subseteq \approx_w$.

None of the four equivalences is a congruence on $\mathcal{E}$. As a classical counterexample, note that $a.0 \approx_w \tau.a.0$ while $a.0 + b.0 \not\approx_w \tau.a.0 + b.0$, where a, b are different non-τ actions. A standard way to get around this problem, as noted in [9], is to introduced a rootedness condition on top of these equivalences to obtain congruence relations for expressions.

▶ **Definition 6.** *Two expressions E and F are rooted branching bisimilar, notation $E =_b F$, if the following hold:*

1. *whenever $E \xrightarrow{a} E'$ then $F \xrightarrow{a} F'$ such that $E' \approx_b F'$;*
2. *whenever $F \xrightarrow{a} F'$ then $E \xrightarrow{a} E'$ such that $E' \approx_b F'$;*
3. *$E \triangleright X$ if and only if $F \triangleright X$.*

Two expressions E and F are rooted delay bisimilar, notation $E =_d F$, if the following hold:

1. *whenever $E \xrightarrow{a} E'$ then $F \Longrightarrow\!\xrightarrow{a} F'$ such that $E' \approx_d F'$;*
2. *whenever $F \xrightarrow{a} F'$ then $E \Longrightarrow\!\xrightarrow{a} E'$ such that $E' \approx_d F'$;*
3. *if $E \triangleright X$ then $F \tilde{\triangleright} X$;*
4. *if $F \triangleright X$ then $E \tilde{\triangleright} X$.*

Two expressions E and F are rooted η-bisimilar, notation $E =_\eta F$, if the following hold:

1. *whenever $E \xrightarrow{a} E'$ then $F \xrightarrow{a}\!\Longrightarrow F'$ such that $E' \approx_\eta F'$;*
2. *whenever $F \xrightarrow{a} F'$ then $E \xrightarrow{a}\!\Longrightarrow E'$ such that $E' \approx_\eta F'$;*
3. *$E \triangleright X$ if and only if $F \triangleright X$.*

Two expressions E and F are rooted weak bisimilar, notation $E =_w F$, if the following hold:

1. *whenever $E \xrightarrow{a} E'$ then $F \Longrightarrow\!\xrightarrow{a}\!\Longrightarrow F'$ such that $E' \approx_w F'$;*
2. *whenever $F \xrightarrow{a} F'$ then $E \Longrightarrow\!\xrightarrow{a}\!\Longrightarrow E'$ such that $E' \approx_w F'$;*
3. *if $E \triangleright X$ then $F \tilde{\triangleright} X$;*
4. *if $F \triangleright X$ then $E \tilde{\triangleright} X$.*

Thus defined, it is easy to prove that the four rooted relations are all congruence relations on the constructions of expressions, and that they are in fact the weakest congruences included in the respective equivalences. From now on we will call $=_b$ branching congruence, $=_d$ delay congruence, $=_\eta$ η-congruence, and $=_w$ weak congruence.

3 Axiomatisation Hierarchy

In [8] van Glabbeek presented an inference system for branching congruence $=_b$. The following is the set of axioms and rules of the inference system (with slight simplification), besides the rules for equational reasoning (reflexivity, symmetry, transitivity, and substituting equal for equal):

S1	$E + F = F + E$	
S2	$E + (F + G) = (E + F) + G$	
S3	$E + E = E$	
S4	$E + \mathbf{0} = E$	
B	$a.(\tau.(E + F) + F) = a.(E + F)$	
R1	$\mu X.E = E\{\mu X.E/X\}$	
R2	if $F = E\{F/X\}$ then $F = \mu X.E$	provided X is guarded in E
R3	$\mu X.(X + E) = \mu X.E$	
R4	$\mu X.(\tau.(\tau.E + F) + G) = \mu X.(\tau.(E + F) + G)$	provided X is unguarded in E
R5	$\mu X.(\tau.(X + E) + F) = \mu X.(\tau.(E + F) + F)$	

We write $\mathbf{SBR} \vdash E = F$ if $E = F$ can be inferred using the above axioms and rules through equational reasoning, where **SBR** stands for the axioms S1-S4 plus axiom B plus rule and axioms R1-R5. Often we will omit **SBR** and just write $\vdash E = F$.

To obtain a complete axiomatisation for $=_d$, one only needs to add the following axiom T2 into the inference system **SBR**:

T2 $\tau.E + E = \tau.E$.

We call the resulting system **SBRT2**, and write $\mathbf{T2} \vdash E = F$ if $E = F$ can be inferred in **SBRT2**.

To obtain a complete axiomatisation for $=_\eta$, one only needs to add the following axiom T3 into the inference system **SBR**:

T3 $a.(E + \tau.F) + a.F = a.(E + \tau.F)$.

We call the resulting system **SBRT3**, and write $\mathbf{T3} \vdash E = F$ if $E = F$ can be inferred in **SBRT3**.

Now to obtain a complete axiomatisation for $=_w$, we can add both T2 and T3 into **SBR**. We call the resulting inference system **SBRT2T3**, and write $\mathbf{T2T3} \vdash E = F$ if $E = F$ can be inferred in **SBRT2T3**.

The name T2 and T3 are from the famous three τ-laws by Hennessy and Milner [3], where they proposed T2 and T3 together with T1: $a.\tau.E = a.E$, to make a complete axiomatisation with respect to $=_w$ for the set of CCS expressions without recursion. In **SBR**, T1 can be easily inferred from B and S4 ($\vdash a.\tau.E = a.(\tau.(E + \mathbf{0}) + \mathbf{0}) = a.(E + \mathbf{0}) = a.E$). Thus T1 is relegated to a theorem in **SBR** as stated in the following lemma, and can be spared from the axioms.

▶ **Lemma 7.** *Let $E \in \mathcal{E}$, then $\vdash a.\tau.E = a.E$.*

The following theorem states the soundness of the inference systems with respect to corresponding congruences.

▶ **Theorem 8.** *Let $E, F \in \mathcal{E}$.*

1. *if $\vdash E = F$ then $E =_b F$;*
2. *if $\mathbf{T2} \vdash E = F$ then $E =_d F$;*
3. *if $\mathbf{T3} \vdash E = F$ then $E =_\eta F$;*
4. *if $\mathbf{T2T3} \vdash E = F$ then $E =_w F$.*

A standard strategy of proving soundness of an axiomatisation with respect to some congruence is first to prove that all the axioms are sound and then to prove that all the inference rules preserve soundness. In this paper, since we concentrate on completeness, we will skip the proof of this theorem. A detailed proof of soundness of **SBR** with respect to $=_b$ can be found in [8] (Corollary 1).

Because the axioms and rules of **SBR** are also axioms and rules of **SBRT2** and of **SBRT3** and of **SBRT2T3**, the following are relationships about these axiomatisations obviously hold.

▶ **Theorem 9.** *Let $E, F \in \mathcal{E}$. If $\vdash E = F$ then $\boldsymbol{T2} \vdash E = F$ and $\boldsymbol{T3} \vdash E = F$ and $\boldsymbol{T2T3} \vdash E = F$.*

The original axioms of the inference system for $=_b$ in [8] also include the equation $\mu X.(\tau.(X + E) + \tau.(X + F) + G) = \mu X.(\tau.(X + E + F) + G)$. However it turns out that this equation can be inferred from S1-S3 and R4-R5, thus it is a derived rule and can be omitted from the set of axioms.

▶ **Definition 10.** *A guarded recursion is an expression of the form $\mu X.E$ where X is guarded in E. An expression is said guarded if every recursive subexpression in it is a guarded recursion.*

▶ **Theorem 11.** *Let $E \in \mathcal{E}$. Then there is a guarded expression E' such that $\vdash E = E'$.*

Proof. Has been proved by van Glabbeek in [8]. ◀

▶ **Lemma 12.** *Let $E \in \mathcal{E}$, E guarded.*
1. *If $E' \in \mathcal{E}_E$ (the transition closure of E), then E' is also guarded.*
2. *There is no infinite τ-transition sequence starting from E ($\xrightarrow{\tau}$ is well-founded).*

Proof. See the proof of Lemma 2 in [8]. ◀

▶ **Lemma 13.** *Let $E \in \mathcal{E}$. Then $\{a.E' \mid E \Longrightarrow\xrightarrow{a}\Longrightarrow E'\}$ and $\{W \mid E \tilde{\triangleright} W\}$ are finite sets.*

Proof. It is easy to prove by induction on the rules which defines $\triangleright$ and $\longrightarrow$, that if $E \triangleright W$ then $W \in fv(E)$, and if $E \xrightarrow{a} E'$ then $fv(E') \subseteq fv(E)$. Thus if $E \tilde{\triangleright} X$ i.e. there is E' such that $E \Longrightarrow E', E' \triangleright X$, then $X \in fv(E') \subseteq fv(E)$. So $\{W \mid E \tilde{\triangleright} W\} \subseteq fv(E)$. Since $fv(E)$ is a finite set, so is $\{W \mid E \tilde{\triangleright} W\}$.

For $E \in \mathcal{E}$, define $sort(E)$ inductively on the structure of E such that:

$$sort(\mathbf{0}) = sort(X) = \emptyset, \qquad sort(E + F) = sort(E) \cup sort(F),$$
$$sort(a.E) = sort(E) \cup \{a\}, \qquad sort(\mu X.E) = sort(E).$$

Then it is easy to prove by induction on the structure of E that $sort(E)$ is a finite set. Next, we prove by induction on the rules which defines the transition relation that if $E \xrightarrow{a} E'$ then $a \in sort(E)$ and $sort(E') \subseteq sort(E)$. Thus it follows that if $E \Longrightarrow\xrightarrow{a}\Longrightarrow E'$ then $a \in sort(E)$. Now to prove that $\{a.E' \mid E \Longrightarrow\xrightarrow{a}\Longrightarrow E'\}$ is a finite set, we note that $\{a.E' \mid E \Longrightarrow\xrightarrow{a}\longrightarrow E'\} \subseteq \{a.E' \mid a \in sort(E), E' \in \mathcal{E}_E\}$. Since $sort(E)$ and $\mathcal{E}_E$ are finite sets, so is $\{a.E' \mid a \in sort(E), E' \in \mathcal{E}_E\}$. Thus $\{a.E' \mid E \Longrightarrow\xrightarrow{a}\Longrightarrow E'\}$ is a finite set. ◀

For a finite set $S = \{E_1, \ldots, E_n\}$ of expressions, let ΣS be an abbreviation for $E_1 + \ldots + E_n$. This notation is justified by the axioms S1-S4. Then with Lemma 13, the Σ notation used in the following lemma is meaningful.

▶ **Lemma 14.** *Let $E \in \mathcal{E}$. Then $\vdash E = \Sigma\{a.E' \mid E \xrightarrow{a} E'\} + \Sigma\{W \mid E \triangleright W\}$.*

Proof. It was proved in [8] (Lemma 6). ◀

▶ **Lemma 15.** *Let $E \in \mathcal{E}$. Then*
1. *if $E \xrightarrow{a} E'$ then $\vdash E = E + a.E'$;*
2. *if $E \triangleright X$ then $\vdash E = E + X$;*
3. *if $E \Longrightarrow \xrightarrow{a} E'$ then $\boldsymbol{T2} \vdash E = E + a.E'$;*
4. *if $E \tilde{\triangleright} W$, then $\boldsymbol{T2} \vdash E = E + W$;*
5. *if $E \xrightarrow{a} \Longrightarrow E'$ then $\boldsymbol{T3} \vdash E = E + a.E'$;*
6. *if $E \Longrightarrow \xrightarrow{a} \Longrightarrow E'$ then $\boldsymbol{T2T3} \vdash E = E + a.E'$.*

Proof. 1. and 2. immediately follows from Lemma 14 and S3.

The rest can be proved by induction on the length of the τ-transition sequence in $\Longrightarrow$. Here we prove 3. as follows. Suppose $E \Longrightarrow \xrightarrow{a} E'$. If the length of the τ-transition sequence in $\Longrightarrow$ is 0, then in this case $E \xrightarrow{a} E'$, which is *1.* If the length of the τ-transition sequence in $\Longrightarrow$ is greater than 0, then there is E'' such that $E \Longrightarrow \xrightarrow{\tau} E'', E'' \xrightarrow{a} E'$, and by the induction hypothesis $\mathbf{T2} \vdash E = E + \tau.E''$ and $\mathbf{T2} \vdash E'' = E'' + a.E'$. Now we have the following reasoning in **SBRT2**:

$$
\begin{aligned}
\mathbf{T2} \vdash E &= E + \tau.E'' && \text{(IH)}\\
&= E + \tau.(E'' + a.E') && \text{(IH)}\\
&= E + \tau.(E'' + a.E') + E'' + a.E' && \text{(T2)}\\
&= E + \tau.(E'' + a.E') + E'' + a.E' + a.E' && \text{(S3)}\\
&= E + \tau.(E'' + a.E') + a.E' && \text{(T2)}\\
&= E + \tau.E'' + a.E'\\
&= E + a.E'
\end{aligned}
$$

◀

▶ **Lemma 16.** *Let $E \in \mathcal{E}$. Then*
1. *$\boldsymbol{T2} \vdash E = \Sigma\{a.E' \mid E \Longrightarrow \xrightarrow{a} E'\} + \Sigma\{W \mid E \tilde{\triangleright} W\}$;*
2. *$\boldsymbol{T3} \vdash E = \Sigma\{a.E' \mid E \xrightarrow{a} \Longrightarrow E'\} + \Sigma\{W \mid E \triangleright W\}$;*
3. *$\boldsymbol{T2T3} \vdash E = \Sigma\{a.E' \mid E \Longrightarrow \xrightarrow{a} \Longrightarrow E'\} + \Sigma\{W \mid E \tilde{\triangleright} W\}$.*

Proof. For 1. we have the following reasoning in **SBRT2**:

$$
\begin{aligned}
\mathbf{T2} \vdash E &= E + \Sigma\{a.E' \mid E \Longrightarrow \xrightarrow{a} E'\} + \Sigma\{W \mid E \tilde{\triangleright} W\} && \text{(3. and 4. of Lemma 15)}\\
&= \Sigma\{a.E' \mid E \xrightarrow{a} E'\} + \Sigma\{W \mid E \triangleright W\}\\
&\quad + \Sigma\{a.E' \mid E \Longrightarrow \xrightarrow{a} E'\} + \Sigma\{W \mid E \tilde{\triangleright} W\} && \text{(Lemma 14)}\\
&= \Sigma\{a.E' \mid E \Longrightarrow \xrightarrow{a} E'\} + \Sigma\{W \mid E \tilde{\triangleright} W\} && \text{(S3)}
\end{aligned}
$$

2. and 3. can be proved as 1. by using Lemma 15 and Lemma 14. ◀

4 Recursive Specifications and Provability

We now describe the recursive equations mentioned in the introduction. They are formally called recursive specifications.

▶ **Definition 17.** *A* recursive specification *S is a finite set of equations*

$$\{X_i = F_i \mid i = 0, 1, \ldots, n-1\}$$

where n different variables $X_0, \ldots, X_{n-1}$ are called the formal variables *of S, and $F_i \in \mathcal{E}$ for $i = 0, \ldots, n-1$. For $E \in \mathcal{E}$, E is said to T-*provably solve (or satisfy) *the recursive specification S above in the variable $X_k \in V_S$ if there are expressions E_i for $i = 0, \ldots, n-1$*

with E being E_k, such that $T \vdash E_i = F_i\{E_j/X_j \mid j = 0, \ldots, n-1\}$ holds for $i = 0, \ldots, n-1$. Define a relation $\xrightarrow{u}_S$ between the formal variables of S such that $X_i \xrightarrow{u}_S X_j$ if $F_i \triangleright X_j$. S is said to be guarded *if there is no infinite $\xrightarrow{u}_S$ transition sequence starting from any $X_i \in V_S$.*

▶ **Theorem 18.** *(Unique solution) If S is a recursive specification with formal variable X_0, then there is an expression E which **SBR**-provably solves S in X_0. Moreover if S is guarded and there are two such expressions E and F which both **SBR**-provably solve S in X_0, then $\vdash E = F$.*

Proof. In Milner [7], Theorem 4.2. ◀

▶ **Definition 19.** *Let $\mathcal{E}_0 \subseteq \mathcal{E}$. $\mathcal{E}_0$ is called a* simple set *if*

1. *$\mathcal{E}_0$ is a finite set;*
2. *$\mathcal{E}_0$ is transition closed, i.e. whenever $E \in \mathcal{E}_0$ and $E \xrightarrow{a} E'$ then $E' \in \mathcal{E}_0$;*
3. *$\xrightarrow{\tau}$ is well-founded in $\mathcal{E}_0$, i.e. there does not exist an infinite sequence of τ-transitions starting from any element in $\mathcal{E}_0$.*

The key step in our completeness proof is to show that if E, F are two expressions in the same simple set $\mathcal{E}_0$ such that $E \approx_b F$, then $\vdash \tau.E = \tau.F$ (in [2] Deng proved this *promotion lemma* for finite processes, which plays a key role in his completeness results). With this we go on to prove the completeness of **SBR** with respect to $=_b$. Since the detailed construction is quite technical, before starting we spend a few words to explain the intuition behind it. For an equivalence relation, let us say $\approx_b$, one can quotient $\mathcal{E}_0$ into equivalence classes, and then construct a minimal labeled transition system such that each equivalence class as a state of the constructed transition system mimics the behaviour of its members. If we can write expressions to describe the constructed labeled transition system, then there is a good chance that we can use the expressions to solve a recursive specification.

In the rest of this section, we will fix a simple set $\mathcal{E}_0$ and construct recursive specifications with respect to it.

Since $\mathcal{E}_0$ is a finite set, the equivalence relation $\approx_b$ partitions it into a finite number of equivalence classes. Thus, we can assume that $\approx_b$ partitions $\mathcal{E}_0$ into n $\approx_b$-equivalence classes, and we write $\{C_1, \ldots, C_n\}$ for the partition. For $E \in C_i$, E is called a *bottom element of* C_i if whenever $E \xrightarrow{\tau} E'$ then $E' \notin C_i$. That is to say, a bottom element of an equivalence class cannot remain in the same class after performing a τ-transition. Take any $E \in C_i$, if E is not a bottom element of C_i, then we can find $E' \in C_i$ such that $E \xrightarrow{\tau} E'$, and since $\xrightarrow{\tau}$ is well-founded for elements of $\mathcal{E}_0$, this cannot go on for ever, which implies that bottom elements must exist in C_i. Now let us fix n expressions $B_1, \ldots, B_n$ such that each B_i is a bottom element of C_i. For $E \in \mathcal{E}_0$, if $E \in C_i$ we call i the index of E, and we write $\iota(E)$ for the index of E. With this notation, the following are two obvious relations hold for the expressions in $\mathcal{E}_0$: A) for all $E \in \mathcal{E}_0$, $E \approx_b B_{\iota(E)}$; B) for all $E, F \in \mathcal{E}_0$, $E \approx_b F$ if and only if $\iota(E) = \iota(F)$.

▶ **Definition 20.** *Define a recursive specification $R^{\mathcal{E}_0} = \{Z_i = H_i \mid i = 1, \ldots, n\}$, where $Z_1, \ldots, Z_n \in \mathcal{V}$ are n different variables which do not occur free in any expressions in $\mathcal{E}_0$, and each H_i is defined as follows:*

$$H_i \equiv \Sigma\{a.Z_{\iota(E)} \mid B_i \xrightarrow{a} E\} + \Sigma\{W \mid B_i \triangleright W\}.$$

Now with the recursive specification $R^{\mathcal{E}_0}$ defined above, according to Theorem 18, there exist n expressions $D_1, \ldots, D_n$, which **SBR**-provably satisfy $R^{\mathcal{E}_0}$, i.e. the following holds for $i = 1, \ldots, n$:

$$\vdash D_i = H_i\{D_1, \ldots, D_n / Z_1, \ldots, Z_n\}. \qquad (1)$$

In fact these D's can be constructed so that they form a canonical solution to $R^{\mathcal{E}_0}$ in that the transition space consists of only $D_1, \ldots, D_n$, which is minimal in size since $D_i \not\approx_b D_j$ when $i \neq j$. However as we only need equality (1) without referring to the actual behaviour of the D's, here we will not further argue the canonicity of the D's (the canonicity and other properties of D's will be clear after Theorem 29 in the next section). Next we will concentrate on using the D's to solve some useful recursive specifications.

▶ **Lemma 21.** *Let $E \in \mathcal{E}_0$. If E is a bottom element, then*

$$\vdash D_{\iota(E)} = \Sigma\{a.D_{\iota(F)} \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\}.$$

Proof. Since $\vdash D_{\iota(E)} = H_{\iota(E)}\{D_i/Z_i \mid i = 1, \ldots, n\}$ (equality (1) above), to prove the lemma, we only need to establish:

$$\vdash H_{\iota(E)}\{D_i/Z_i \mid i = 1, \ldots, n\} = \Sigma\{a.D_{\iota(F)} \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\}.$$

According to Definition 20, the left hand side of the above equation is $\Sigma\{a.D_{\iota(F)} \mid B_{\iota(E)} \xrightarrow{a} F\} + \Sigma\{W \mid B_{\iota(E)} \triangleright W\}$ where $B_{\iota(E)}$ is the chosen bottom element in the equivalence class of E. As both E and $B_{\iota(E)}$ are bottom elements of the same equivalence class, a simple fact is that $E \triangleright W$ if and only if $B_{\iota(E)} \triangleright W$, and $E \xrightarrow{a} F$ if and only if $B_{\iota(E)} \xrightarrow{a} F'$ such that $\iota(F) = \iota(F')$ (this easily follows from Definition 5 and the condition that bottom elements cannot perform τ-transition without moving state to a different equivalence class). From this simple fact it follows that

$$\vdash \Sigma\{a.D_{\iota(F)} \mid B_{\iota(E)} \xrightarrow{a} F\} + \Sigma\{W \mid B_{\iota(E)} \triangleright W\} = \Sigma\{a.D_{\iota(F)} \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\}$$

as both sides has the same set of summands. ◀

▶ **Lemma 22.** *Let $E \in \mathcal{E}_0$, then*
$\vdash \tau.D_{\iota(E)} + D_{\iota(E)} = \tau.D_{\iota(E)} + D_{\iota(E)} + \Sigma\{a.D_{\iota(F)} \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\}$.

Proof. Since $\vdash D_{\iota(E)} = H_{\iota(E)}\{D_i/Z_i \mid i = 1, \ldots, n\}$ (equality (1) above), to prove the lemma we only need to establish:

$$\begin{aligned} \vdash \quad & \tau.D_{\iota(E)} + H_{\iota(E)}\{D_i/Z_i \mid i = 1, \ldots, n\} = \\ & \tau.D_{\iota(E)} + H_{\iota(E)}\{D_i/Z_i \mid i = 1, \ldots, n\} + \Sigma\{a.D_{\iota(F)} \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\}. \end{aligned}$$

For that, with axioms S1, S2, S3, we need to show that the extra summands on the right hand side of the equality in $\Sigma\{a.D_{\iota(F)} \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\}$ are also summands in $\tau.D_{\iota(E)} + H_{\iota(E)}\{D_i/Z_i \mid i = 1, \ldots, n\}$, which can be verified with the condition that $E \approx_b B_{\iota(E)}$ and $B_{\iota(E)}$ is a bottom element of $C_{\iota(E)}$. Consider the summand W with $E \triangleright W$. Since $E \approx_b B_{\iota(E)}$, there exists $E' \in C_{\iota(E)}$ such that $B_{\iota(E)} \Longrightarrow E', E' \triangleright W$. Because $B_{\iota(E)}$ is a bottom element which cannot perform any τ while staying within $C_{\iota(E)}$, E' must be $B_{\iota(E)}$, thus according to the definition of H_i in Definition 20 W is a summand in $H_{\iota(E)}$, and also in $H_{\iota(E)}\{D_i/Z_i \mid i = 1, \ldots, n\}$. Consider the summand $a.D_{\iota(F)}$ with $E \xrightarrow{a} F$. Since $E \approx_b B_{\iota(E)}$, then either $a = \tau$ and $F \in B_{\iota(E)}$ and in this case $a.D_{\iota(F)}$ is just $\tau.D_{\iota(E)}$, or

there exists $E' \in B_{\iota(E)}$ such that $B_{\iota(E)} \Longrightarrow E', E' \xrightarrow{a} F'$ such that $F \approx_b F'$. Because $B_{\iota(E)}$ is a bottom element, E' must be $B_{\iota(E)}$, according to the definition of H_i in Definition 20, $a.Z_{\iota(F')}$ is a summand in $H_{\iota(E)}$, thus $a.D_{\iota(F')}$ (which in this case is the same as $a.D_{\iota(F)}$) is a summand in $H_{\iota(E)}\{D_i/Z_i \mid i = 1, \ldots, n\}$. ◀

▶ **Lemma 23.** *Let $E \in \mathcal{E}_0$, then $\vdash \tau.D_{\iota(E)} = \tau.(\Sigma\{a.D_{\iota(F)} \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\})$.*

Proof. If E is a bottom element, then the lemma follows from Lemma 21.

If E is not a bottom element, i.e. there is E' such that $E \xrightarrow{\tau} E'$ and $\iota(E) = \iota(E')$, then

$$
\begin{aligned}
&\vdash \tau.D_{\iota(E)} = \tau.(\tau.D_{\iota(E)} + D_{\iota(E)}) && \text{(B)}\\
&= \tau.(\tau.D_{\iota(E)} + D_{\iota(E)} + \Sigma\{a.D_{\iota(F)} \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\}) && \text{(Lemma 22)}\\
&= \tau.(\tau.(\tau.D_{\iota(E)} + D_{\iota(E)} + \Sigma\{a.D_{\iota(F)} \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\})\\
&\quad + \Sigma\{a.D_{\iota(F)} \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\}) && \text{(B)}\\
&= \tau.(\tau.(\tau.D_{\iota(E)} + D_{\iota(E)}) + \Sigma\{a.D_{\iota(F)} \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\}) && \text{(Lemma 22)}\\
&= \tau.(\tau.D_{\iota(E)} + \Sigma\{a.D_{\iota(F)} \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\}) && \text{(B)}\\
&= \tau.(\Sigma\{a.D_{\iota(F)} \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\}). && (\dagger)
\end{aligned}
$$

† is because $\iota(E) = \iota(E')$ and $\tau.D_{\iota(E)}$ is the same as $\tau.D_{\iota(E')}$, while $\tau.D_{\iota(E')}$ is a summand in $\Sigma\{a.D_{\iota(F)} \mid E \xrightarrow{a} F\}$. ◀

▶ **Lemma 24** (Promotion). *Let $E_1, E_2 \in \mathcal{E}_0$. If $E_1 \approx_b E_2$ then $\vdash \tau.E_1 = \tau.E_2$.*

Proof. First note that for each $E \in \mathcal{E}_0$ the following holds:

$$
\begin{aligned}
&\vdash \tau.D_{\iota(E)} = \tau.(\Sigma\{a.D_{\iota(F)} \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\}) && \text{(Lemma 23)}\\
&= \tau.(\Sigma\{a.\tau.D_{\iota(F)} \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\}). && \text{(Lemma 7)}
\end{aligned}
$$

On the other hand

$$
\begin{aligned}
&\vdash \tau.E = \tau.(\Sigma\{a.F \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\}) && \text{(Lemma 14)}\\
&= \tau.(\Sigma\{a.\tau.F \mid E \xrightarrow{a} F\} + \Sigma\{W \mid E \triangleright W\}). && \text{(Lemma 7)}
\end{aligned}
$$

Thus, for each $E \in \mathcal{E}_0$ it holds that $\vdash \tau.D_{\iota(E)} = \tau.E$, because both $\tau.D_{\iota(E)}$ and $\tau.E$ **SBR**-provably solve the following guarded recursive specification on X_E:

$$S = \{X_E = \tau.(\Sigma\{a.X_F \mid E \xrightarrow{a} F\}) + \Sigma\{W \mid E \triangleright W\} \mid E \in \mathcal{E}_0\}.$$

The well-foundedness of $\xrightarrow{u}_S$ easily follows from the well-foundedness of $\xrightarrow{\tau}$ in $\mathcal{E}_0$ which is a simple set. Thus S is a guarded recursive specification. Now if $E_1 \approx_b E_2$, then $\iota(E_1) = \iota(E_2)$. Thus $\vdash \tau.E_1 = \tau.D_{\iota(E_1)} = \tau.D_{\iota(E_2)} = \tau.E_2$. ◀

▶ **Theorem 25** (Completeness of **SBR**). *Let $E, F \in \mathcal{E}$. If $E =_b F$ then $\vdash E = F$.*

Proof. By Lemma 11 there exist guarded expressions E', F' such that $\vdash E = E', \vdash F = F'$. By the soundness of **SBR**, $E' =_b E =_b F =_b F'$. Let $\mathcal{E}_0 = \mathcal{E}_{E'} \cup \mathcal{E}_{F'}$, where $\mathcal{E}_{E'}, \mathcal{E}_{F'}$ are the sets of expressions reachable from E' and F' respectively. $\mathcal{E}_0$ is obviously transition closed. By Theorem 4 $\mathcal{E}_0$ is a finite set. By Lemma 12 $\xrightarrow{\tau}$ is well-founded in $\mathcal{E}_0$. Thus $\mathcal{E}_0$ is a simple set. By Lemma 14 $\vdash F' = \Sigma S_1 + \Sigma S_2$ where $S_1 = \{a.F'' \mid F' \xrightarrow{a} F''\}, S_2 = \{W \mid F' \triangleright W\}$. According to Lemma 13 S_1, S_2 are finite sets, let $S_1 = \{a_1.F_1, \ldots, a_k.F_k\}$, $S_2 = \{W_1, \ldots, W_m\}$.

Because $E' =_b F'$, for each $a_i.F_i \in S_1$ there is E'' such that $E' \xrightarrow{a_i} E''$ and $E'' \approx_d F_i$, then by Lemma 7 and Lemma 24 (note that $E'', F_i \in \mathcal{E}_0$) $\vdash a_i.E'' = a_i.\tau.E'' = a_i.\tau.F_i = a_i.F_i$, and by Lemma 15 $\vdash E' = E' + a_i.E''$. It follows that $\vdash E' = E' + a_i.F_i(A)$. For each $W_j \in S_2$, since $E' =_b F'$, $E' \triangleright W_j$, then by Lemma 15 $\vdash E' = E' + W_j(B)$. After these preparation we have the following reasoning in **SBR**:

$$\begin{aligned}
\vdash E' + F' = \;& E' + \Sigma_{i=1}^{k} a_i.F_i + \Sigma_{j=1}^{m} W_j && \text{(Lemma 14)}\\
=\;& E' + a_1.F_1 + \Sigma_{i=2}^{k} a_i.F_i + \Sigma_{j=1}^{m} W_j &&\\
=\;& E' + \Sigma_{i=2}^{k} a_i.F_i + \Sigma_{j=1}^{m} W_j && \text{(A above)}\\
=\;& E' + \Sigma_{j=1}^{m} W_j && \text{(after } k \text{ steps)}\\
=\;& E' + W_1 + \Sigma_{j=2}^{m} W_j &&\\
=\;& E' + \Sigma_{j=2}^{m} W_j && \text{(B above)}\\
=\;& E' && \text{(after } m \text{ steps)}
\end{aligned}$$

In the same way we can prove $\vdash E' + F' = F'$. Thus $\vdash E' = F'$, and $\vdash E = F$. ◀

To summarize, so far we proved the completeness of **SBR** with respect to $=_b$ by proving promotion lemma (Lemma 24) through constructing solution to the recursive specification $R^{\mathcal{E}_0}$. This plan of proof can be easily adapted to work for the completeness of **SBRT2** w.r.t $=_d$, that of **SBRT3** w.r.t $=_\eta$, and that of **SBRT2T3** w.r.t. $=_w$. Very few adjustment is required, including to quotient with proper equivalence (of course), to change the transition $B_i \xrightarrow{a} E$ in $R^{\mathcal{E}_0}$ (Definition 20) accordingly to $B_i \Longrightarrow \xrightarrow{a} E$ and $B_i \xrightarrow{a} \Longrightarrow E$ and $B_i \Longrightarrow \xrightarrow{a} \Longrightarrow E$, and in establishing the corresponding promotion lemma to use results in Lemma 16 instead of Lemma 14 to work out proper recursive specifications corresponding to the S in Lemma 24. Instead of going through the same plan to establish the other three completeness results, here we shall satisfy ourselves by applying the ideas which are needed in proving the completeness of **SBRT2T3** on deriving equality in **SBRT2T3** of two concrete expressions.

▶ **Example 26.** Let E and F be $b.\mu X.(a.a.X + \tau.\mu Y.a.Y)$ and $b.\mu Z.\tau.a.Z$ respectively, in this example we show **SBRT2T3** $\vdash E = F$. The two expressions are modified from an example in [6] (the modification is to prevent F being able to solve the recursive specification of E easily). By 3. of Lemma 16 the following equalities are provable in **SBRT2T3**:

$$\begin{aligned}
\mathbf{T2T3} \vdash\;& E &=\;& b.\mu X.(a.a.X + \tau.\mu Y.a.Y) + b.\mu Y.a.Y\\
\mathbf{T2T3} \vdash\;& \mu X.(a.a.X + \tau.\mu Y.a.Y) &=\;& a.a.\mu X.(a.a.X + \tau.\mu Y.a.Y) + \tau.\mu Y.a.Y + a.\mu Y.a.Y\\
\mathbf{T2T3} \vdash\;& a.\mu X.(a.a.X + \tau.\mu Y.a.Y) &=\;& a.\mu X.(a.a.X + \tau.\mu Y.a.Y) + a.\mu Y.a.Y\\
\mathbf{T2T3} \vdash\;& \mu Y.a.Y &=\;& a.\mu Y.a.Y\\
\mathbf{T2T3} \vdash\;& F &=\;& b.\mu Z.\tau.a.Z + b.a.\mu Z.\tau.a.Z\\
\mathbf{T2T3} \vdash\;& \mu Z.\tau.a.Z &=\;& \tau.a.\mu Z.\tau.a.Z + a.\mu Z.\tau.a.Z + a.a.\mu Z.\tau.a.Z\\
\mathbf{T2T3} \vdash\;& a.\mu Z.\tau.a.Z &=\;& a.\mu Z.\tau.a.Z + a.a.\mu Z.\tau.a.Z
\end{aligned}$$

So, E and F are **SBRT2T3**-provably solve the following guarded recursive specification S (below on the left) in X_1 and Y_1 respectively, with $E_2 \equiv \mu X.(a.a.X + \tau.\mu Y.a.Y), E_3 \equiv a.\mu X.(a.a.X + \tau.\mu Y.a.Y), E_4 \equiv \mu Y.a.Y$ provably solving S in X_2, X_3, X_4 respectively, and $F_2 \equiv \mu Z.\tau.a.Z, F_3 \equiv a.\mu Z.\tau.a.Z$ provably solving S in Y_2, Y_3 respectively:

$$
\begin{array}{lrcl}
S: & X_1 & = & b.X_2 + b.X_4 \\
& X_2 & = & a.X_3 + \tau.X_4 + a.X_4 \\
& X_3 & = & a.X_2 + a.X_4 \\
& X_4 & = & a.X_4 \\
& Y_1 & = & b.Y_2 + b.Y_3 \\
& Y_2 & = & \tau.Y_3 + a.Y_2 + a.Y_3 \\
& Y_3 & = & a.Y_2 + a.Y_3
\end{array}
\qquad
\begin{array}{lrcl}
T: & X_1 & = & b.X_2 + b.X_4 \\
& X_2 & = & \tau.(a.X_3 + \tau.X_4 + a.X_4) \\
& X_3 & = & \tau.(a.X_2 + a.X_4) \\
& X_4 & = & \tau.a.X_4 \\
& Y_1 & = & b.Y_2 + b.Y_3 \\
& Y_2 & = & \tau.(\tau.Y_3 + a.Y_2 + a.Y_3) \\
& Y_3 & = & \tau.(a.Y_2 + a.Y_3)
\end{array}
$$

Then by using Lemma 7, we easily obtain that E and F **SBRT2T3**-provably solve the guarded recursive specification T (above on the right) in X_1 and Y_1 respectively, with $\tau.E_2, \tau.E_3, \tau.E_4$ provably solving T in X_2, X_3, X_4, and $\tau.F_2, \tau.F_3$ in Y_2, Y_3 respectively.

On the other hand, the minimal reachable state space containing E and F is $\{E, E_2, E_3, E_4, F, F_2, F_3\}$, which is divided into two $\approx_w$-equivalence classes $C_1 = \{E, F\}$ with bottom element F, and $C_2 = \{E_2, E_3, E_4, F_2, F_3\}$ with bottom element F_3. Using the construction of Definition 20, from C_1 and C_2 we obtain recursive specification $R = \{Z_1 = b.Z_2, Z_2 = a.Z_2\}$, which is provably solved by $b.\mu Z.a.Z$ and $\mu Z.a.Z$ in Z_1 and Z_2 respectively in **SBRT2T3**. Now it is easy to see that $b.\mu Z.a.Z$ provably solve T in both X_1 and Y_1, with $\tau.\mu Z.a.Z$ provably solving T in all X_2, X_3, X_4, Y_2, Y_3. Finally with E and $b.\mu Z.a.Z$ both provably solving T in X_1 and F and $b.\mu Z.a.Z$ both provably solving T in Y_1 where T is guarded, we obtain $\mathbf{T2T3} \vdash E = b.\mu Z.a.Z$ and $\mathbf{T2T3} \vdash F = b.\mu Z.a.Z$, hence $\mathbf{T2T3} \vdash E = F$.

5 Saturation Results and Completeness of All Axiomatisations

With the completeness result of **SBR** with respect to $=_b$, we can also use the method employed in [1] to obtain the completeness results of the extended axiomatisations, i.e. by using the notion of saturation to reduce the completeness of the extended axiomatisations to the established completeness of **SBR** w.r.t $=_b$. With this approach, besides the completeness results, the saturation results (Theorem 30) are interesting in their own rights.

▶ **Definition 27.** *A set of expressions S is called* saturated *if for each $E \in S$:*

1. *whenever $E \xrightarrow{a}\xrightarrow{\tau} F$ then $E \xrightarrow{a} F$;*
2. *whenever $E \xrightarrow{\tau}\xrightarrow{a} F$ then $E \xrightarrow{a} F$;*
3. *whenever $E \xrightarrow{\tau} E'$ and $E' \rhd X$ then $E \rhd X$.*

S is called η-saturated if 1. is required to hold for each $E \in S$, and S is called d-saturated if 2. and 3. are required to hold for each $E \in S$.

An expression $E \in \mathcal{E}$ is called saturated *(and* η-saturated, *d-*saturated *respectively) if there is some $S \subseteq \mathcal{E}$ with $E \in S$ such that S is transition closed and saturated (and η-saturated, d-saturated respectively).*

▶ **Theorem 28.** *Let $E, F \in \mathcal{E}$.*

1. *If E and F are d-saturated, then $E =_d F$ implies $E =_b F$;*
2. *If E and F are η-saturated, then $E =_\eta F$ implies $E =_b F$;*
3. *If E and F are saturated, then $E =_w F$ implies $E =_b F$.*

Proof. See [1], Theorem 4.6. ◀

To prove the main result of this section, Theorem 30, we need the following theorem which is a strengthened version of the existence part in Theorem 18 (a proof can be found in the appendix).

▶ **Theorem 29.** *Let $\{X_i = F_i \mid i = 1, \ldots, n\}$ be a recursive specification. Then there exist n expressions $E_1, \ldots, E_n$ such that E_i **SBR**-provably solves $\{X_i = F_i \mid i = 1, \ldots, n\}$ in variable X_i. Moreover, the following hold for $i = 1, \ldots, n$:*

for $a \in \mathcal{A}_\tau, E \in \mathcal{E}$, $E_i \xrightarrow{a} E$ if and only if $F_i\{E_1, \ldots, E_n/X_1, \ldots, X_n\} \xrightarrow{a} E$ and for $W \in \mathcal{V}$, $E_i \triangleright W$ if and only if $F_i\{E_1, \ldots, E_n/X_1, \ldots, X_n\} \triangleright W$.

▶ **Theorem 30** (Saturation)**.** *Let E be a guarded expression. Then there exist guarded expressions E_d, E_η, and E_w, such that E_d is d-saturated and $\mathbf{T2} \vdash E = E_d$, E_η is η-saturated and $\mathbf{T3} \vdash E = E_\eta$, and E_w is saturated and $\mathbf{T2T3} \vdash E = E_w$.*

Proof. Here we only show the existence of E_w. By the same procedure we can construct E_d and E_η. As E is a guarded expression, we can show that $\mathcal{E}_E$ (the transition closure of E) is a simple set (in the same way to show that $\mathcal{E}_0$ is a simple set in the proof of Theorem 25). By Lemma 16 for each $F \in \mathcal{E}_E$ it holds that

$$\mathbf{T2T3} \vdash F = \Sigma\{a.F' \mid F \Longrightarrow\xrightarrow{a}\Longrightarrow F'\} + \Sigma\{W \mid F \triangleright\!\!\!\!\sim W\}.$$

Thus each $F \in \mathcal{E}_E$ provably solves X_F in the following recursive specification $S^{\mathcal{E}_E}$:

$$\{X_F = \Sigma\{a.X_{F'} \mid F \Longrightarrow\xrightarrow{a}\Longrightarrow F'\} + \Sigma\{W \mid F \triangleright\!\!\!\!\sim W\} \mid F \in \mathcal{E}_E\}.$$

According to Theorem 29, there exist a set of expressions $\{D_F \mid F \in \mathcal{E}_E\}$ such that $D_F \xrightarrow{a} H$ for some expression H if and only if $\Sigma\{a.D_{F'} \mid F \Longrightarrow\xrightarrow{a}\Longrightarrow F'\} + \Sigma\{W \mid F \triangleright\!\!\!\!\sim W\} \xrightarrow{a} H$ and $D_F \triangleright W$ for some $W \in \mathcal{V}$ if and only if $\Sigma\{a.D_{F'} \mid F \Longrightarrow\xrightarrow{a}\Longrightarrow F'\} + \Sigma\{W \mid F \triangleright\!\!\!\!\sim W\} \triangleright W$. Note that $\Sigma\{a.D_{F'} \mid F \Longrightarrow\xrightarrow{a}\Longrightarrow F'\} + \Sigma\{W \mid F \triangleright\!\!\!\!\sim W\} \xrightarrow{a} H$ if and only if $H \equiv D_{F'}$ for some F' such that $F \Longrightarrow\xrightarrow{a}\Longrightarrow F'$, thus $D_F \xrightarrow{a} H$ if and only if $H \equiv D_{F'}$ where $F \Longrightarrow\xrightarrow{a}\Longrightarrow F'$. Which shows that $\{D_F \mid F \in \mathcal{E}_E\}$ is transition closed. Next we show that $\{D_F \mid F \in \mathcal{E}_E\}$ is saturated. For that, take arbitrary $F \in \mathcal{E}_E$, and suppose $D_F \xrightarrow{\tau} H_1 \xrightarrow{a} H_2$. Then there is F_1 such that $F \Longrightarrow\xrightarrow{\tau}\Longrightarrow F_1$ and $H_1 \equiv D_{F_1}$ and also there is F_2 such that $F_1 \Longrightarrow\xrightarrow{a}\Longrightarrow F_2$ and $H_2 \equiv D_{F_2}$, and in this case $F \Longrightarrow\xrightarrow{a}\Longrightarrow F_2$, thus $D_F \xrightarrow{a} D_{F_2} \equiv H_2$. In the same way we can show that if $D_F \xrightarrow{a} H_1 \xrightarrow{\tau} H_2$ then $D_F \xrightarrow{a} H_2$, and if $D_F \xrightarrow{\tau} H, H \triangleright W$ then $D_F \triangleright W$. Thus $\{D_F \mid F \in \mathcal{E}_E\}$ is a saturated set, and D_E is a saturated expression. Since E is guarded, $\xrightarrow{\tau}$ is well-founded in $\mathcal{E}_E$, from which it easily follows that $S^{\mathcal{E}_E}$ is a guarded recursive specification. Now both E and D_E provably solve $S^{\mathcal{E}_E}$ in the variable X_E, and $S^{\mathcal{E}_E}$ is guarded, thus by Theorem 18 $\mathbf{T2T3} \vdash E = D_E$, and D_E is saturated since it is in the closed and saturated set $\{D_F \mid F \in \mathcal{E}_E\}$. Thus we find D_E for the wanted E_w. ◀

With Theorem 30, the rest of the completeness results follows from the completeness of **SBR** w.r.t. $=_b$.

▶ **Theorem 31.** *Let $E, F \in \mathcal{E}$.*
1. *if $E =_d F$ then $\mathbf{T2} \vdash E = F$;*
2. *if $E =_\eta F$ then $\mathbf{T3} \vdash E = F$;*
3. *if $E =_w F$ then $\mathbf{T2T3} \vdash E = F$.*

Proof. Here we only prove 1., the rest can be established in the same way. Let $E =_d F$. By Theorem 11 there exist guarded expressions E_1, F_1 such that $\vdash E = E_1, \vdash F = F_1$, thus also $\mathbf{T2} \vdash E = E_1, \mathbf{T2} \vdash F = F_1$. By Theorem 30 there exist d-saturated and guarded expressions

E_2, F_2 such that $\mathbf{T2} \vdash E_1 = E_2, \mathbf{T2} \vdash F_1 = F_2$. By the soundness of **SBRT2** (Theorem 8), $E =_d E_1 =_d E_2$ and $F =_d F_1 =_d F_2$, thus $E_2 =_d F_2$. Since E_2, F_2 are both d-saturated, by Theorem 28, $E_2 =_b F_2$ follows from $E_2 =_d F_2$. Then $\vdash E_2 = F_2$ follows from the completeness of **SBR**, thus also $\mathbf{T2} \vdash E_2 = F_2$, and $\mathbf{T2} \vdash E = E_1 = E_2 = F_2 = F_1 = F$. ◀

6 Related Work

In [5], Milner proposed a set of axioms and rules to infer strong congruence (where τ is treated just as any other action) for regular behaviours and proved the completeness of the inference system by merging the equation sets. That was the first time when the equation set merging strategy was introduced. Later in [7], Milner proposed a set of axioms and rules to infer weak congruence for regular behaviours and used the same strategy to prove the completeness of the inference system. Although our axiomatisation for $=_w$ in this paper is obtained by expanding van Glabbeek's axiomatisation for $=_b$ and has different axioms than Milner's axiomatisation for $=_w$, it can be proved that the two axiomatisations are equivalent in the sense that all the axioms and rules of one can be derived in the other axiomatisation and vise versa.

In [8], van Glabbeek proposed a complete axiomatisation of branching congruence for regular expressions. His axiomatisation is the base for the axiomatisations in the hierarchy studied in this paper. He used Milner's strategy of merging recursive equation sets (recursive specifications) to arrive at the completeness result. In the proof of the main theorem of completeness for guarded expressions, to bridge the gap between equivalence and congruence he used a construction which is an implicit form of the promotion lemma. In principle van Glabbeek's construction can be adapted to work for the completeness proof of the extended axiomatisations, however the interplay between such construction and the inevitable process of saturation could become messy when merging the recursive equation sets.

In [4], Lohrey and the co-authors presented an axiomatisation hierarchy for divergence sensitive weak congruences. They also used the strategy of merging recursive equation sets to arrive at the completeness results. It is expected that our approach could work for divergence sensitive variations of bisimulation based congruences which have not been treated here.

In [2], based on the promotion lemma, Deng presented a uniform completeness proof for the axiomatisations of five congruences: branching congruence, η-congruence, quasi-branching congruence, and weak congruence in the basic CCS without recursion. We have not treated quasi-branching congruence which is sufficiently similar to branching congruence and on which there shall be no difficulty to apply our method.

In [1], Aceto and the co-authors gave complete axiomatisations of branching, delay, weak, and η-congruences for expressions which use prefix iteration instead of recursion to generate infinite behaviours. Prefix iteration is a simpler syntax than recursion in that a normal form exists for every expression of that kind, as a result the complex strategy of joining equation sets is not needed in that case. The work about saturation results in section 6 of this paper follows closely the framework laid out in [1].

7 Conclusion

In this paper we put up a hierarchy of axiomatisations of four well-known congruences with various τ-abstract level for regular expressions, and proposed a new strategy for proving completeness which works uniformly on four axiomatisations. Instead of merging recursive equations as performed in the well-known approach proposed by Milner which usually causes

multiple increase of the number of recursive equations, in the new approach we construct canonical solutions, for which one only deals with recursive equations not exceeding the original number. We hope that this will set up a foundation for more feasible implementation of automated proof tools, which after inputting two expressions will automatically output a shorter formal proof of their equality, if any proof exists.

References

1 L. Aceto, R. van Glabbeek, W. Fokkink, and A. Ingĺfsdóttir. Axiomatizing prefix itteration with silent steps. *Information and computation*, 127(1):26–40, 1996. doi:10.1006/inco.1996.0047.

2 Y. Deng. A simple completeness proof for the axiomatisations of weak behavioural equivalences. *Bulletin of the European Association for Theoretical Computer Science*, 172:359–397, 2007.

3 M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the Association for Computing Machinery*, 32(1):137–161, 1985. doi:10.1145/2455.2460.

4 M. Lohrey, P.R. D'Argenio, and H. Hermanns. Axiomatising divergence. *Information and computation*, 203:115–144, 2005. doi:10.1016/j.ic.2005.05.007.

5 R. Milner. A complete inference system for a class of regular behaviours. *Journal of computer and system sciences*, 28:439–466, 1984. doi:10.1016/0022-0000(84)90023-0.

6 R. Milner. *Communication and Concurrency*. Prentice–Hall, 1989.

7 R. Milner. A complete axiomatisation system for observational congruence of finite-state behaviours. *information and computation*, 81:227–247, 1989. doi:10.1016/0890-5401(89)90070-9.

8 R. J. van Glabbeek. A complete axiomatisation for branching bisimulation congruence of finite-state behaviours. *In A.M Borzyszkowski & S. Sokolowski, editors: Proceedings 18th International Symposium on Mathematical Foundations of Computer Science, MFCS'93, Gdansk, Opland, August/September 1993, LNCS 711, Springer*, pages 473–484, 1993. doi:10.1007/3-540-57182-5_39.

9 R. J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of the Association for Computing Machinery*, 43(3):555–600, 1996. doi:10.1145/233551.233556.

10 D. Walker. Bisimulation and divergence. *Information and computation*, 85:220–241, 1990. doi:10.1016/0890-5401(90)90048-M.

A Proof of Theorem 29

In the proof we need to use the lemma of substitution, which we state as follows without a proof. The proof is a routine syntax analysis.

▶ **Lemma 32** (Substitution). *Let $E, F, E_1, \ldots, E_n \in \mathcal{E}$, $X, X_1, \ldots, X_n$ be variables which are pairwise different. Then the following two equalities hold:*

$$E\{F/X\}\{E_1/X_1, \ldots, E_n/X_n\} \equiv E\{F\{E_1/X_1, \ldots, E_n/X_n\}/X, E_1/X_1, \ldots, E_n/X_n\},$$
$$E\{E_1/X_1, \ldots, E_n/X_n\}\{F/X\} \equiv E\{E_1\{F/X\}/X_1, \ldots, E_n\{F/X\}/X_n, F/X\}.$$

Proof of Theorem 29. Let us name the recursive specification S. First we prove the following for each E_i by induction on the size of S (i.e. the number of equations in S):

1. for $a \in \mathcal{A}_\tau, E \in \mathcal{E}$, $E_i \xrightarrow{a} E$ if and only if $F_i\{E_1, \ldots, E_n/X_1, \ldots, X_n\} \xrightarrow{a} E$ and for $W \in \mathcal{V}$, $E_i \triangleright W$ if and only if $F_i\{E_1, \ldots, E_n/X_1, \ldots, X_n\} \triangleright W$;
2. $fv(E_i) \subseteq \bigcup\{fv(F_j) \mid 1 \leq j \leq n\} - \{X_1, \ldots, X_n\}$.

Once this is established, E_i provably solves $\{X_i = F_i \mid i = 1, \ldots, n\}$ in variable X_i follows from 1. and Lemma 14.

For the base case S has just one equation, let E_1 be $\mu X_1.F_1$. Then 1. holds by the fact that for $a \in \mathcal{A}_\tau$, $E \in \mathcal{E}$, $\mu X_1.F_1 \xrightarrow{a} E$ if and only if $F_1\{\mu X_1.F_1/X_1\} \xrightarrow{a} E$, which follows from the operational semantics in Definition 1., 2. holds because $fv(\mu X_1.F_1) = fv(F_1) - \{X_1\}$.

For the induction step, let

$$S' = \{X_i = F_i\{\mu X_n.F_n/X_n\} \mid i = 1, \ldots, n-1\}.$$

Then S' is a recursive specification of size $n-1$. By the induction hypothesis there exist $n-1$ expressions $E_1, \ldots, E_{n-1}$ such that the following hold for each $1 \leq i \leq n-1$:

3. for $a \in \mathcal{A}_\tau, E \in \mathcal{E}$, $E_i \xrightarrow{a} E$ iff $F_i\{\mu X_n.F_n/X_n\}\{E_1/X_1, \ldots, E_{n-1}/X_{n-1}\} \xrightarrow{a} E$;

4. $fv(E_i) \subseteq \bigcup\{fv(F_j\{\mu X_n.F_n/X_n\}) \mid 1 \leq j \leq n-1\} - \{X_1, \ldots, X_{n-1}\}$.

Now let $E_n \equiv \mu X_n.F_n\{E_1/X_1, \ldots, E_{n-1}/X_{n-1}\}$, then by the operational semantics in Definition 1 $E_n \xrightarrow{a} E$ if and only if $F_n\{E_1/X_1, \ldots, E_{n-1}/X_{n-1}\}\{E_n/X_n\} \xrightarrow{a} E$. By the lemma of substitution, $F_n\{E_1/X_1, \ldots, E_{n-1}/X_{n-1}\}\{E_n/X_n\} \equiv F_n\{E_1/X_1, \ldots, E_n/X_n\}$ (note that X_n is not free in $F_j\{\mu X_n.F_n/X_n\}$ for $1 \leq j \leq n-1$, with condition 4) above X_n is not a free variable in E_i for $i = 1, \ldots, n-1$, thus $E_i\{E_n/X_n\} \equiv E_i$). Again by the lemma of substitution, we also have the following equalities for $1 \leq i \leq n-1$:

$$\begin{aligned}
&F_i\{\mu X_n.F_n/X_n\}\{E_1/X_1, \ldots, E_{n-1}/X_{n-1}\} \\
&\equiv F_i\{\mu X_n.F_n\{E_1/X_1, \ldots, E_{n-1}/X_{n-1}\}/X_n, E_1/X_1, \ldots, E_{n-1}/X_{n-1}\} \\
&\equiv F_i\{E_1/X_1, \ldots, E_n/X_n\}
\end{aligned}$$

together with 3), with $E_1, \ldots, E_n$ we arrive at the claim 1. for S. Direct calculation gives $fv(F_j\{\mu X_n.F_n/X_n\}) \subseteq (fv(F_j) \cup fv(F_n)) - \{X_n\}$, thus from 4) the following hold for $1 \leq i \leq n-1$:

$$\begin{aligned}
&fv(E_i) \subseteq \bigcup\{(fv(F_j) \cup fv(F_n)) - \{X_n\} \mid 1 \leq j \leq n-1\} - \{X_1, \ldots, X_{n-1}\} \subseteq \\
&\bigcup\{fv(F_j) \mid 1 \leq j \leq n\} - \{X_1, \ldots, X_n\}.
\end{aligned}$$

Hence we arrive at the claim 2. for S. ◀

Universality Problem for Unambiguous VASS

Wojciech Czerwiński
University of Warsaw, Poland
wczerwin@mimuw.edu.pl

Diego Figueira
Université Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, Talence, France
diego.figueira@labri.fr

Piotr Hofman
University of Warsaw, Poland
piotrek.hofman@gmail.com

Abstract

We study languages of unambiguous VASS, that is, Vector Addition Systems with States, whose transitions read letters from a finite alphabet, and whose acceptance condition is defined by a set of final states (*i.e.*, the coverability language). We show that the problem of universality for unambiguous VASS is ExpSpace-complete, in sheer contrast to Ackermann-completeness for arbitrary VASS, even in dimension 1. When the dimension $d \in \mathbb{N}$ is fixed, the universality problem is PSpace-complete if $d \geq 2$, and coNP-hard for 1-dimensional VASSes (also known as *One Counter Nets*).

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases unambiguity, vector addition systems, universality problems

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.36

Funding *Wojciech Czerwiński*: Supported by the ERC grant LIPA, agreement no. 683080.
Diego Figueira: ANR BraVAS (ANR-17-CE40-0028); ANR DéLTA (ANR-16-CE40-0007).

Acknowledgements We thank Lorenzo Clemente for leading us to the NC^2 membership for UFA universality problem.

1 Introduction

Determinism is a central notion of computational models, it ensures that there is one way to proceed for every input. It often enables constructions which would not be possible without it and allows for efficient algorithms. While the relation between deterministic vs non-deterministic models is extensively studied, there exists also a less understood middle ground of *unambiguous* systems. In the case of models accepting word languages, a model is said to be *unambiguous* if for every word in its language, there is exactly one accepting run, which is a much weaker restriction than determinism. Unambiguity, although featuring non-determinism, often causes some problems to be computationally easier. As a prominent example, the universality problem for finite automata (*i.e.*, whether all words over the alphabet are accepted by the automaton), which is PSpace-complete in general, is known to be in PTime in the unambiguous case [14] and even in NC^2 [15]. While the study of unambiguous models of computation has lately attracted some attention, in some settings it remains, by and large, an unexplored area.

In particular, there has been considerable volume of research on unambiguous *finite automata* (see [1] for a nice overview). One way to design a polynomial time algorithm for the universality problem on finite automata is to show that the shortest word which is not in the language, if any, is of at most linear length. Then, by counting the number of linear length runs one may answer the problem. The existence of a linear counterexample for universality

and its PTime algorithm, led to the conjecture, formulated by Colcombet [1], that for every unambiguous finite automaton (UFA) there exists another UFA of polynomial size accepting the complement of its language. This conjecture was later shown false by Raskin [12]. As it turns out, there is a family of UFA such that for accepting the complement of UFA with n states even nondeterministic finite automaton (NFA) needs a super-polynomial number of states – at least $\Theta(n^{\log\log\log n})$. The universality problem for UFA is actually known to be not only in PTime, but even in NC^2 [15], the class of problems solvable by uniform families of circuits with $\mathcal{O}(\log^2 n)$ depth and binary fan-in. The work [15] in fact solves the more general problem of *path equivalence* for two NFA: is the number of accepting runs on w the same for both automata, for every word w? However, to the best of our knowledge the best known lower bound for the problem is NL-hardness, so the exact complexity of universality problem for UFA is still open even in the simplest possible setting of finite automata.

There was also research about the universality problem and related ones for unambiguous register automata. In [9] authors have shown that the containment problem for unambiguous register automata is in 2ExpSpace and even in ExpSpace if the number of registers is fixed, which implies similar upper bounds for the universality problem. Without the unambiguity assumption, even the universality problem (and even with just one register) can be shown undecidable [10] or Ackermann-hard [3] depending on the concrete model of register automata.

It is not by accident that existing research focuses on universality, equivalence and containment of languages of unambiguous systems, and that there are efficient algorithms for these problems under the assumption of unambiguity. Unambiguity speaks about the language of a system, so it is natural to hope that problems related to the language of the systems may become more tractable. But for the most natural problem concerning the language, i.e., for the emptiness problem one cannot hope for improvement. This is because for most of the systems one can relabel transitions giving each one a unique label. Then the system becomes deterministic and in consequence unambiguous. The language changes, but it is empty iff the original language was empty, which intuitively explains why the emptiness problem shouldn't be any easier for unambiguous systems compared to general non-deterministic ones. On the other hand, it is more reasonable to expect that the universality problem might be easier since both the universality problem and the unambiguity property are universal properties of the form "*For all words, [...]*".

Our contribution

The foremost goal of this paper is to push the understanding of unambiguity further. We focus on the universality problem, which is arguably the most natural first step, that may open the way for further studies on the equivalence, co-finiteness, containment and other problems for languages. The universality problem was studied for finite automata and register automata under the unambiguity assumption. In our opinion, the most interesting yet unsolved cases in which one can expect some progress assuming unambiguity are One Counter Nets (called also 1-dimensional VASS here) and its generalization Vector Addition Systems with States (VASS).

The universality checking for VASS with state acceptance is known to be decidable by the use of well quasi-order techniques [6] (the paper shows decidability of trace universality, but language universality can be reduced to that problem). However the problem is also known to be Ackermann-complete even for 1-dimensional VASS [5], so hardly tractable. For deterministic VASS it is quite easy to show that the universality problem can be decided in PTime. Therefore, it is natural to hope for improvement under the unambiguity restriction.

Our main contribution is ExpSpace membership of the universality problem for unambiguous VASS. We believe that it is the most interesting result and it was as well the most challenging problem and technically involved solution. We actually have shown that this problem is ExpSpace-complete. For the completeness of the picture we have also analyzed the complexity of the problem for d-dimensional VASS for fixed $d \in \mathbb{N}$. We have shown that the problem is PSpace-complete for every $d \geq 2$. For $d = 1$ we have shown coNP-hardness, although we do not have the matching upper bound, we conjecture that it is coNP-complete. We additionally consider the variant of the problem in which the numbers in the input are encoded in unary. Finally, we study also the problem of unambiguity checking (*i.e.*, given a VASS, is it unambiguous?). All our results are listed in Section 3.

2 Preliminaries

We use the letter Σ to denote a finite alphabet, $\mathbb{Z}$ to denote the set of all integers, and $\mathbb{N}$ the set of non-negative integers. We use ε to denote the empty string, and Σ_ε to denote $\Sigma \cup \{\varepsilon\}$. We use $A \subseteq_{fin} B$ to denote that A is a finite subset of B, and $\wp_{fin}(A)$ to denote the set of all finite subsets of A. We use $\bar{u}, \bar{v}, \bar{w}, \ldots$ to denote vectors of numbers, and we use $\bar{0}$ to denote the all-0 vector and $\bar{1}$ to denote the all-1 vector. We use $[i, j]$ for $i, j \in \mathbb{N}$, $i \leq j$ to denote the set $\{i, i+1, \ldots, j-1, j\}$. For a vector $\bar{u} \in \mathbb{Z}^d$ and $i \in [1, d]$ we denote by $\bar{u}[i]$ the i-th coordinate of $\bar{u}$. For a word $w \in \Sigma^*$ and $i \in \mathbb{N}$, $i > 0$ we denote by $w[i]$ the i-th letter of w. For $\bar{u}, \bar{v} \in \mathbb{Z}^d$ we write $\bar{u} \preceq \bar{v}$ if for all $i \in [1, d]$ we have $\bar{u}[i] \leq \bar{v}[i]$. We define the minimum of $\bar{u}$ and $\bar{v}$ as $\min(\bar{u}, \bar{v})[i] = \min(\bar{u}[i], \bar{v}[i])$ for any $i \in [1, d]$.

We consider a Vector Addition Systems with States (VASS) of dimension $d \in \mathbb{N}$ as a tuple $\mathcal{A} = (\Sigma, d, Q, q_0, \delta, F)$ where Σ is a finite alphabet, Q is a finite state space, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta \subseteq_{fin} Q \times \Sigma_\varepsilon \times \mathbb{Z}^d \times Q$ is the set of transitions. We often write transition (p, a, v, q) as $p \xrightarrow{a;v} q$. We will henceforth write *d-VASS* to denote a VASS of fixed dimension d. A *configuration* of $\mathcal{A}$ is a pair of a state $q \in Q$ and a vector $\bar{u} \in \mathbb{N}^d$, that we usually note $q(\bar{u})$. If c is a configuration, we write $c[i]$ to denote the i-th coordinate of the vector it contains. A *run* of $\mathcal{A}$ from a configuration $q(\bar{u})$ to a configuration $q'(\bar{v})$ reading the word $w \in \Sigma^*$ is a sequence of transitions $(r_1, \alpha_1, \bar{v}_1, r'_1) \cdots (r_n, \alpha_n, \bar{v}_n, r'_n) \in \delta^*$ such that: (i) $r_1 = q$ and $r'_n = q'$, (ii) $r'_i = r_{i+1}$ for every $1 \leq i < n$; (iii) $w = \alpha_1 \cdots \alpha_n$; (iv) $\bar{u} + \sum_{i \leq j} \bar{v}_i \in \mathbb{N}^k$ for every $1 \leq j \leq n$; and (v) $\bar{v} = \bar{u} + \sum_{i \leq n} \bar{v}_i$. If we further have $q' \in F$, we say that such run is *accepting*. We henceforth say that a configuration c is *reachable* from a configuration c' if there is a run from c' to c. The *effect* of a transition $(r, \alpha, \bar{v}, r')$ is the vector $\bar{v} \in \mathbb{Z}^d$, the *effect of a run* is the sum of effects of the transitions therein. The *norm* of a VASS $\mathcal{A}$ is the maximal absolute value of a number occurring in its transition, and we denote it by $|\mathcal{A}|$. The *language* of a configuration c in $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A}, c)$, is the set of all $w \in \Sigma^*$ with an accepting run from c. We call $q_0(\bar{0})$ the *initial configuration* where q_0 is the initial state. If c is the initial configuration then we just say language of $\mathcal{A}$ and write $\mathcal{L}(\mathcal{A})$ instead of $\mathcal{L}(\mathcal{A}, c)$. A VASS $\mathcal{A}$ is *unambiguous* if for every $w \in \Sigma^*$ there is no more than one accepting run starting from the initial configuration and reading w. The *unambiguity checking problem* for VASS is the problem of, given a VASS $\mathcal{A}$, decide whether it is unambiguous. An automaton over Σ (finite automaton or VASS) is *universal* if it accepts the language Σ^*. The *universality problem* for VASS is the problem of, given a VASS $\mathcal{A}$, decide whether it is universal. We will henceforth assume that the numbers contained in the transitions of VASSes are always encoded in binary if not explicitly indicated otherwise.

Observe that we work with VASS with ε-transitions, the reason for doing so is that it is a natural model, the upper bounds still hold in this more general setup, and we can also derive tight lower bounds by making use of ε-transitions. We do not know whether adding ε-transitions increases the class of recognized languages, not even in the non-deterministic case. It seems to us a rather difficult question.

Let us recall now the main result of the Rackoff construction [11]. Let us denote $A_{M,d,n} = (2n^2(M+1)^2)^{(4d)^{d-1}}$. We present here an adaptation of the Rackoff argument with an explicit bound on the length of an accepting run.

▶ **Proposition 1** (Adaptation of the Rackoff construction). *If a language of a d-VASS with norm M and n states is nonempty then there exists an accepting run of length at most $A_{M,d,n}$.*

Proof. Let $C = 2n^2(M+1)^2$. We proceed by induction on d. For $d = 1$ assume there is some accepting run with no configuration repeating. Then in its prefix of length nM there is definitely first a configuration $q(x)$ and later a configuration $q(\bar{y})$ for some state q and counter values $x < y$. Then we can change this accepting run into an accepting run of length at most $nM + (nM)^2 + n - 1$. We first pump the infix from $q(x)$ do $q(y)$ exactly nM times obtaining then a configuration $q(z)$ with $z = y + nM(y-x) \geq nM$. As some accepting state is reachable from q then it is also reachable by a run of length smaller than n. This run (and any of its prefixes) can, at worst, have a negative effect of value $(n-1)M$, and thus it can be triggered from $q(z)$, since $z \geq nM$. In this way, we get an accepting run of length at most $nM + (nM)^2 + n - 1 \leq C$, proving the base case.

For the inductive step, assume that there is an accepting run $s(\bar{0}) \xrightarrow{\rho} f(\bar{v})$ in a $(d+1)$-VASS with norm M and n states. Let $K_d = C^{(4d)^{d-1}}$. We distinguish two cases:

(i) the norm of every configuration on ρ is bounded by $C \cdot K_d$;

(ii) the norm of some configuration on ρ exceeds $C \cdot K_d$.

Without loss of generality we can assume that no configuration on ρ appears more than once, otherwise we can "unpump" ρ to obtain a shorter one. Observe that in the first case (i), the length of ρ is bounded by $D = (C \cdot K_d)^{d+1}$ (we will bound D later on).

In the second case (ii), the run ρ might be long, but we will show that there is another short accepting run ρ'. Let $p(\bar{u})$ be the first configuration on ρ with norm exceeding $C \cdot K_d$. Let $s(\bar{0}) \xrightarrow{\rho_1} p(\bar{u}) \xrightarrow{\rho_2} f(\bar{v})$. Clearly, the length of ρ_1 is bounded by D by a similar reasoning as in the case (i). We will replace ρ_2 with a "short" run π, so that $c \xrightarrow{\pi} f(\bar{v}')$. First note that some coordinate of $p(\bar{u})$ must have value greater or equal to $C \cdot K_d$; without loss of generality, assume it is the last one, that is, the $(d+1)$-st coordinate. Let us now ignore the last coordinate in the VASS. By inductive hypothesis, there is a sequence of transitions π of length at most K_d such that $p(\bar{u}_d) \xrightarrow{\pi_d} f(\bar{v}'_d)$, where π_d is the result of ignoring the last coordinate of π, and $\bar{u}_d, \bar{v}_d \in \mathbb{N}^d$ are the results of ignoring the last coordinate of $\bar{u}, \bar{v}$. Consider now the sequence of transitions π starting in $p(\bar{u})$. Its length is bounded by K_d, so its effect on the $(d+1)$-st coordinate is not smaller than $-M \cdot K_d$. Since $\bar{u}[d+1] \geq C \cdot K_d \geq M \cdot K_d$, then π is indeed a valid run from $p(\bar{u})$ to $f(\bar{v}')$ for some $\bar{v}' \in \mathbb{N}^{d+1}$. Therefore, the run $\rho_1 \cdot \pi$ is accepting from $s(\bar{0})$ as $s(\bar{0}) \xrightarrow{\rho_1} p(\bar{u}) \xrightarrow{\pi} f(\bar{v}')$. The length of $\rho_1 \cdot \pi$ is at most $D + K_d$.

In order to finish the argument in case (ii) we need to show that $D + K_d \leq K_{d+1}$, through the following sequence of (very rough) estimations

$$\begin{aligned} D + K_d \leq 2D \leq C \cdot D = C \cdot (C \cdot K_d)^{d+1} = C \cdot \big((C \cdot C^{(4d)^{d-1}}\big)^{d+1} \\ = C^{((4d)^{d-1}+1)(d+1)+1} \leq C^{4(4d)^{d-1} \cdot (d+1)} \leq C^{(4(d+1))^d} = K_{d+1}. \end{aligned}$$

Observe that in case (i), the bound $D \leq K_{d+1}$ is trivial. ◀

The language emptiness problem for VASS (*i.e.*, given a VASS, does it accept at least one word?) is, basically, equivalent to the coverability problem, which is known to be ExpSpace-complete as shown by the lower bound of Lipton [8] and the upper-bound of Rackoff [11]. The coverability problem is the problem of, given a VASS $\mathcal{A}$ and two configurations c_1, c_2, whether there is a run from c_1 to some configuration c_2' such that $c_2' \succeq c_2$. In our setting, this result can be restated as the language emptiness problem for VASS being ExpSpace-complete, even when all transitions are ε-transitions, and hence the language is either $\emptyset$ or $\{\varepsilon\}$. What is more, the construction of Lipton is unambiguous: if there is an accepting run, there is exactly one. Indeed, the only situation in which Lipton's construction is ambiguous along a run is when it guesses whether the value of some counter is zero or non-zero. However, the run of a wrong guess is never an accepting one, as the guess is always followed by a verification. This is formalized in the next lemma. Let us denote by ε-VASS, a VASS whose every transition reads ε (and thus the alphabet is not important here).

▶ **Lemma 2** (consequence of [8, 11]). *The problem of whether an unambiguous ε-VASS has an empty language is ExpSpace-complete.*

3 Results

We summarize all our results in the next two theorems. Detailed proofs will come in the sections that follow.

▶ **Theorem 3.** *The universality problem for*
- **(i)** *VASS is ExpSpace-complete, both with binary and unary encodings;*
- **(ii)** *d-VASS with unary encoding is in NC^2 and NL-hard, for every $d \geq 1$;*
- **(iii)** *d-VASS with binary encoding is PSpace-complete, for every $d \geq 2$;*
- **(iv)** *1-VASS (One Counter Net) with binary encoding is coNP-hard.*

▶ **Theorem 4.** *The unambiguity checking problem for*
- **(i)** *VASS is ExpSpace-complete, both with binary and unary encodings;*
- **(ii)** *d-VASS with unary encoding is NL-complete, for every $d \geq 1$;*
- **(iii)** *d-VASS with binary encoding is PSpace-complete, for every $d \geq 2$;*
- **(iv)** *1-VASS with binary encoding is coNP-hard.*

The main technical contribution lies in the ExpSpace bounds on the universality problem in Theorem 3(i). The upper bound will need some insights on the structure of accepting runs in unambiguous VASS which happen to have a universal language. The remaining upper bounds will follow easily from this one. The PSpace, and coNP lower bounds of items (iii), and (iv) are also of interest, as they reveal different ways in which unambiguity can encode non-trivial properties. The ExpSpace lower bound of item (i) follows easily from Lemma 2. All the remaining results of Theorems 3 and 4 are either easy, or follow from simple adaptations of the three results just mentioned.

It is interesting to observe that complexity results on universality seem to coincide with the complexity of emptiness for the non-deterministic version of the considered classes. Notice also that closing the "gap" between NC^2 and NL in Theorem 3(ii) would imply in particular solving the corresponding problem for UFA, which is an open question.

We observe that, as a corollary, we obtain procedures for testing the equivalence problem between an unambiguous VASS and a regular language. Indeed, the language of an unambiguous VASS $\mathcal{A}$ is equal to a regular language L if, and only if, the VASS $\mathcal{B}$ resulting from the union of $\mathcal{A}$ and the DFA corresponding to the complement of L is unambiguous and universal.

Organization

We will prove Theorem 3 in Section 4 and Theorem 4 in Section 5. Each of these sections is divided into an "upper bounds" and "lower bounds" subsections. For reference, the upper and lower bounds of item (i) of Theorem 3 are shown in Propositions 5 and 17 respectively; item (ii) in Propositions 16 and 21; item (iii) in Propositions 15 and 19; and item (iv) in Proposition 20. The upper and lower bounds of item (i) of Theorem 4 are shown in Propositions 22 and 23 respectively; item (ii) in Propositions 22 and 24; item (iii) in Propositions 22 and 25; and item (iv) in Proposition 26.

4 Testing for Universality

In this section we will prove Theorem 3. Most of the section will be dedicated to proving the ExpSpace upper bound of item (i).

4.1 Upper bounds

▶ **Proposition 5** (Theorem 3(i) upper bound). *The universality problem for unambiguous VASSes is in ExpSpace.*

The proof strategy is as follows. First, we define an abstraction of a configuration, called an N-profile, for $N \in \mathbb{N}$, which is the result of replacing every number bigger than or equal to N with N in a configuration. The intuition is that any number bigger or equal N is so big that we can disregard its exact value. We next show that in certain circumstances, for any unambiguous d-VASS V with n states two configurations having equal $f(|V|, d, n)$-profile have also the same language, where f is some fixed doubly-exponential function. This fact allows us to construct an unambiguous finite automaton $\mathcal{A}$ of doubly-exponential size, whose every state corresponds to one $f(|V|, d, n)$-profile, and such that $\mathcal{A}$ is universal if, and only if, V is universal. As universality of UFAs is in NC^2 and therefore in PolyLogSpace, this gives us an ExpSpace algorithm for checking universality.

For any number $N \in \mathbb{N}$, the *N-profile* of a configuration $(q, \bar{v}) \in Q \times \mathbb{N}^d$ is the pair $(q, \min(\bar{v}, N \cdot \bar{1}))$. Let $B_{M,d,n} = M \cdot A_{M,2d,2n^2}$, and let $C_{M,d,n} = M \cdot (B_{M,d,n} + 1)^d$.

We start with a useful lemma which bounds the length of runs witnessing ambiguity.

▶ **Lemma 6.** *Let V be a d-VASS with norm M and n states. If V is ambiguous then there exist two different runs accepting the same word of length at most $A_{M,2d,2n^2}$ each.*

Proof. Consider the following $2d$-VASS V', which accepts exactly these words, which have at least two different accepting runs from the initial configuration of V. The VASS V' guesses two different runs of V and simulates them, it is quite similar to a synchronized product of V with itself. In its $2d$ counters V' keeps counter valuations of two configurations of V of the simulated runs. State of V' is a pair of states of V together with one bit of information indicating whether the two simulated runs have already differed or they are the same till that moment. VASS V' accepts if states of both simulated runs are accepting and the bit indicates that they have differed (even if now they are in the same state). It is easy to see that V' indeed accepts words, which have two different accepting runs in V. Therefore if V is ambiguous then $\mathcal{L}(V')$ is nonempty. Notice that the norm of V' is bounded by M, as the norm of V is. Therefore by Proposition 1 if $\mathcal{L}(V')$ is nonempty then there is an accepting run of V' of length at most $A_{M,2d,2n^2}$. Notice that the existence of such a run implies the existence of two different runs of V over the same word, which additionally also have length bounded by $A_{M,2d,2n^2}$. This finishes the proof. ◀

We state two basic properties of VASS which will be useful throughout.

▷ Claim 7. For any two configurations c and c' of a VASS V with equal $(|V| \cdot N)$-profile, if ρ is an accepting run from c of length at most N then ρ is also accepting from c'.

▷ Claim 8 (language monotonicity). If $q(\bar{u})$ and $q(\bar{v})$ are two configurations of a VASS V with $\bar{u} \preceq \bar{v}$ then $\mathcal{L}(V, q(\bar{u})) \subseteq \mathcal{L}(V, q(\bar{v}))$.

The following is the key lemma which will enable the improved complexity for the universality problem.

▶ **Lemma 9.** *Let V be a universal, unambiguous d-VASS with n states. Then, any two configurations with equal $B_{|V|,d,n}$-profile reachable from the initial configuration have the same set of accepting runs (in particular, they have the same language).*

Proof. By means of contradiction, let $c_1, c_2 \in Q \times \mathbb{N}^d$ be two configurations reachable from the initial configuration c_{init} with the same $B_{|V|,d,n}$-profile, but different sets of accepting runs. Let ρ be an accepting run from c_1 but not from c_2, reading the word w.

Let $c_{\text{init}} \xrightarrow{u} c_2$. The word uw is accepted by V since it is universal, so there must be a configuration c_2' such that $c_{\text{init}} \xrightarrow{u} c_2'$ and $w \in \mathcal{L}(V, c_2')$. Therefore w is accepted both from configuration c_1 with the run ρ and from configuration c_2' with some accepting run $\hat{\rho}$. There are two cases to consider: either (i) $c_2 \neq c_2'$, or (ii) $c_2 = c_2'$ and $\hat{\rho} \neq \rho$.

For (i), let us first consider an (ambiguous) VASS $\tilde{V}$, being the result of adding ε-labelled self-loops with effect $\bar{0}$ in every state to V. Clearly, for every configuration c we have $\mathcal{L}(V, c) = \mathcal{L}(\tilde{V}, c)$. Let us confider a $2d$-VASS V', which is a synchronized product of $\tilde{V}$ with itself: transitions, initial and accepting states are defined in a natural way. Product is synchronized, so for any $a \in \Sigma_\varepsilon$ there is an a-labelled transition in the product V' iff there exist a-labelled transitions in the two components, both identical with $\tilde{V}$. For two configurations $c = q(\bar{u})$ and $c' = q'(\bar{u}')$ of V we denote by $\mathcal{L}(V', c, c')$ the language $\mathcal{L}(V', (q, q')(\bar{u}, \bar{u}'))$. Notice that, by construction, $\mathcal{L}(V', c, c')$ is the intersection of $\mathcal{L}(V, c)$ and $\mathcal{L}(V, c')$. Therefore the word w belongs to $\mathcal{L}(V', c_1, c_2')$. By Proposition 1 there exists an accepting run ρ' of V' of length at most $A_{|V|,2d,n^2}$ reading a word w' from $\mathcal{L}(V', c_1, c_2') = \mathcal{L}(V, c_1) \cap \mathcal{L}(V, c_2')$. Consider the projection ρ_1 of ρ' onto the first copy of $\tilde{V}$. We know thus that ρ_1 is accepting from c_1. Further, the absolute value of the effect of ρ_1 on every coordinate is at most $|V| \cdot A_{|V|,2d,n^2} \leq |V| \cdot A_{|V|,2d,2n^2} = B_{|V|,d,n}$. Recall that c_1 and c_2 have the same $B_{|V|,d,n}$-profile, so by Claim 7 if ρ_1 is accepting from c_1 then it is also accepting from c_2. Therefore $w' \in \mathcal{L}(V, c_2)$ and $w' \in \mathcal{L}(V, c_2')$, which means that there are two distinct accepting runs over uw' in V, contradicting the fact that it is unambiguous.

For (ii), we have that there are two distinct accepting runs for w from $\max(c_1, c_2)$, namely ρ and $\hat{\rho}$. Then, by Lemma 6, there exist two different runs ρ_1 and ρ_2 from $\max(c_1, c_2)$ of length at most $A_{|V|,2d,2n^2}$ accepting the same word w'. Since c_1 and c_2 have the same $B_{|V|,d,n}$-profile, where $B_{|V|,d,n} = |V| \cdot A_{|V|,2d,2n^2}$, by Claim 7 both ρ_1 and ρ_2 are accepting from configuration c_2, and thus there are two distinct accepting runs over uw' in V, contradicting the fact that it is unambiguous. ◀

▶ **Corollary 10.** *If a universal, unambiguous d-VASS V with n states contains an accepting run with two configurations c_1 and c_2 such that c_1 occurs before c_2, then*

- **(i)** *if c_1 and c_2 have equal $B_{|V|,d,n}$-profile, then $c_1 \preceq c_2$;*
- **(ii)** *for every $i \in [1, d]$, $c_1[i] - c_2[i] < C_{|V|,d,n}$.*

Proof.

(i) By means of contradiction, let c_1 and c_2 be configurations with the same profile such that $c_1 \not\preceq c_2$, meaning that $c_1[i] > c_2[i]$ for some i. Let $\rho_1\rho_2\rho_3$ be an accepting run of V, such that ρ_1 reaches the configuration c_1 from the initial configuration, and ρ_2 reaches the configuration c_2 from configuration c_1. Since the effect of ρ_2 decrements component i, it is easy to see that there is some $k \in \mathbb{N}$ such that $(\rho_2)^k\rho_3$ is an accepting run from c_1 but not from c_2, contradicting Lemma 9 above.

(ii) Suppose there is a decrement of at least $C_{|V|,d,n}$ at some coordinate i. Since $C_{|V|,d,n} = |V| \cdot (B_{|V|,d,n}+1)^d$ is at least the number of $B_{|V|,d,n}$-profiles times the biggest effect of a transition, this means that at least $k = B_{|V|,d,n}$ distinct configurations $c'_1, \ldots, c'_k$ occur in the run between c_1 and c_2 such that $c_1[i] > c'_1[i] > c'_2[i] > \cdots > c'_k[i]$. Hence, among $c_1, c'_1, \ldots, c'_k$ there must be two equal $B_{|V|,d,n}$-profile configurations, contradicting the item (i) above. ◀

This last statement can be informally understood as follows: if V is universal, then it is still universal if configurations are abstracted by their $C_{|V|,d,n}$-profiles. We now formalize what this means. Let us fix an unambiguous VASS V, and let us henceforth write ω as short for $C_{|V|,d,n}$. For any configuration c let $\lfloor c \rfloor$ denote its ω-profile, that is, $\lfloor q(\bar{u}) \rfloor = q(\min(\bar{u}, \omega \cdot \bar{1}))$. Let $V = (\Sigma, d, Q_V, q_V, \delta_V, F_V)$ be an unambiguous VASS. We construct a finite automaton $\mathcal{A}_V = (\Sigma, Q_\mathcal{A}, q_\mathcal{A}, \delta_A, F_\mathcal{A})$ in the following way:

- the set of states $Q_\mathcal{A}$ is the set of pairs $Q_V \times [0,\omega]^d$;
- the initial state $q_\mathcal{A}$ is $q_V(\bar{0})$;
- the set of final states $F_\mathcal{A}$ consists of all the pairs having the first coordinate in F_V, namely $F_\mathcal{A} = F_V \times [0,\omega]^d$;
- $\delta_\mathcal{A}$ is the set of all transitions $p(\bar{u}) \xrightarrow{a} q(\lfloor \bar{u}+\bar{v} \rfloor)$ such that $(p,a,\bar{v},q) \in \delta_V$ and $\bar{u}+\bar{v} \in \mathbb{N}^d$.

We now show that $\mathcal{A}_V$ is unambiguous, and that it is universal iff V is universal.

▶ **Lemma 11.** *For every run $p_1(\bar{u}_1) \xrightarrow{a_1} p_2(\bar{u}_2) \xrightarrow{a_2} \cdots p_n(\bar{u}_n) \xrightarrow{a_n} p_{n+1}(\bar{u}_{n+1})$ of $\mathcal{A}_V$ there is a run $(p_1, a_1, \bar{v}_1, p_2) \cdots (p_n, a_n, \bar{v}_n, p_{n+1})$ of V such that $\bar{v}_1 + \cdots + \bar{v}_i \geq \bar{u}_i$ for every $i \in [1,n]$.*

Proof. This can by shown by induction on n. It suffices to replace every transition $p_i(\bar{u}_i) \xrightarrow{a_i} p_{i+1}(\bar{u}_{i+1})$ of $\mathcal{A}_V$ by a transition $(p_i, a_i, \bar{v}, p_{i+1}) \in \delta_V$ such that $\bar{u}_{i+1} = \lfloor \bar{u}_i + \bar{v} \rfloor$, which exists by construction. ◀

As a consequence of the previous lemma, if there are two distinct accepting runs for a word w in $\mathcal{A}_V$, then there are also two distinct accepting runs over w in V. In other words:

▶ **Lemma 12.** *If V is unambiguous then $\mathcal{A}_V$ is unambiguous.*

▶ **Lemma 13.** *V is universal if, and only if, $\mathcal{A}_V$ is universal.*

Proof. Observe first that $\mathcal{L}(\mathcal{A}_V) \subseteq \mathcal{L}(V)$ by Lemma 11. Hence, if $\mathcal{A}_V$ is universal, so is V. For the converse direction, suppose V is universal, and let us show that $\mathcal{A}_V$ is universal as well. Let $\rho = (q_0, a_1, \bar{v}_1, q_1) \cdots (q_{n-1}, a_n, \bar{v}_n, q_n)$ be the accepting run of $w = a_1 \cdots a_n$ in V. Let us consider the run $\rho' = (q_0(\bar{x}_0), a_1, q_1(\bar{x}_1)) \cdots (q_{n-1}(\bar{x}_{n-1}), a_n, q_n(\bar{x}_n))$ of $\mathcal{A}_V$, where $\bar{x}_0 = \bar{0}$ and for every $i > 0$, $\bar{x}_i = \lfloor \bar{x}_{i-1} + \bar{v}_i \rfloor$. We claim that ρ' is an accepting run on $\mathcal{A}_V$. By means of contradiction, if ρ' is not a run, there must be some $q_i(\bar{x}_i) \xrightarrow{a_{i+1}} q_{i+1}(\bar{x}_{i+1})$ which is not a transition of $\mathcal{A}_V$. This can only happen if some configuration on ρ reaches some big counter value at a position j which later decreases by at least ω. More concretely, this means that there are, among the configurations reachable through ρ, two configurations c, c' such that c appears before c' and for some $j \in [1,k]$ we have $c[j] - c'[j] > \omega$. But this would contradict Corollary 10-(ii). Hence, ρ' is an accepting run and thus $\mathcal{A}_V$ is universal. ◀

Notice that the automaton $\mathcal{A}_V$ has a doubly-exponential number of states. As checking its universality is polynomial-time in its size [1], which is doubly exponential, the problem is in 2ExpTime. In order to design an ExpSpace algorithm we need a bit more work. The following lemma together with Lemma 13 finishes the proof of Proposition 5.

▶ **Lemma 14.** *Checking universality of $\mathcal{A}_V$ is in ExpSpace.*

Proof. Notice first that the function $V \mapsto \mathcal{A}_V$ can be easily computed in ExpSpace. Indeed, a state of $\mathcal{A}_V$ is described by a pair consisting of a state from Q_V and a vector $\bar{v} \in [0,\omega]^d$, where $\omega = C_{|V|,d,|Q_V|} = |V| \cdot (|V| \cdot (4|Q_V|^4(|V|+1)^2)^{(8d)^{2d-1}} + 1)^d$ is doubly exponential with respect to the description size of V, and therefore it can be kept in ExpSpace. It is then possible to iterate through all the possible pairs in $(Q_V, [0,\omega]^d)$ in ExpSpace and for every state output the transitions outgoing from this state.

By [15] checking universality of UFA without cycles containing only ε-labelled transitions (ε-cycles) is in NC^2, namely in the class of languages recognizable by uniform families of circuits of depth $\mathcal{O}(\log^2(n))$ and binary branching, where n is the number of inputs. A simple procedure which eliminates all the ε-cycles (*i.e.*, all the transitions involved in ε-cycles) can be designed to be in NL. Observe that eliminating ε-cycles does not change the language of unambiguous automata, since no accepting run can contain a transition from an ε-cycle (such a run extended by the ε-cycle would be also accepting, which would violate the unambiguity assumption). Since $\mathrm{NL} \subseteq \mathrm{NC}^2$ and NC^2 is closed under composition, we obtain that the universality problem for an arbitrary UFA (possibly with ε-transitions) is in NC^2 as well. It is folklore that NC^2 is included in poly-logarithmic space (actually in the deterministic space $\log^2 n$). Indeed, one can simply simulate a circuit of depth D and binary branching in space D.

It is now enough to argue that the composition of ExpSpace and PolyLogSpace is included in ExpSpace. This result is also folklore, we sketch here a proof. Any algorithm in the composition of ExpSpace and PolyLogSpace can be seen as a PolyLogSpace algorithm inputting the output of an ExpSpace machine, potentially of a doubly exponential length. This doubly exponential output cannot be kept by an ExpSpace algorithm, but one can simulate the composition by a PolyLogSpace algorithm asking ExpSpace oracles for particular letters of its input. Such an algorithm in turn can be simulated easily in ExpSpace. We keep three exponential size pieces of the information: (i) the space of the oracle, (ii) the index of the doubly exponential input being currently transferred to the oracle, and (iii) the space of the poly-logarithmic algorithm, which is poly-logarithmic with respect to the doubly exponential input, hence exponential. Therefore indeed $\mathrm{ExpSpace} \circ \mathrm{PolyLogSpace} \subseteq \mathrm{ExpSpace}$, which finishes the proof. ◀

Let us now analyze the situation for a fixed dimension $d \in \mathbb{N}$. The number of states of $\mathcal{A}_V$ equals $|Q_V|$ times $|V| \cdot (|V| \cdot (4|Q_V|^4(|V|+1)^2)^{(8d)^{2d-1}} + 1)^d$, which for a fixed d is a polynomial depending on $|Q_V|$ and $|V|$. This immediately implies that for V represented in unary the size of $\mathcal{A}_V$ is polynomial, while for $|V|$ represented in binary the size of $\mathcal{A}_V$ is exponential in the size of the input. A proof almost identical to that of Lemma 14, where we substitute ExpSpace with PSpace, yields the following result.

▶ **Proposition 15** (Theorem 3(iii) upper bound)**.** *For every fixed $d \in \mathbb{N}$ the universality problem for binary represented, unambiguous d-VASS is in PSpace.*

In a similar way we solve the case of unary represented d-VASSes. In this case, we replace ExpSpace with the class of problems solvable in logarithmic space L. We also use the fact that L composed with NC^2 is included in NC^2, which is immediately implied by a trivial closure of NC^2 by composition and inclusion $\mathrm{L} \subseteq \mathrm{NC}^2$. Then we get the following.

▶ **Proposition 16** (Theorem 3(ii) upper bound). *For every fixed $d \in \mathbb{N}$ the universality problem for unary represented, unambiguous d-VASS is in NC^2.*

4.2 Lower bounds

▶ **Proposition 17** (Theorem 3(i) lower bound). *The universality problem for unambiguous VASS is ExpSpace-hard, even on a one-letter alphabet.*

Proof. We reduce from the problem of whether an unambiguous ε-VASS has an empty language, which is ExpSpace-hard as observed in Lemma 2. Given an unambiguous ε-VASS $\mathcal{A} = (\{a\}, d, Q, q_0, \delta, F)$, we build an unambiguous VASS $\mathcal{B}$ on a one-letter alphabet $\{a\}$ such that $\mathcal{L}(\mathcal{B}) = a^*$ if $\mathcal{L}(\mathcal{A}) = \{\varepsilon\}$ and $\mathcal{L}(\mathcal{B}) = \emptyset$ otherwise. $\mathcal{B}$ is the result of adding a new final state q_f to $\mathcal{A}$, and transitions $(q, a, \bar{0}, q_f)$ for every $q \in F \cup \{q_f\}$. ◀

▶ **Corollary 18.** *The co-finiteness problem for unambiguous VASS, that is, whether the complement of its language is finite, is ExpSpace-hard.*

We leave open the question of whether the lower bound of Proposition 17 still holds for unambiguous VASS without epsilon transitions.

The following proposition proves the lower bound of Theorem 3(iii).

▶ **Proposition 19** (Theorem 3(iii) lower bound). *The universality problem for unambiguous 2-VASS is PSpace-hard.*

Proof. We reduce from the bounded one-counter automata reachability problem, which is known to be PSpace-hard [4, Corollary 10]. This problem can be stated as follows: given a 1-VASS $\mathcal{A} = (\Sigma, 1, Q_\mathcal{A}, q, \delta_\mathcal{A}, F)$, a number $N \in \mathbb{N}$ encoded in binary, and a configuration $p(m)$, is there a run $(r_1, \alpha_1, u_1, r_1') \cdots (r_n, \alpha_n, u_n, r_n')$ from $q(0)$ to $p(m)$ such that $\sum_{i \leq j} u_i \leq N$ for every $1 \leq j \leq n$? The alphabet is not important for this problem, we can consider that every transition reads the letter a.

Let $\mathcal{A}$, N, $p(m)$ be the input of the aforementioned problem. We now construct, in polynomial time, an unambiguous 2-VASS $\mathcal{B} = (\Sigma, 2, Q, q_0, \delta, F)$, such that it is universal if, and only if, the answer to the input is negative – the statement then follows by closure under complement of PSpace. Concretely, the language of $\mathcal{B}$ is essentially the set of all sequences of transitions in $(\delta_\mathcal{A})^*$ which *do not* contain a run from $q(0)$ to $p(m)$ as a prefix. Intuitively, the construction of $\mathcal{B}$ from $\mathcal{A}$ can be divided into two steps. First we change the N-bounded 1-VASS into a 2-VASS by simulating configuration $q(i)$ by $q(i, N-i)$. However, this 2-VASS might be far from being universal. Therefore, we add to it a lot of transitions such that it is almost universal: the only way for a word not to be accepted is to reach a configuration corresponding to $p(m)$.

The construction of $\mathcal{B}$ is as follows. The alphabet Σ is defined as $\delta_\mathcal{A} \dot{\cup} \{\star\}$; the state set Q is defined as $Q_\mathcal{A} \dot{\cup} \{\bot, q_f, q_0\}$; and the set of final states is $F = Q \setminus \{\bot\}$, where $\bot$ is a *sink state*. $\mathcal{B}$ will always keep the invariant that the sum of its two components is equal to N on all configurations with state in $Q_\mathcal{A}$ reachable from the initial configuration $q_0(0,0)$. Further, the transition graph is as in $\delta_\mathcal{A}$ but labels are used to enforce unambiguity. This is done by initializing the vector in $(0, N)$ as the first thing the automaton does (by adding a new initial state q_0 and transition $(q_0, t, (0, N), q)$ from it to the initial state of $\mathcal{A}$), and additionally translating every transition $t = (r, a, h, r') \in \delta_\mathcal{A}$ into $(r, t, (h, -h), r')$. Now we need to assure that the only way to be not accepted is to reach configuration $p(m, N-m)$. For that purpose we add a special transition reading $\star$ with effect $(-m, m-N)$ and going from p to the sink state $\bot$. All the other sequences of transitions need to be made accepting. For that we add an extra accepting state q_f and a lot of transitions leading to it. Concretely, $\mathcal{B}$ has these transitions:

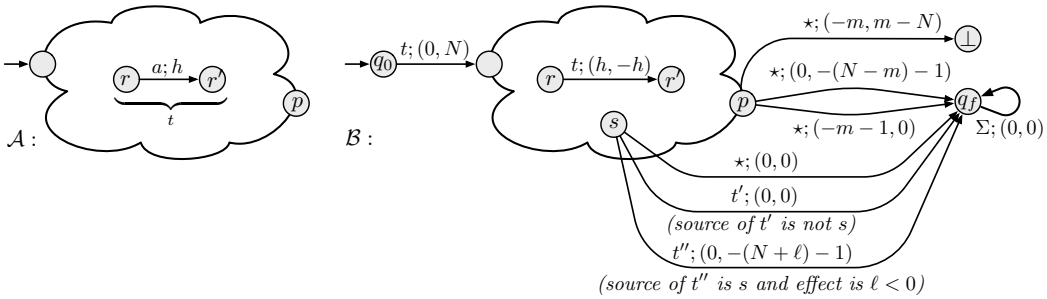


Figure 1 Definition of $\mathcal{B}$. An arrow labelled "$\alpha; \bar{x}$" denotes a transition reading α with effect $\bar{x}$.

- **(i)** the initial transition $(q_0, t, (0, N), q)$ for every $t \in \Sigma$;
- **(ii)** a "simulating" transition $(r, t, (h, -h), r')$ for every $t = (r, a, h, r') \in \delta_{\mathcal{A}}$;
- **(iii)** a transition from p to $\bot$ reading $\star$ with effect $(-m, m - N)$;
- **(iv)** a transition from every $r \in Q_{\mathcal{A}} \setminus \{p\}$ to q_f reading $\star$ with effect $(0, 0)$;
- **(v)** two transitions from p to q_f reading $\star$, one with effect $(-m - 1, 0)$ and one with effect $(0, -(N - m) - 1)$;
- **(vi)** a transition from every $r \in Q_{\mathcal{A}}$ to q_f reading $t \in \Sigma \setminus \{\star\}$ with effect $(0, 0)$ if the t-labelled transition is not outgoing from r;
- **(vii)** a transition from every $r \in Q_{\mathcal{A}}$ to q_f reading $t \in \Sigma \setminus \{\star\}$ with effect $(0, -(N + \ell) - 1)$ if the t-labelled transition is outgoing from r and has effect $\ell < 0$;
- **(viii)** $\bar{0}$-effect self-loops on q_f, with all possible letters of Σ.

Figure 1 contains a depiction of the construction. We now show the correctness of the reduction. Observe first that, by construction, all configurations c of $\mathcal{B}$ reachable from $q_0(0, 0)$ are N-bounded. Further, if the state of c is from $Q_{\mathcal{A}}$, then the sum of its components is equal to N.

We show that $\mathcal{B}$ is unambiguous. What is more, we will show that for every configuration $r(\bar{u}_0)$ reachable from $q_0(0, 0)$ and for every letter $a \in \Sigma$ there is at most one outgoing transition from r reading a that can be applied to $r(u_0, u_0')$. By means of contradiction, suppose that there are two distinct transitions $(r, a, (u_1, u_1'), r_1), (r, a, (u_2, u_2'), r_2) \in \delta$ such that $(u_0, u_0') + (u_1, u_1') \in \mathbb{N}^2$ and $(u_0, u_0') + (u_2, u_2') \in \mathbb{N}^2$. By construction, the only possibility is that one transition is a simulating transition as defined in (ii), and the other transition is as defined in (vii). In particular, a must be a transition from $\delta_{\mathcal{A}}$, r, r_1 are states from $Q_{\mathcal{A}}$, and $r_2 = q_f$. By the above observation, $u_0 + u_0' = N$, and by construction (item (vii)), $u_1 < 0$ and $u_2' = -(N + u_1) - 1$. Since $u_0' + u_2' \geq 0$ by the hypothesis $(u_0, u_0') + (u_2, u_2') \in \mathbb{N}^2$, we can replace u_2' with the equality $u_2' = -(N + u_1) - 1$ just observed, and we obtain $u_0' - (N + u_1) - 1 \geq 0$. Since we also know that $u_0 + u_0' = N$ by the observation above, we can further replace u_0' with $N - u_0$ in $u_0' - (N + u_1) - 1 \geq 0$, and we obtain $u_0 + u_1 < 0$. Note that this contradicts the hypothesis $(u_0, u_0') + (u_1, u_1') \in \mathbb{N}^2$. The contradiction comes from assuming that both transitions were possible to trigger.

We finally show that $\mathcal{B}$ is universal if, and only if, there is no N-bounded run from $q(0)$ to $p(m)$ in $\mathcal{A}$. Observe first that for every word $w \in \Sigma^*$ there is exactly one run of $\mathcal{B}$ reading w. If there is an N-bounded run ρ from $q(0)$ to $p(m)$ in $\mathcal{A}$, it follows that the run of $\mathcal{B}$ reading $\rho\star$ ends in the sink state $\bot$, and thus $\rho\star \notin \mathcal{L}(\mathcal{B})$, witnessing the fact that $\mathcal{B}$ is not universal. If, on the other hand, there is a run of $\mathcal{B}$ ending in state $\bot$, it must be reading a word of the form $\rho\star$ where ρ is an N-bounded run from $q(0)$ to $p(m)$ in $\mathcal{A}$. Since $\bot$ is the sole state which is not accepting and since, as observed before, for all words there is a run, it follows that if $\mathcal{B}$ is universal, then there is no N-bounded run in from $q(0)$ to $p(m)$ in $\mathcal{A}$. ◀

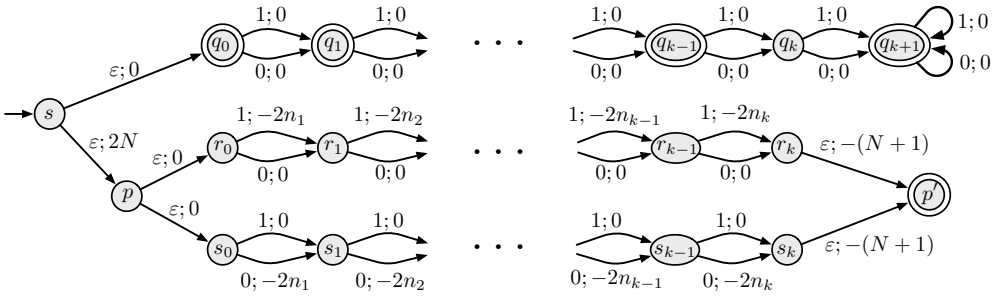


Figure 2 Definition of V_S. An arrow labelled "$i; \ell$" denotes a transition reading i with effect ℓ. Double circled states are final.

Finally, we show coNP-hardness for universality of one counter nets.

▶ **Proposition 20** (Theorem 3(iv)). *The universality problem for unambiguous 1-VASS is coNP-hard.*

Proof. We equivalently will show that non-universality problem for unambiguous 1-VASSes is NP-hard. The reduction is from the Perfect Partition problem. In the Perfect Partition problem we are given a finite set of natural numbers $S = \{n_1, \ldots, n_k\} \subseteq_{\text{FIN}} \mathbb{N}$ and we are supposed to answer whether the set of indices $[1, k]$ can be partitioned into two subsets $I_1, I_2 \subseteq [1, k]$ such that $\sum_{i \in I_1} n_i = \sum_{i \in I_2} n_i$. Such a partition is called a *perfect partition*. All the numbers are binary represented. The Perfect Partition problem is known to be NP-hard [7].

For an instance of a Perfect Partition problem $S \subseteq_{\text{FIN}} \mathbb{N}$ we build an unambiguous 1-VASS V_S such that perfect partition for S exists if and only if the 1-VASS is not universal. Let $\sum_{i \in [1,k]} n_i = N$, note that the perfect partition exists iff there is a set of indices I such that $\sum_{i \in I} n_i = N/2$. Every word of length k encodes a natural number S_w in the following way: for $w \in \{0,1\}^k$ we define $S_w = \sum_{i | w[i]=1} n_i$. We will design V_S in such a way that $\mathcal{L}(V_S) \subseteq \{0,1\}^*$ will always contain all the words of length different than k. Among words of length k language $\mathcal{L}(V_S)$ will contain exactly these for which $S_w \neq N/2$. Then indeed $\mathcal{L}(V_S)$ would be not universal iff set S has a perfect partition.

The 1-VASS V_S is defined in Figure 2. It consists of two parts: the top part, with states q_0 to q_{k+1} accepts all the words of length different than $|S| = k$, while the bottom part accepts some words of length k.

It is immediate to see that the top part accepts all words of length different to k, and all of them by exactly one run.

The bottom part consists of states: p, p', $r_0, r_1, \ldots, r_k$ and $s_0, s_1, \ldots, s_k$, where only the state p' is accepting. Notice that transitions in states r_i are mirrored with respect to transitions in states s_i, namely effect of a transition over some letter from r_i equals the effect of the transition over the other letter in s_i. Let us inspect now how an accepting run over $w \in \{0,1\}^k$ can look like. Every such run starts from $q_0(0)$ and then goes to $p(2N)$. Then it splits into two runs, to $r_0(2N)$ and $s_0(2N)$ and from this moment on there are two runs: one in some state r_i and the other in the corresponding state s_i. Then after reading the whole w the two runs are in configurations $r_k(2N - 2S_w)$ and $s_k(2N - 2(N - S_w)) = s_k(2S_w)$. Notice that $0 \leq S_w \leq N$, so both configurations are indeed always reachable. Now comes the last transition from either r_k or s_k to p'. Observe that if $S_w \neq N/2$ then exactly one of them can be fired. Indeed if $S_w \neq N/2$ so $2S_w \neq N$ then exactly one of the numbers $2S_w$ and

$2N - 2S_w$ equals at least $N + 1$. Then from exactly one of the configurations $r_k(2N - 2S_w)$ and $s_k(2S_w)$ counter value $N + 1$ can be subtracted and the run over the word w will reach an accepting configuration $p'(c)$ for some $c \geq 0$. Then we have $w \in \mathcal{L}(V_S)$ and exactly one accepting run over w. On the other hand assume now that $S_w = N/2$. Then the two reached configurations are $r_k(N)$ and $s_k(N)$. In none of them counter value $N + 1$ can be subtracted, which means that in that case no accepting run over w exists and $w \notin \mathcal{L}(V_S)$. Therefore indeed V_S is unambiguous and importantly $\mathcal{L}(V_S)$ is not universal iff there exists a perfect partition for S. This finishes the proof. ◀

▶ **Proposition 21** (Theorem 3(ii) lower bound). *The universality problem for d-VASS with unary encoding is NL-hard, for every $d \geq 1$.*

Proof. This already holds for UFA. ◀

5 Testing for Unambiguity

Here we will prove Theorem 4. As we will see, upper bounds follow from the emptiness problem and lower bounds from adaptations of the reductions from the previous section.

5.1 Upper bounds

We will next prove the upper bound of Theorem 4(i), (ii) and (iii) namely:

▶ **Proposition 22** (Theorem 4(i), (ii) and (iii) upper bound). *The unambiguity checking problem is:*

(i) *in ExpSpace for VASSes with binary encoding;*
(ii) *in NL for d-VASSes with unary encoding for any fixed $d \in \mathbb{N}$;*
(iii) *in PSpace for d-VASSes with binary encoding for any fixed $d \in \mathbb{N}$.*

Proof. By Lemma 6 if a d-VASS with norm M and n states is ambiguous then there exists two different runs of length at most $A_{M,2d,2n^2}$ accepting the same word. Number $A_{M,2d,2n^2} = (4n^4(M+1)^2)^{(8d)^{2d-1}}$ is doubly exponential wrt. the size of the VASS representation when M is given in binary and d is not fixed. For fixed d an M given in binary $A_{M,2d,2n^2}$ is exponential wrt. the input and for fixed d and M given in unary it is polynomial wrt. the input. Therefore the algorithm, which enumerates all the pairs of different runs of length up to $A_{M,2d,2n^2}$ and checks whether some pair accepts the same word works in ExpSpace, PSpace and NL, respectively, which finishes the proof. ◀

5.2 Lower bounds

▶ **Proposition 23** (Theorem 4(i) lower bound). *The unambiguity checking problem for VASS with unary encoding is ExpSpace-hard.*

Proof. We reduce from the problem of whether an unambiguous ε-VASS has an empty language, which is ExpSpace-complete as mentioned in Lemma 2 (it is a consequence of Lipton's construction [8]). To an unambiguous ε-VASS we add one state accepting the empty word ε. Then the constructed VASS is unambiguous iff the original one has empty language, which finishes the ExpSpace-hardness proof. ◀

▶ **Proposition 24** (Theorem 4(ii) lower bound). *The unambiguity checking problem for d-VASS with unary encoding is NL-hard.*

Proof. This is already true for finite automata. ◀

▶ **Proposition 25** (Theorem 4(iii) lower bound). *The unambiguity checking problem for 2-VASS is* PSPACE*-hard.*

Proof. This is a corollary of the construction in the proof of Proposition 19. One can adapt the automaton by now having $\bot$ as a sole accepting state, and all other states as non-accepting, and adding a transition $(\bot, \varepsilon, (0,0), \bot)$, in such a way that $\mathcal{B}$ is unambiguous if, and only if, there is no run that reaches $\bot$. ◀

▶ **Proposition 26** (Theorem 4(iv)). *The unambiguity checking problem for* 1*-VASS is* CONP*-hard.*

Proof. A construction very similar to the one used to show CONP-hardness of universality (Proposition 20) can be used to show that unambiguity checking for 1-VASS is CONP-hard. If instead of transitions $r_k \xrightarrow{\varepsilon;-(N+1)} p'$ and $s_k \xrightarrow{\varepsilon;-(N+1)} p'$ we have transitions $r_k \xrightarrow{\varepsilon;-N} p'$ and $s_k \xrightarrow{\varepsilon;-N} p'$, then V_S is ambiguous if and only if there is a perfect partition for S. This shows that ambiguity checking is NP-hard and unambiguity checking is CONP-hard. ◀

6 Discussion

We leave open the question about the exact complexity of universality problem for unambiguous 1-VASS with transitions represented in binary, which we showed to be PSPACE-easy and CONP-hard. We conjecture that it is CONP-complete. Another question that we leave open is the complexity of the universality problem for VASS without ε-transitions; our EXPSPACE-hardness of Proposition 17 crucially uses ε-transitions, and it is not clear whether it can be adapted to avoid them. We conjecture that the universality problem for unambiguous VASS without ε-transitions is still EXPSPACE-hard. An open question related to the gap of Theorem 3(ii) is the one about the precise complexity of the universality problem for unambiguous finite automata, which is NL-hard and only known to be in NC^2 [15].

While we have focused our study on the universality and unambiguity checking problems for unambiguous VASS, we point out that there are many intriguing unanswered problems on unambiguous systems. In particular, closely related to the universality problem are: co-finiteness, equivalence and inclusion problems. The universality problem is often strongly connected with the equivalence and inclusion problems. As observed in Section 3, the techniques allow for answering the equivalence problem with a regular language. However, equivalence between two unambiguous VASS seems a more difficult question. In particular, observe that trying to reduce $L(\mathcal{A}) \subseteq L(\mathcal{B})$ to $L((\mathcal{A}) \cap L(\mathcal{B})) \cup \overline{L(\mathcal{B})} = \Sigma^*$ would fail in this case, since VASS and unambiguous VASS are not closed under complement – in fact, the only VASSes whose complement is a VASS are those denoting regular languages [2].

It is natural to ask about the decidability and complexity of these problems for most fundamental models of computation: finite automata, one counter nets, VASS or even pushdown automata (PDA) under the assumption of unambiguity. We give some examples. While equivalence of VASS languages is undecidable, is it decidable for unambiguous VASS? Language equivalence is undecidable for PDA and decidable for deterministic PDA (by the celebrated result of Sénizergues [13]), but might it still be decidable for unambiguous PDA? And what about universality?

References

1. Thomas Colcombet. Unambiguity in automata theory. In *Proceedings of DCFS 2015*, pages 3–18, 2015.
2. Wojciech Czerwiński, Sławomir Lasota, Roland Meyer, Sebastian Muskalla, K. Narayan Kumar, and Prakash Saivasan. Regular separability of well-structured transition systems. In *29th International Conference on Concurrency Theory, CONCUR 2018*, volume 118 of *LIPIcs*, pages 35:1–35:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.35.
3. Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009. doi:10.1145/1507244.1507246.
4. John Fearnley and Marcin Jurdziński. Reachability in two-clock timed automata is PSPACE-complete. *Inf. Comput.*, 243:26–36, 2015.
5. Piotr Hofman and Patrick Totzke. Trace inclusion for one-counter nets revisited. In Joël Ouaknine, Igor Potapov, and James Worrell, editors, *Proceedings of RP 2014*, volume 8762 of *Lecture Notes in Computer Science*, pages 151–162. Springer, 2014.
6. Petr Jančar, Javier Esparza, and Faron Moller. Petri nets and regular processes. *J. Comput. Syst. Sci.*, 59(3):476–503, 1999.
7. Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations 1972*, pages 85–103, 1972.
8. Richard Lipton. The reachability problem requires exponential space. *Department of Computer Science. Yale University*, 62, 1976.
9. Antoine Mottet and Karin Quaas. The containment problem for unambiguous register automata. In *Proceedings of STACS 2019*, pages 53:1–53:15, 2019.
10. Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.
11. Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978.
12. Mikhail Raskin. A superpolynomial lower bound for the size of non-deterministic complement of an unambiguous automaton. In *Proceedings of ICALP 2018*, pages 138:1–138:11, 2018.
13. Géraud Sénizergues. L(A)=L(B)? decidability results from complete formal systems. *Theor. Comput. Sci.*, 251(1-2):1–166, 2001.
14. Richard Edwin Stearns and Harry B. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM J. Comput.*, 14(3):598–611, 1985.
15. Wen-Guey Tzeng. On path equivalence of nondeterministic finite automata. *Inf. Process. Lett.*, 58(1):43–46, 1996.

Reachability in Two-Dimensional Vector Addition Systems with States: One Test Is for Free

Jérôme Leroux
LaBRI, Université Bordeaux, CNRS, Bordeaux-INP, Talence, France
jerome.leroux@labri.fr

Grégoire Sutre
LaBRI, Université Bordeaux, CNRS, Bordeaux-INP, Talence, France
gregoire.sutre@labri.fr

Abstract

Vector addition system with states is an ubiquitous model of computation with extensive applications in computer science. The reachability problem for vector addition systems is central since many other problems reduce to that question. The problem is decidable and it was recently proved that the dimension of the vector addition system is an important parameter of the complexity. In fixed dimensions larger than two, the complexity is not known (with huge complexity gaps). In dimension two, the reachability problem was shown to be PSPACE-complete by Blondin et al. in 2015. We consider an extension of this model, called 2-TVASS, where the first counter can be tested for zero. This model naturally extends the classical model of one counter automata (OCA). We show that reachability is still solvable in polynomial space for 2-TVASS. As in the work Blondin et al., our approach relies on the existence of small reachability certificates obtained by concatenating polynomially many cycles.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Counter machine, Vector addition system, Reachability problem, Formal verification, Infinite-state system

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.37

Related Version A full version of the paper is available at [21], `https://arxiv.org/abs/2007.09096`.

Funding This work was supported by the grant ANR-17-CE40-0028 of the French National Research Agency ANR (project BRAVAS).

1 Introduction

Context. Vector addition systems with states (VASS for short) is an ubiquitous model of computation with extensive applications in computer science. This model, equivalent to Petri nets, is defined as a finite state automaton with transitions acting on a set of counters ranging over the nonnegative integers by adding integers. The number of counters is called the dimension and we write d-VASS for a VASS with d counters. The central problem on VASS is the reachability problem since many other problems are reducible to reachability questions. This problem was first proved to be hard for the exponential-space complexity by Lipton [23] in 1976. At that time, the decidability of the problem was open. Three years later [11], the reachability problem for 2-VASS was proved to be decidable by Hopcroft and Pansiot by observing that reachability sets of 2-VASS are *semilinear*. Dimension two is a special case since in contrast reachability sets of 3-VASS are not semilinear in general. A few years later, the reachability problem for VASS was proved to be decidable in any dimension by Mayr [24, 25] thanks to an algorithm simplified later by Kosaraju [12] and Lambert [13]. Recently, the problem was revisited by Leroux [15, 16, 17] by observing that the reachability problem can be decided with a simple algorithm based on semilinear

inductive invariants. Despite recent improvements on the reachability problem, the exact complexity is still open; the known lower-bound is Tower-hard [4] and the known upper-bound is Ackermannian-easy [18].

When adding to VASS the ability to test counters for zero, the reachability problem becomes undecidable in dimension two via a direct simulation of two-counters Minsky machines [26, Chapter 14]. In dimension one, the class of VASS that we obtain by adding zero-tests are usually called one counter automata (OCA for short). The reachability problem for that class was proved to be NP-complete in [10]. The class of OCA can be naturally extended by introducing the class of d-TVASS (or just TVASS when the dimension d is not fixed) corresponding to a d-VASS extended with zero-tests on the first counter. In that context, a OCA is just a 1-TVASS. The reachability problem is known to be decidable for TVASS in any dimension [28, 2], but the complexity is open, even in dimension two.

In dimension two, the reachability problem for VASS is known to be PSPACE-complete. This result was obtained thanks to a series of results from several authors. The PSPACE lower-bound was proved in [8] and PSPACE membership was obtained as follows (notice that the problem was recently revisited in [5]). First of all, the reachability relation was proved to be semilinear in [19] by observing that it is *flattenable*, meaning that the reachability relation can be captured by a finite set of regular expressions, so called *linear path schemes*, of the form $\alpha_0\beta_1^*\alpha_1\cdots\beta_k^*\alpha_k$ where $\alpha_0,\ldots,\alpha_k$ are paths and $\beta_1,\ldots,\beta_k$ are cycles in the underlying graph of the VASS. It was then proved in [1] that these regular expressions can be exponentially bounded, and k is bounded by a polynomial in the number of states. By introducing a system of inequalities over some variables $n_1,\ldots,n_k$ counting the number of times the cycles $\beta_1,\ldots,\beta_k$ are iterated, an exponential bound on small paths witnessing reachability was derived from a small solution theorem [27, 3]. From such a bound, it follows that the reachability problem is decidable in PSPACE.

Our contribution. In this paper, we are interested in the complexity of the reachability problem for 2-TVASS. We successfully follow the approach used for 2-VASS and outlined above. This approach is not easily lifted to 2-TVASS, because the presence of zero-tests breaks a fundamental property of VASS, namely *monotonicity*. By means of new proof techniques to deal with zero-tests on a single counter, we obtain the following results:

- We show that the reachability relation of a 2-TVASS is flattenable. Our proof does not provide by itself any complexity bound but it is direct and simple, and it provides a description of the reachability relation by *linear path schemes* $\alpha_0\beta_1^*\alpha_1\cdots\beta_k^*\alpha_k$.
- We prove that these linear path schemes can be exponentially bounded, and the number k can be polynomially bounded in the number of states of the 2-TVASS. This bound is obtained via a detour through the class of *weighed one counter automata* (WOCA for short). We believe that our results on WOCA may be of independent interest.
- We derive an exponential bound on paths witnessing reachability thanks to a small solution theorem. From that bound, we deduce that the reachability problem for 2-TVASS is decidable in polynomial space, and so is PSPACE-complete. This is, to our knowledge, one of the few problems on extended VASS whose precise complexity is known.

Related work. TVASS are naturally related to other classical extensions of VASS by observing that a reset is a "weak test", and a testable counter is a "weak stack".

By extending d-VASS with resets on the two first counters, we obtain the class of d-RRVASS. It is known that the reachability problem for this class is undecidable if $d \geq 3$ while it is decidable for $d = 2$ [6]. Since a reset can be simulated by a test, the class of

2-TRVASS obtained from 2-VASS by allowing tests on the first counter and resets on the second one, contains the 2-RRVASS. In [9], we proved that the reachability problem for 2-TRVASS is decidable by proving that the reachability relation is effectively semilinear. It worth noticing that this relation is *not* flattenable, and the complexity of the reachability problem for 2-RRVASS and 2-TRVASS is still open. When dealing with the *lossy semantics* (i.e., when counters can be decreased arbitrarily at any step of the execution), tests and resets have exactly the same behavior. In that case, the reachability problem for lossy Minsky machines of arbitrary dimension becomes decidable and the exact complexity is Ackermannian complete [29].

The class of TVASS is also related to the class of pushdown VASS (PVASS for short) obtained by extending VASS with a stack over a finite alphabet. A PVASS can easily simulate any TVASS since a testable counter can be simulated with a stack. The decidability of the reachability problem is open even for 1-PVASS. We proved in [22] that the control-state reachability problem for 1-PVASS is decidable. The complexity is still open.

Due to space constraints, some proofs are missing and some proofs are only sketched. Detailed proofs can be found in the full version of the paper [21].

2 Flattenability of 2-TVASS

Preliminaries. The usual sets of *integers* and *nonnegative integers* are denoted by $\mathbb{Z}$ and $\mathbb{N}$, respectively. For any $a, b \in \mathbb{Z}$, we let $[a,b] \stackrel{\text{def}}{=} \{z \in \mathbb{Z} \mid a \leq z \leq b\}$. A d-dimensional *vector* of integers is a tuple $\mathbf{v} = (v_1, \ldots, v_d)$ in $\mathbb{Z}^d$. Its ith *component* v_i is also written $\mathbf{v}(i)$. We denote by $\|\mathbf{v}\|$ its *infinity norm* $\max\{|v_1|, \ldots, |v_d|\}$. A *word* over some alphabet Σ is a finite sequence $w = a_1 \cdots a_n$ of elements $a_i \in \Sigma$. The *length* of w is $|w| \stackrel{\text{def}}{=} n$. Given two binary relations R and S over some set, we let $R \fatsemi S \stackrel{\text{def}}{=} \{(x,z) \mid \exists y : x\, R\, y\, S\, z\}$ denote their *relational composition*. The *powers* of a binary relation R are inductively defined by $R^1 \stackrel{\text{def}}{=} R$ and $R^{n+1} \stackrel{\text{def}}{=} R \fatsemi R^n$.

Vector Addition Systems with States and One Test. A TVASS is a vector addition system with states (VASS) such that the first counter can be tested for zero. Formally, a *d-dimensional TVASS* (shortly called a *d-TVASS*), is a triple $\mathcal{V} = (Q, \Sigma, \Delta)$ where Q is a finite nonempty set of *states*, $\Sigma \subseteq \mathbb{Z}^d \cup \{\mathtt{tst}\}$ is a finite set of *actions*, and $\Delta \subseteq Q \times \Sigma \times Q$ is a finite set of *transitions*. Even though they are not mentioned explicitly, $\mathcal{V}$ implicitly comes with d counters $\mathtt{c}_1, \ldots, \mathtt{c}_d$ whose values range over nonnegative integers. Actions in Σ are either *addition* actions $\mathbf{a} \in \mathbb{Z}^d$ or the *zero-test* action $\mathtt{tst}$. Intuitively, an addition action $\mathbf{a} = (a_1, \ldots, a_d)$ performs the instruction $(\mathtt{c}_1, \ldots, \mathtt{c}_d) \leftarrow (\mathtt{c}_1 + a_1, \ldots, \mathtt{c}_d + a_d)$, provided that all counters remain nonnegative ; the zero-test action $\mathtt{tst}$ tests the first counter for zero and leaves all counters unchanged. We let $A \stackrel{\text{def}}{=} \{(p,\sigma,q) \in \Delta \mid \sigma \in \mathbb{Z}^d\}$ and $T \stackrel{\text{def}}{=} \{(p,\sigma,q) \in \Delta \mid \sigma = \mathtt{tst}\}$ denote the sets of *addition* transitions and *zero-test* transitions, respectively. The notation $\|\Sigma\|$ stands for $\max_{\mathbf{a}} \|\mathbf{a}\|$ where $\mathbf{a}$ ranges over addition actions (or $\{\mathbf{0}\}$ if there are none). A *d-dimensional VASS* (shortly called a *d-VASS*) is a d-TVASS whose set of actions Σ excludes $\mathtt{tst}$, i.e., $\Sigma \subseteq \mathbb{Z}^d$.

We define the operational semantics of a d-TVASS $\mathcal{V} = (Q, \Sigma, \Delta)$ as follows. A *configuration* of $\mathcal{V}$ is a pair $(q, \mathbf{x})$ where $q \in Q$ is a state and $\mathbf{x} \in \mathbb{N}^d$ is a vector denoting the contents of the counters $\mathtt{c}_1, \ldots, \mathtt{c}_d$. For the sake of readability, configurations $(q, \mathbf{x})$ are written $q(\mathbf{x})$ in the sequel. For each transition $\delta \in \Delta$, we let $\xrightarrow{\delta}$ denote the least binary relation over configurations satisfying the following rules:

$$\frac{\delta = (p, \mathbf{a}, q) \qquad \mathbf{x} \in \mathbb{N}^d \qquad \mathbf{x} + \mathbf{a} \geq \mathbf{0}}{p(\mathbf{x}) \xrightarrow{\delta} q(\mathbf{x} + \mathbf{a})} \qquad \frac{\delta = (p, \mathtt{tst}, q) \qquad \mathbf{x} \in \mathbb{N}^d \qquad \mathbf{x}(1) = 0}{p(\mathbf{x}) \xrightarrow{\delta} q(\mathbf{x})}$$

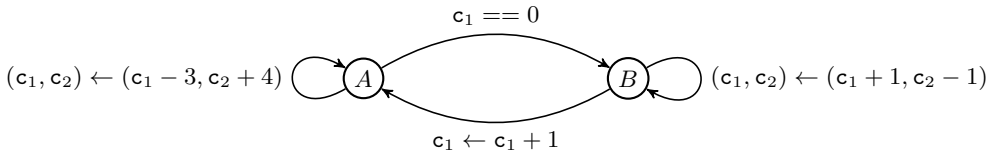


Figure 1 A simple 2-dimensional TVASS with actions written in verbose pseudo-code (to help the reader). Instructions of the form $(\mathtt{c}_1, \mathtt{c}_2) \leftarrow (\mathtt{c}_1 + a_1, \mathtt{c}_2 + a_2)$ stand for addition actions (a_1, a_2). The instruction $\mathtt{c}_1 == 0$ stands for the zero-test action `tst`.

Given a word $\pi = \delta_1 \cdots \delta_n$ of transitions $\delta_i \in \Delta$, we denote by $\xrightarrow{\pi}$ the binary relation over configurations defined as the relational composition $\xrightarrow{\delta_1} \circ \cdots \circ \xrightarrow{\delta_n}$. The relation $\xrightarrow{\varepsilon}$ denotes the identity relation on configurations. Given a subset $L \subseteq \Delta^*$, we let $\xrightarrow{L}$ denote the union $\bigcup_{\pi \in L} \xrightarrow{\pi}$. The relation $\xrightarrow{\Delta^*}$, also written $\xrightarrow{*}$, is called the *reachability relation* of $\mathcal{V}$. Observe that $\xrightarrow{*}$ is the reflexive-transitive closure of the *step* relation $\rightarrow \stackrel{\text{def}}{=} \xrightarrow{\Delta}$.

A *run* is a finite, alternating sequence $(q_0(\mathbf{x}_0), \delta_1, q_1(\mathbf{x}_1), \ldots, \delta_n, q_n(\mathbf{x}_n))$ of configurations and transitions, satisfying $q_{i-1}(\mathbf{x}_{i-1}) \xrightarrow{\delta_i} q_i(\mathbf{x}_i)$ for all $i \in [1, n]$. Note that this condition entails that $q_0(\mathbf{x}_0) \xrightarrow{\delta_1 \cdots \delta_n} q_n(\mathbf{x}_n)$. The word $\delta_1 \cdots \delta_n$ is called the *trace* of the run and n is its *length*.

▶ **Example 1.** Consider the 2-TVASS depicted in Figure 1. There are two states, namely A and B, and four transitions, namely:

$$\begin{array}{lcllcl} \delta_{AA} & = & (A\,,(-3,4)\,,A) & \qquad \delta_{BB} & = & (B\,,(1,-1)\,,B) \\ \delta_{AB} & = & (A\,,\ \mathtt{tst}\ \,,B) & \qquad \delta_{BA} & = & (B\,,\ (1,0)\ \,,A)\,. \end{array}$$

Starting from the configuration $A(3,5)$, the zero-test transition δ_{AB} cannot be taken as the first counter is not zero. But we can take the loop on A and reach the configuration $A(0,9)$, which is formally written as the step $A(3,5) \xrightarrow{\delta_{AA}} A(0,9)$. We may then move to B via the zero-test transition, take the loop on B four times, and get back to A. This yields the run $\rho = (A(3,5), \delta_{AA}, A(0,9), \delta_{AB}, B(0,9), \delta_{BB}, B(1,8), \ldots, \delta_{BB}, B(4,5), \delta_{BA}, A(5,5))$. The trace of this run is $\pi = \delta_{AA}\delta_{AB}(\delta_{BB})^4\delta_{BA}$, and so we have $A(3,5) \xrightarrow{\pi} A(5,5)$.

The run ρ witnesses the fact that $A(3,5) \xrightarrow{*} A(5,5)$. In a standard 2-VASS, i.e., without zero-test, the run ρ could be "replayed" from the larger configuration $A(5,5)$. More precisely, we would have $A(3,5) \xrightarrow{\pi} A(5,5) \xrightarrow{\pi} A(7,5)$. This is not the case in our 2-TVASS. Even though $A(3,5) \xrightarrow{\pi} A(5,5)$, it does not hold that $A(5,5) \xrightarrow{\pi} A(7,5)$. Indeed, $A(5,5) \xrightarrow{\delta_{AA}} A(2,9)$ and the zero-test transition δ_{AB} cannot be taken from $A(2,9)$.

We cannot replay ρ from the larger configuration $A(5,5)$. Nonetheless, the configuration $A(7,5)$ is reachable from $A(3,5)$ in our 2-TVASS. In fact, it holds that $A(3,5) \xrightarrow{*} A(3+2k,5)$ for every $k \in \mathbb{N}$. This property will be shown in Example 2. ⌟

Linear Path Schemes and Flattenability. Consider a d-TVASS $\mathcal{V} = (Q, \Sigma, \Delta)$. A *path* from a state $p \in Q$ to a state $q \in Q$ is either the empty word ε or a nonempty word $\delta_1 \cdots \delta_n$ of transitions, with $\delta_i = (p_i, \sigma_i, q_i)$, such that $p_0 = p$, $q_n = q$ and $q_{i-1} = p_i$ for all $i \in [1, n]$. Note that for every word $\pi \in \Delta^*$, if the relation $\xrightarrow{\pi}$ is not empty then π is a path. The converse does not hold in general (but it holds if $\pi \in A^*$, i.e., if no zero-test occurs in π). A *cycle* on a state $q \in Q$ is a path from q to q.

A *linear path scheme* from a state $p \in Q$ to a state $q \in Q$ is a regular expression L of the form $L = \alpha_0\beta_1^*\alpha_1 \cdots \beta_k^*\alpha_k$ where $\alpha_0\beta_1\alpha_1 \cdots \beta_k\alpha_k$ is a path from p to q and each β_i is a cycle. We call $\beta_1, \ldots, \beta_k$ the *cycles* of L. Its *length* is $|L| \stackrel{\text{def}}{=} |\alpha_0\beta_1\alpha_1 \cdots \beta_k\alpha_k|$ and its $*$-*length* is $|L|_* \stackrel{\text{def}}{=} k$. We slightly abuse notation and also write L for the language associated to a linear path scheme L.

▶ **Example 2.** We claimed at the end of Example 1 that $A(3,5) \xrightarrow{*} A(3+2k,5)$ for every $k \in \mathbb{N}$. The case where $k = 0$ is trivial, so let us prove this property assuming that $k > 0$. Consider the linear path scheme

$$L = \delta_{AA} \cdot (\delta_{AB}\delta_{BB}\delta_{BB}\delta_{BA}\delta_{AA})^* \cdot \delta_{AB} \cdot (\delta_{BB})^* \cdot \delta_{BA} .$$

Note in passing that L has length $|L| = 9$ and $*$-length $|L|_* = 2$. To simplify notation, let $\pi \stackrel{\text{def}}{=} \delta_{AB}\delta_{BB}\delta_{BB}\delta_{BA}\delta_{AA}$. Observe that $A(0,x) \xrightarrow{\pi} A(0,x+2)$ for every $x \geq 2$. It follows that $A(3,5) \xrightarrow{\delta_{AA}} A(0,9) \xrightarrow{\pi^{k-1}} A(0,2k+7) \xrightarrow{\delta_{AB}} B(0,2k+7) \xrightarrow{(\delta_{BB})^{2k+2}} B(2k+2,5) \xrightarrow{\delta_{BA}} A(2k+3,5)$. We have shown that $A(3,5) \xrightarrow{L} A(3+2k,5)$ for every $k > 0$. ⌟

A binary relation R over configurations is called *flattenable*[1] if there exists a finite set Λ of linear path schemes such that $R \subseteq \bigcup_{L \in \Lambda} \xrightarrow{L}$. It is readily seen that the class of flattenable binary relations is closed under union and relational composition. We say that a d-TVASS $\mathcal{V}$ is *flattenable* when its reachability relation $\xrightarrow{*}$ is flattenable.

Flattenability of TVASS in Dimension Two. We showed sixteen years ago in [19] that 2-VASS are flattenable. Our approach was refined ten years later by Blondin et al. to provide bounds on the resulting linear path schemes and to show that the reachability problem for 2-VASS is solvable in polynomial space [1].

▶ **Theorem 3** ([19, 1])**.** *Every 2-VASS is flattenable. Furthermore, for every configurations $p(\mathbf{x})$ and $q(\mathbf{y})$ of a 2-VASS $\mathcal{V} = (Q, \Sigma, \Delta)$ such that $p(\mathbf{x}) \xrightarrow{*} q(\mathbf{y})$, there exists a linear path scheme L with $|L| \leq (|Q| + \|\Sigma\|)^{O(1)}$ and $|L|_* \leq O(|Q|^2)$ such that $p(\mathbf{x}) \xrightarrow{L} q(\mathbf{y})$.*

The remainder of this section is devoted to the extension of the first part of Theorem 3 to 2-TVASS. The existence of small linear path schemes witnessing flattenability will be shown in Sections 3 and 4, and ensuing complexity results will be presented in Sections 5 and 6.

Consider a 2-TVASS $\mathcal{V} = (Q, \Sigma, \Delta)$. We introduce, for each state $q \in Q$, the binary relation $\updownarrow_q$ over configurations defined by $\updownarrow_q = \{(q(0,x), q(0,y)) \mid q(0,x) \xrightarrow{*} q(0,y)\}$. This relation is called the *vertical loop relation* on q. We let $\updownarrow$ denote the union $\bigcup_{q \in Q} \updownarrow_q$. We first provide a decomposition of $\xrightarrow{*}$ in terms of $\xrightarrow{A^*}$, $\xrightarrow{T}$ and $\updownarrow$. Recall that A and T are the sets of addition transitions and zero-test transitions, respectively.

▶ **Lemma 4.** *It holds that* $\xrightarrow{*} \subseteq \left(\xrightarrow{A^*} \cup \xrightarrow{T} \cup \updownarrow\right)^{2|Q|+1}$.

The binary relations $\xrightarrow{A^*}$ and $\xrightarrow{T}$ are already known to be flattenable. This is a consequence of Theorem 3 for the former, and flattenability is obvious for the latter. As flattenable binary relations are closed under union and relational composition, it remains to show that $\updownarrow$ is flattenable. Vertical loops $q(0,x) \xrightarrow{*} q(0,y)$ either increase the second counter (i.e., $y > x$), or decrease it (i.e., $y < x$), or leave it unchanged (i.e., $y = x$). Let $\uparrow_q$ and $\downarrow_q$ denote the subrelations of $\updownarrow_q$ that correspond to the first and second cases, respectively. We first prove that the relations $\uparrow_q$ are flattenable.

[1] The same notion is often called *flattable* in the literature. It was simply called *flat* in [19].

Fix a state $q \in Q$ and assume that $\uparrow_q$ is not empty (otherwise it is trivially flattenable). We introduce the sequence $(D_x)_{x\in\mathbb{N}}$ of subsets of $\mathbb{N}$ defined by

$$D_x \;=\; \{d \in \mathbb{N} \mid q(0,x) \xrightarrow{*} q(0,x+d)\}$$

We derive from the monotonicity of 2-TVASS with respect to the second counter that $D_0 \subseteq D_1 \subseteq D_2 \cdots$ and that[2] $(D_x + D_x) \subseteq D_x$ for every $x \in \mathbb{N}$. These two properties entail that the sequence $(D_x)_{x\in\mathbb{N}}$ is ultimately stationary. Indeed, suppose by contradiction that there is an infinite increasing subsequence $\{0\} \subsetneq D_{x_0} \subsetneq D_{x_1} \subsetneq D_{x_2} \subsetneq \cdots$. We may extract $c, d_1, d_2, \ldots$ such that $c \in D_{x_0}$, $c > 0$, and $d_i \in D_{x_i} \setminus D_{x_{i-1}}$ for all $i > 0$. By the pigeonhole principle, some congruence class modulo c contains infinitely many d_i. So there exists $0 < i < j$ and $k \in \mathbb{N}$ such that $d_j = d_i + kc$. As d_i and c are both in D_{x_i}, we get from $(D_{x_i} + D_{x_i}) \subseteq D_{x_i}$ that $d_j \in D_{x_i}$, which is impossible since $D_{x_i} \subseteq D_{x_{j-1}}$ and $d_j \notin D_{x_{j-1}}$. We have shown that there exists $t \in \mathbb{N}$ such that $D_x = D_t$ for all $x \geq t$.

Recall that $\uparrow_q$ was assumed to be nonempty. So there exists $h \geq 0$, $m > 0$ and a run from $q(0,h)$ to $q(0,h+m)$. Let β denote the trace of this run. Note that β is a nonempty cycle on q. We derive from $q(0,h) \xrightarrow{\beta} q(0,h+m)$ that $(d+m) \in D_x$ for all $x \in \mathbb{N}$ and $d \in D_x$ with $d \geq h$. It follows that each D_x may be decomposed into $D_x = F_x \cup (B_x + \mathbb{N}m)$ where F_x and B_x are finite subsets of $\mathbb{N}$ such that $b \geq h$ for all $b \in B_x$. For every $d \in D_x$, let $\alpha_{x,d}$ denote the trace of some run from $q(0,x)$ to $q(0,x+d)$. Consider the finite set Λ of linear path schemes defined by

$$\Lambda \;=\; \bigcup_{x\leq t} \Lambda_x \qquad\qquad \text{and} \qquad\qquad \Lambda_x \;=\; \{\alpha_{x,f} \mid f \in F_x\} \;\cup\; \{\alpha_{x,b}\beta^* \mid b \in B_x\}\,.$$

Observe that Λ is finite as it collects the linear path schemes in Λ_x only for $x \leq t$. The linear path schemes in Λ_x with $x > t$ are redundant because of the above-established stabilization property of $(D_x)_{x\in\mathbb{N}}$. We obtain the following lemma by construction.

▶ **Lemma 5.** *It holds that $\uparrow_q \subseteq \bigcup_{L\in\Lambda} \xrightarrow{L}$, hence, the relation $\uparrow_q$ is flattenable.*

Notice that a decreasing vertical loop $q(0,x) \xrightarrow{*} q(0,y)$ with $y < x$ is an increasing vertical loop in the 2-TVASS $\overline{\mathcal{V}}$ obtained from $\mathcal{V}$ by reversing the effect of each transition, i.e., $\overline{\Delta} = \{\overline{\delta} \mid \delta \in \Delta\}$ where $\overline{(p,\mathbf{a},q)} = (q,-\mathbf{a},p)$ and $\overline{(p,\mathtt{tst},q)} = (q,\mathtt{tst},p)$. Put differently, the relation $\downarrow_q$ in $\mathcal{V}$ coincides with the relation $\uparrow_q$ in $\overline{\mathcal{V}}$. By applying Lemma 5 to $\overline{\mathcal{V}}$ and taking the mirror image of the resulting linear path schemes, we get that the relation $\downarrow_q$ in $\mathcal{V}$ is also flattenable. Since $\updownarrow = \bigcup_{q\in Q} \updownarrow_q$ and $\updownarrow_q \subseteq \uparrow_q \cup \downarrow_q \cup \xrightarrow{\varepsilon}$ for every $q \in Q$, we obtain that $\updownarrow$ is flattenable. Together with Lemma 4 and Theorem 3, this concludes the proof of the following theorem.

▶ **Theorem 6.** *Every 2-TVASS is flattenable.*

We have presented in this section a direct and simple proof that the reachability relation of every 2-TVASS is flattenable. This result entails, in particular, that the reachability relation is effectively semilinear for 2-TVASS (which was already known [9]) and computable by cycle acceleration techniques (see, e.g., [20]).

To derive complexity results from flattenability, we need to bound the length of linear path schemes witnessing flattenability. This requires a finer analysis of 2-TVASS runs than what was done for Theorem 6 (the latter will not be used in the remainder of the paper).

[2] The sum $A + B$ of two subsets $A, B \subseteq \mathbb{Z}$ is defined as $\{a + b \mid a \in A \wedge b \in B\}$.

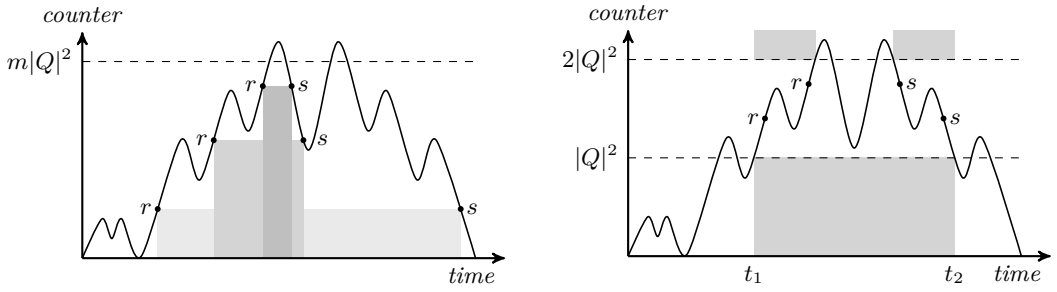


Figure 2 Illustration of the pumping lemmas for one-counter automata (Lemma 7 on the left and Lemma 9 on the right). The curves show the evolution of the counter along a run. Gray areas denote forbidden zones for the counter.

3 A Detour via Weighted One-Counter Automata

We have given in the previous section a simple proof that 2-TVASS are flattenable. This proof provides no bound on the length of the resulting linear path schemes, though. To obtain small linear path schemes witnessing flattenability, we take a detour via weighted one-counter automata. The rationale is that a 2-TVASS behaves like a one-counter automaton equipped with an additional counter (that cannot be tested for zero). When this additional counter is allowed to become negative, actions on it can be seen as weights. We show in this section that, in a weighted one-counter automaton, the reachable weights between two mutually reachable configurations $p(0)$ and $q(0)$ can be obtained via small linear path schemes. This will yield, for 2-TVASS, small linear path schemes for the reachability subrelations $p(0,x) \xrightarrow{*} q(0,y)$ such that x and y are large and $q(0,y) \xrightarrow{*} p(0,z)$ for some z. For simplicity, we consider weighted one-counter automata where addition actions and weights are in $\{-1,0,1\}$

A *weighted one-counter automaton* (shortly called a *WOCA*), is a quadruple $\mathcal{A} = (Q,\Sigma,\Delta,\lambda)$ where (Q,Σ,Δ) is a 1-TVASS such that $\Sigma = \{-1,0,1,\mathtt{tst}\}$ and $\lambda : \Delta \to \{-1,0,1\}$ is a *weight* function. All notions defined in Section 2 for 1-TVASS naturally carry over to WOCA. The weight function is extended to words in Δ^* by $\lambda(\delta_1 \cdots \delta_n) = \lambda(\delta_1) + \cdots + \lambda(\delta_n)$. The *weight* of a run is the weight of its trace. For notational convenience, we write $p(x) \xrightarrow[w]{\pi} q(y)$ when $p(x) \xrightarrow{\pi} q(y)$ and $w = \lambda(\pi)$. Similarly, we let $p(x) \xrightarrow[w]{*} q(y)$ stand for the existence of $\pi \in \Delta^*$ such that $p(x) \xrightarrow[w]{\pi} q(y)$.

As mentioned before, this section is devoted to the proof that, for any states p and q in a WOCA, the weights $w \in \mathbb{Z}$ such that $p(0) \xrightarrow[w]{*} q(0) \xrightarrow{*} p(0)$ can be obtained via small linear path schemes (see Theorem 12). We start with two pumping lemmas on runs of one-counter automata. The first one, Lemma 7, can be seen as an iterated version of the pumping lemma for one-counter languages due to Latteux [14]. It is an easy consequence of the classical *hill-cutting* technique for one-counter automata (often attributed to Valiant and Paterson [30]). The second one, Lemma 9, tunes the hill-cutting technique so as to obtain short extracted cycles. This second pumping lemma is crucial to obtain linear path schemes with short cycles. The hill-cutting techniques used in both lemmas are illustrated in Figure 2.

We assume for the remainder of this section that $\mathcal{A} = (Q,\Sigma,\Delta,\lambda)$ is a WOCA and that $p,q \in Q$ are states of $\mathcal{A}$.

▶ **Lemma 7.** *If $p(0) \xrightarrow{\pi} q(0)$ then for every $m > 0$ such that $|\pi| \geq m^2|Q|^3$, there exists a factorization $\pi = \alpha\beta_1 \cdots \beta_m\gamma\theta_m \cdots \theta_1\eta$, with $\beta_i\theta_i \neq \varepsilon$ for all $i \in [1, m]$, verifying*

$$p(0) \xrightarrow{\alpha\beta_1^{n_1}\cdots\beta_m^{n_m}\gamma\theta_m^{n_m}\cdots\theta_1^{n_1}\eta} q(0)$$

for every $n_1, \ldots, n_m \in \mathbb{N}$.

Proof Sketch. If the counter remains below $m|Q|^2$ then some configuration repeats at least $m + 1$ times, and the subruns in-between can be iterated arbitrarily many times. The cycles β_i come from these subruns and the cycles θ_i are empty. Otherwise, the run contains a "high hill" and we extract, for each counter value in $[0, m|Q|^2]$, a pair of configurations with this counter value (see Figure 2 (left)). This extraction proceeds from the inside of the hill towards the outside. Some pair of states (r, s) necessarily occurs $m + 1$ times in this extraction. The subruns between the r configurations provide the cycles β_i and the subruns between the s configurations provide the cycles θ_i. The extraction guarantees that these cycles can be iterated arbitrarily many times. ◀

▶ **Corollary 8.** *If $p(x) \xrightarrow{*} q(y)$ then $p(x) \xrightarrow{\pi} q(y)$ for some $\pi \in \Delta^*$ such that $|\pi| < (|Q|+x+y)^3$.*

▶ **Lemma 9.** *If $p(0) \xrightarrow{\pi} q(0)$ with $|\pi| \geq 2|Q|^3$ then there exists $r, s \in Q$, $x, d \in \mathbb{N}$, and a factorization $\pi = \alpha\beta\gamma\theta\eta$, with $\beta\theta \neq \varepsilon$ and no zero-test transition in γ, such that $x+d \leq 2|Q|^2$, $|\beta\theta| \leq 2|Q|^3$ and verifying*

$$p(0) \xrightarrow{\alpha} r(x) \xrightarrow{\beta^n} r(x + nd) \xrightarrow{\gamma} s(x + nd) \xrightarrow{\theta^n} s(x) \xrightarrow{\eta} q(0)$$

for every $n \in \mathbb{N}$.

Proof Sketch. If the counter remains below $2|Q|^2$ then some configuration repeats at least twice. So there is a subrun of length at most $2|Q|^3$ from some configuration $r(x)$ to the same configuration $r(x)$, and this subrun can be iterated arbitrarily many times. The cycle β comes from this subrun and the cycle θ is empty. Otherwise, the run contains a "high hill" and we extract, for each counter value in $[|Q|^2, 2|Q|^2]$, a pair of configurations with this counter value (see Figure 2 (right)). For the counter value $|Q|^2$, the pair of configurations is extracted from the inside of the hill towards the outside. Let t_1 and t_2 denote the positions of these configurations. For the counter values in $[|Q|^2 + 1, 2|Q|^2]$, this extraction proceeds from the outside of the hill – but limited to $[t_1, t_2]$ – towards the inside. Some pair of states (r, s) necessarily occurs twice in this extraction. The subrun between the two r configurations provides the cycle β and the subrun between the two s configurations provides the cycle θ. The extraction guarantees that these cycles can be iterated arbitrarily many times. By construction, the counter remains below $2|Q|^2$ in the subrun providing the cycle β (except possibly for the last configuration). If $\beta > |Q|^3$ then some configuration repeats at least twice in this subrun and we can proceed as in the first case of the proof. The same reasoning can also be applied to the cycle θ. ◀

We now exploit the two previous pumping lemmas to obtain short runs with appropriate weights. First, we show in Lemma 10 that if there is a run from $p(0)$ to $q(0)$ with positive (resp. negative) weight, then there is a short one. In fact, this lemma will be used in the particular case where $p = q$ to get short cyclic runs with positive (resp. negative) weights. Second, we show in Lemma 11 that, assuming that $p(0)$ and $q(0)$ are mutually reachable, if there is a run from $p(0)$ to $q(0)$ whose weight is in a given congruence class modulo $m > 0$, then there is a short one and the weight difference between the two runs can be "qualitatively compensated" by a cyclic run on $q(0)$.

▶ **Lemma 10.** *If $p(0) \xrightarrow[w]{*} q(0)$ for some $w \neq 0$ then $p(0) \xrightarrow[u]{\pi} q(0)$ for some $u \neq 0$ having the same sign as w and some $\pi \in \Delta^*$ such that $|\pi| \leq 539|Q|^9$.*

Proof. We only consider the case where the weight w is positive. The case where w is negative is symmetric. Assume that the set $\{\pi \in \Delta^* \mid p(0) \xrightarrow{\pi} q(0) \wedge \lambda(\pi) > 0\}$ is not empty, and take a word π of minimal length in that set. If $|\pi| < 2|Q|^3$ then we are done. Otherwise, by Lemma 9, there exists $r, s \in Q$, $x, d \in \mathbb{N}$, and a factorization $\pi = \alpha\beta\gamma\theta\eta$ satisfying the conditions of Lemma 9. Observe that $p(0) \xrightarrow{\alpha\gamma\eta} q(0)$ and $|\alpha\gamma\eta| < |\alpha\beta\gamma\theta\eta|$. We deduce from the minimality of π that $\lambda(\alpha\gamma\eta) \leq 0$. This entails that $\lambda(\beta\theta) = \lambda(\pi) - \lambda(\alpha\gamma\eta) > 0$. By Corollary 8, since $p(0) \xrightarrow{*} r(x)$ and $s(x) \xrightarrow{*} q(0)$, there exists α', η' both of length at most $(|Q| + x)^3$ such that $p(0) \xrightarrow{\alpha'} r(x)$ and $s(x) \xrightarrow{\eta'} q(0)$. Similarly, since $r(x) \xrightarrow{*} s(x)$ via a run with no zero-test, there exists γ' with no zero-test and of length $|\gamma'| \leq (|Q| + 2x)^3$ such that $r(x) \xrightarrow{\gamma'} s(x)$. As $x \leq 2|Q|^2$, we get that $|\alpha'\gamma'\eta'| \leq (5|Q|^2)^3 + 2(3|Q|^2)^3 \leq 179|Q|^6$. Now consider the word π' defined by $\pi' \stackrel{\text{def}}{=} \alpha'\beta^n\gamma'\theta^n\eta'$ where $n = 1 + |\alpha'\gamma'\eta'|$. Note that $p(0) \xrightarrow{\pi'} q(0)$ as γ' contains no zero-test and the factorization $\pi = \alpha\beta\gamma\theta\eta$ satisfies the conditions of Lemma 9. Moreover, $\lambda(\pi') = \lambda(\alpha'\gamma'\eta') + n\lambda(\beta\theta)$, hence, $\lambda(\pi') \geq -|\alpha'\gamma'\eta'| + n > 0$. We deduce from the minimality of π that $|\pi| \leq |\pi'|$. It remains to show that π' is short. By construction, $|\pi'| = |\alpha'\gamma'\eta'| + n|\beta\theta| \leq 179|Q|^6 + 180|Q|^6 \cdot 2|Q|^3$. We obtain that $|\pi'| \leq 539|Q|^9$, which concludes the proof of the lemma. ◀

▶ **Lemma 11.** *If $p(0) \xrightarrow[w]{*} q(0) \xrightarrow{*} p(0)$ then for every $m > 0$, there exists $\pi \in \Delta^*$ with $|\pi| < m^2|Q|^3$ verifying $p(0) \xrightarrow{\pi} q(0)$ and $\lambda(\pi) \equiv w \pmod m$, and such that if $w \neq \lambda(\pi)$ then $q(0) \xrightarrow[v]{*} q(0)$ for some $v \neq 0$ having the same sign as $w - \lambda(\pi)$.*

Proof. Assume that $p(0) \xrightarrow[w]{*} q(0) \xrightarrow{*} p(0)$ and let $m > 0$. Let $C(\pi)$ denote the condition that if $w \neq \lambda(\pi)$ then $q(0) \xrightarrow[v]{*} q(0)$ for some $v \neq 0$ having the same sign as $w - \lambda(\pi)$. Consider the set S of all words $\pi \in \Delta^*$ such that $p(0) \xrightarrow{\pi} q(0)$, $\lambda(\pi) \equiv w \pmod m$ and $C(\pi)$ holds. This set is not empty since $p(0) \xrightarrow[w]{*} q(0)$. Take a word π of minimal length in S and let us prove that $|\pi|$ meets the desired bound. Suppose, by contradiction, that $|\pi| \geq m^2|Q|^3$. By Lemma 7, there exists a factorization $\pi = \alpha\beta_1 \cdots \beta_m\gamma\theta_m \cdots \theta_1\eta$, with $\beta_i\theta_i \neq \varepsilon$ for all $i \in [1, m]$, such that $p(0) \xrightarrow{\alpha\beta_1^{n_1} \cdots \beta_m^{n_m}\gamma\theta_m^{n_m} \cdots \theta_1^{n_1}\eta} q(0)$ for every $n_1, \ldots, n_m \in \mathbb{N}$. Let $u_i \stackrel{\text{def}}{=} \lambda(\beta_1 \cdots \beta_i\theta_i \cdots \theta_1)$ for every $i \in [0, m]$, with the understanding that $u_0 = \lambda(\varepsilon) = 0$, and consider the sequence $u_0, \ldots, u_m$. By the pigeonhole principle, there exists $0 \leq i < j \leq m$ such that u_i and u_j are in the same congruence class modulo m. So $u_j = u_i + u$ for some $u \in \mathbb{Z}m$. This means that $\lambda(\beta_{i+1} \cdots \beta_j\theta_j \cdots \theta_{i+1}) = u$. Now, for each $k \in \mathbb{N}$, let π'_k denote the word obtained from π by taking the cycles $\beta_{i+1}, \ldots, \beta_j$ and $\theta_j, \ldots, \theta_{i+1}$ exactly k times, formally, $\pi'_k \stackrel{\text{def}}{=} \alpha\beta_1 \cdots \beta_i\beta_{i+1}^k \cdots \beta_j^k\beta_{j+1} \cdots \beta_m\gamma\theta_m \cdots \theta_{j+1}\theta_j^k \cdots \theta_{i+1}^k\theta_i \cdots \theta_1\eta$. It is readily seen that $p(0) \xrightarrow{\pi'_k} q(0)$ and $\lambda(\pi'_k) = \lambda(\pi) + (k-1)u$.

Let us prove that $\pi'_0 \in S$. We have already shown that $p(0) \xrightarrow{\pi'_0} q(0)$ and $\lambda(\pi'_0) = \lambda(\pi) - u$, hence, $\lambda(\pi'_0) \equiv w \pmod m$. It remains to show that $C(\pi'_0)$ holds. Let $s, s' \in \{-1, 0, 1\}$ denote the signs of $w - \lambda(\pi)$ and $w - \lambda(\pi'_0)$, respectively. If $s' = 0$ then $C(\pi'_0)$ holds trivially. If $s' \neq 0$ and $s = s'$ then $C(\pi'_0)$ holds because $C(\pi)$ holds. If $s' = 1$ and $s \leq 0$ then $\lambda(\pi'_0) < w \leq \lambda(\pi)$, hence, $u > 0$. It follows from $\lambda(\pi'_k) = \lambda(\pi) + (k-1)u$ that $p(0) \xrightarrow[v]{*} q(0)$ for infinitely many $v > 0$. As $q(0) \xrightarrow{*} p(0)$, we deduce that $q(0) \xrightarrow[v]{*} q(0)$ for some $v > 0$, hence, $C(\pi'_0)$ holds. If $s' = -1$ and $s \geq 0$ then $\lambda(\pi) \leq w < \lambda(\pi'_0)$, hence, $u < 0$. It follows from

$\lambda(\pi'_k) = \lambda(\pi)+(k-1)u$ that $p(0) \xrightarrow{*}_{v} q(0)$ for infinitely many $v < 0$. As $q(0) \xrightarrow{*} p(0)$, we deduce that $q(0) \xrightarrow{*}_{v} q(0)$ for some $v < 0$, hence, $C(\pi'_0)$ holds. We have shown in all cases that $\pi'_0 \in S$. This contradicts the minimality of π since $|\pi'_0| = |\pi| - |\beta_{i+1} \cdots \beta_j \theta_j \cdots \theta_{i+1}| < |\pi|$. ◀

We are now ready to prove the main result of this section, namely that the reachable weights between two mutually reachable configurations $p(0)$ and $q(0)$ can be obtained via small linear path schemes.

▶ **Theorem 12.** *Let $\mathcal{A} = (Q, \Sigma, \Delta, \lambda)$ be a WOCA. For every states $p, q \in Q$ and weight $w \in \mathbb{Z}$ verifying $p(0) \xrightarrow{*}_{w} q(0) \xrightarrow{*} p(0)$, there exists $\alpha, \beta \in \Delta^*$ and $n \in \mathbb{N}$ such that $p(0) \xrightarrow{\alpha\beta^n}_{w} q(0)$, $q(0) \xrightarrow{\beta} q(0)$ and $|\alpha\beta| \leq (2|Q|)^{39}$.*

Proof. Assume that $p(0) \xrightarrow{*}_{w} q(0) \xrightarrow{*} p(0)$. We start by fixing two short cyclic runs on $q(0)$, one with positive weight and one with negative weight, as follows. By Lemma 10, if $q(0) \xrightarrow{*}_{w} q(0)$ for some $w > 0$ then $q(0) \xrightarrow{\beta} q(0)$ for some $\beta \in \Delta^*$ such that $\lambda(\beta) > 0$ and $|\beta| \leq 539|Q|^9$. Let $\beta \stackrel{\text{def}}{=} \varepsilon$ otherwise. Analogously, if $q(0) \xrightarrow{*}_{w} q(0)$ for some $w < 0$ then $q(0) \xrightarrow{\theta} q(0)$ for some $\theta \in \Delta^*$ such that $\lambda(\theta) < 0$ and $|\theta| \leq 539|Q|^9$. Let $\theta \stackrel{\text{def}}{=} \varepsilon$ otherwise.

By Lemma 11, for each $m \in \{1, \lambda(\beta), -\lambda(\theta), -\lambda(\beta)\lambda(\theta)\}$ such that $m > 0$, there exists $\alpha_m \in \Delta^*$ with $|\alpha_m| < m^2|Q|^3$ verifying $p(0) \xrightarrow{\alpha_m} q(0)$ and $\lambda(\alpha_m) \equiv w \pmod m$, and such that if $w \neq \lambda(\alpha_m)$ then $q(0) \xrightarrow{*}_{v} q(0)$ for some $v \neq 0$ having the same sign as $w - \lambda(\alpha_m)$. Let $u_m \stackrel{\text{def}}{=} w - \lambda(\alpha_m)$ and note that $u_m \in \mathbb{Z}m$. Moreover, $u_m > 0$ implies $\beta \neq \varepsilon$ and $u_m < 0$ implies $\theta \neq \varepsilon$. We now consider four cases depending on the emptiness of β and θ.

If $\beta = \theta = \varepsilon$ then we use α_m and u_m for $m \stackrel{\text{def}}{=} 1$. We derive from $\beta = \theta = \varepsilon$ that $u_m = 0$. It follows that $w = \lambda(\alpha_m) + u_m = \lambda(\alpha_m\beta)$.

If $\beta \neq \varepsilon$ and $\theta = \varepsilon$ then we use α_m and u_m for $m \stackrel{\text{def}}{=} \lambda(\beta) > 0$. We derive from $\theta = \varepsilon$ that $u_m \geq 0$. As $u_m \in \mathbb{Z}m$, we get that $u_m = n\lambda(\beta)$ for some $n \in \mathbb{N}$. It follows that $w = \lambda(\alpha_m) + u_m = \lambda(\alpha_m\beta^n)$.

If $\beta = \varepsilon$ and $\theta \neq \varepsilon$ then we use α_m and u_m for $m \stackrel{\text{def}}{=} -\lambda(\theta) > 0$. We derive from $\beta = \varepsilon$ that $u_m \leq 0$. As $u_m \in \mathbb{Z}m$, we get that $u_m = n\lambda(\theta)$ for some $n \in \mathbb{N}$. It follows that $w = \lambda(\alpha_m) + u_m = \lambda(\alpha_m\theta^n)$.

If $\beta \neq \varepsilon$ and $\theta \neq \varepsilon$ then we use α_m and u_m for $m \stackrel{\text{def}}{=} -\lambda(\beta)\lambda(\theta) > 0$. As $u_m \in \mathbb{Z}m$, we get that $u_m = n\lambda(\beta)\lambda(\theta)$ for some $n \in \mathbb{Z}$. If $n \leq 0$ then $w = \lambda(\alpha_m) + u_m = \lambda(\alpha_m\beta^{n\lambda(\theta)})$. If $n \geq 0$ then $w = \lambda(\alpha_m) + u_m = \lambda(\alpha_m\theta^{n\lambda(\beta)})$.

We have shown in each case that $w = \lambda(\alpha_m\gamma^n)$ for some $m \in \{1, \lambda(\beta), -\lambda(\theta), -\lambda(\beta)\lambda(\theta)\}$ with $m > 0$, some $\gamma \in \{\beta, \theta\}$ and some $n \in \mathbb{N}$. Moreover, our choice of β and θ ensures that $m \leq (539|Q|^9)^2$, $q(0) \xrightarrow{\gamma} q(0)$ and $|\gamma| \leq 539|Q|^9$. Recall that $p(0) \xrightarrow{\alpha_m} q(0)$ and $|\alpha_m| < m^2|Q|^3$. It follows that $p(0) \xrightarrow{\alpha_m\gamma^n}_{w} q(0)$ and $|\alpha_m| \leq (539|Q|^9)^4|Q|^3$. We obtain that $|\alpha_m\gamma| \leq (2|Q|)^{39}$, which concludes the proof of the theorem. ◀

4 Succinct Flattenability of 2-TVASS

We have shown in Section 2 that 2-TVASS are flattenable. We now prove that flattenability of 2-TVASS can be witnessed by small linear path schemes.

We first introduce a binary relation $\rightsquigarrow$ over the states of a 2-TVASS $\mathcal{V}$, defined by $p \rightsquigarrow q$ if $p(0,x) \xrightarrow{*} q(0,y)$ for some $x, y \in \mathbb{N}$. Notice that this relation is transitive since $p(0,x) \xrightarrow{*} q(0,y)$ and $q(0,x') \xrightarrow{*} r(0,y')$ implies $p(0,x+x') \xrightarrow{*} q(0,y+x') \xrightarrow{*} r(0,y+y')$ by

monotonicity. As mentioned in Section 3, a WOCA can be associated to any 2-TVASS by considering actions on the second counter, the one that is not tested for zero, as weights. Under this observation, $p \rightsquigarrow q$ if, and only if, $p(0) \xrightarrow{*} q(0)$ in the associated WOCA. This observation also provides a way to convert Theorem 12 to the following lemma.

▶ **Lemma 13.** *There exists a constant $h \geq 1$ such that, for every 2-TVASS $\mathcal{V} = (Q, \Sigma, \Delta)$, if $p(0,x) \xrightarrow{*} q(0,y)$ with $x, y \geq (|Q| + \|\Sigma\|)^h$ and $q \rightsquigarrow p$, then there exists a linear path scheme L with $|L| \leq (|Q| + \|\Sigma\|)^{O(1)}$ and $|L|_* = 1$ such that $p(0,x) \xrightarrow{L} q(0,y)$.*

Proof. A 2-TVASS cannot be directly translated into a WOCA since some addition transitions $(p, \mathbf{a}, q)$ may satisfy $\|\mathbf{a}\| > 1$. However, by introducing intermediate states and transitions between p and q, we can overcome this problem. It follows that we can assume, without loss of generality, that every addition transition $(p, \mathbf{a}, q)$ satisfies $\|\mathbf{a}\| \leq 1$. Additionally, by introducing for each state p and each addition transition δ an intermediate state, we can assume that if $(p, \mathbf{a}, q)$ and $(p, \mathbf{b}, q)$ are two addition transitions such that $\mathbf{a}(1) = \mathbf{b}(1)$ then $\mathbf{a}(2) = \mathbf{b}(2)$. Thanks to this assumption, we can associate to a 2-TVASS $\mathcal{V} = (Q, \Sigma, \Delta)$ a WOCA $(Q, \Sigma', \Delta', \lambda)$ in such a way $(p, (a, b), q)$ is a transition in $\mathcal{V}$ if, and only if, (p, a, q) is a transition in the WOCA weighted by b, and such that $(p, \mathtt{tst}, q)$ is a transition in $\mathcal{V}$ if, and only if, $(p, \mathtt{tst}, q)$ is a transition in the WOCA, and in that case the transition is weighted by zero. Now, let us consider two configurations $p(0,x)$ and $q(0,y)$ with $x, y \geq (2|Q|)^{39}$ such that $p(0,x) \xrightarrow{*} q(0,y)$ and $q \rightsquigarrow p$ in $\mathcal{V}$. Notice that $p(0) \xrightarrow[w]{*} q(0)$ and $q(0) \xrightarrow{*} p(0)$ in the WOCA with $w = y - x$. From Theorem 12, it follows that there exists a path π from p to q, a cycle θ on q, and $n \in \mathbb{N}$ such that $p(0) \xrightarrow[w]{\pi\theta^n} q(0)$ with $|\pi\theta| \leq (2|Q|)^{39}$. Notice that π and θ in the WOCA corresponds to a path α and a cycle β in $\mathcal{V}$, respectively. Since $x, y \geq (2|Q|)^{39} \geq |\alpha\beta|$, observe that $p(0,x) \xrightarrow{\alpha\beta^n} q(0,y)$ since each execution of β can decrease or increase the second counter by a value bounded by $|\beta| \leq (2|Q|)^{39}$. ◀

The previous lemma captures the reachability relation of a 2-TVASS between configurations $p(0,x)$ and $q(0,y)$ with $p \rightsquigarrow q \rightsquigarrow p$ and x, y are large. In order to capture the same relation when x or y are small, the following result will be useful.

▶ **Theorem 14** ([7])**.** *For every 2-VASS $\mathcal{V} = (Q, \Sigma, \Delta)$, and for every configurations $p(\mathbf{x})$ and $q(\mathbf{y})$ such that $p(\mathbf{x}) \xrightarrow{*} q(\mathbf{y})$ in $\mathcal{V}$, there exists a path π such that $p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$ and satisfying*

$$|\pi| \leq (|Q| + \|\Sigma\| + \|\mathbf{x}\| + \|\mathbf{y}\|)^{O(1)} .$$

We are now ready to refine Lemma 5 with complexity bounds. Recall that $\updownarrow_q$ is the vertical loop relation on q defined by $\updownarrow_q = \{(q(0,x), q(0,y)) \mid q(0,x) \xrightarrow{*} q(0,y)\}$.

▶ **Lemma 15.** *For every 2-TVASS $\mathcal{V} = (Q, \Sigma, \Delta)$ and state $q \in Q$, we have $\updownarrow_q \subseteq \bigcup_{L \in \Lambda} \xrightarrow{L}$ for some finite set Λ of linear path schemes L such that $|L| \leq (|Q| + \|\Sigma\|)^{O(1)}$ and $|L|_* \leq O(|Q|^2)$.*

Proof. Let $h \geq 1$ be the constant of Lemma 13, and let $c \geq 1$ be a constant satisfying $O(1) \leq c$ and $O(|Q|^2) \leq c|Q|^2$ in Lemma 13, Theorem 3, and Theorem 14. Let us consider a 2-TVASS $\mathcal{V} = (Q, \Sigma, \Delta)$ and let $N = |Q| + \|\Sigma\|$.

Observe that a run from a configuration $q(0,x)$ to a configuration $q(0,y)$ can be split in such a way:

$$q(0,x) = q_1(0,x_1) \xrightarrow{A^* \cup T} q_2(0,x_2) \cdots \xrightarrow{A^* \cup T} q_k(0,x_k) = q(0,y)$$

Moreover, by removing some parts of such a run, we can assume that the configurations $q_j(0,x_j)$ are pairwise distinct.

Notice that if $x_j < N^h$ for every $j \in [1,k]$, then $k \leq |Q| \cdot N^h \leq N^{h+1}$. By applying Theorem 14, we deduce that for every $j \in [2,k]$, there exists a path α_j such that $q_{j-1}(0,x_{j-1}) \xrightarrow{\alpha_j} q_j(0,x_j)$ with $|\alpha_j| \leq (N+2N^h)^c$. In particular α defined as $\alpha_2 \cdots \alpha_k$ is a path such that $q(0,x) \xrightarrow{\alpha} q(0,y)$ with $|\alpha| \leq k \cdot (N+2N^h)^c \leq N^e$ for some constant e. We are done with the linear path scheme $L = \alpha$. So, we can assume that there exists j such that $x_j \geq N^h$. In that case, we introduce j_{min} and j_{max} respectively defined as the minimal and the maximal j satisfying this property.

Let us prove that there exists a linear path scheme L_{min} such that $|L_{\mathsf{min}}| \leq N^e + N^c$ and $|L_{\mathsf{min}}|_* \leq c|Q|^2$ and such that $q(0,x) \xrightarrow{L_{\mathsf{min}}} q_{j_{\mathsf{min}}}(0,x_{j_{\mathsf{min}}})$. Observe that if $j_{\mathsf{min}} = 1$, the proof is immediate with L_{min} reduced to the empty path. If $j_{\mathsf{min}} > 1$, as $x_j < N^h$ for every $1 \leq j < j_{\mathsf{min}}$, we deduce from the previous paragraph that there exists a path α_{min} with a length bounded by N^e such that $q(0,x) \xrightarrow{\alpha_{\mathsf{min}}} q_{j_{\mathsf{min}}-1}(0,x_{j_{\mathsf{min}}-1})$. Recall that $q_{j_{\mathsf{min}}-1}(0,x_{j_{\mathsf{min}}-1}) \xrightarrow{A^* \cup T} q_{j_{\mathsf{min}}}(0,x_{j_{\mathsf{min}}})$. Based on Theorem 3, we deduce that there exists a linear path scheme L_0 such that $q_{j_{\mathsf{min}}-1}(0,x_{j_{\mathsf{min}}-1}) \xrightarrow{L_0} q_{j_{\mathsf{min}}}(0,x_{j_{\mathsf{min}}})$ with $|L_0| \leq N^c$ and $|L_0|_* \leq c|Q|^2$. By considering $L_{\mathsf{min}} = \alpha_{\mathsf{min}} L_0$ we are done. Symmetrically, there exists a linear path scheme L_{max} such that $|L_{\mathsf{max}}| \leq N^e + N^c$ and $|L_{\mathsf{max}}|_* \leq c|Q|^2$ and such that $q_{j_{\mathsf{max}}}(0,x_{j_{\mathsf{max}}}) \xrightarrow{L_{\mathsf{max}}} q(0,y)$.

Note that $q_{j_{\mathsf{max}}} \rightsquigarrow q_k = q_1 \rightsquigarrow q_{j_{\mathsf{min}}}$, hence, $q_{j_{\mathsf{max}}} \rightsquigarrow q_{j_{\mathsf{min}}}$. By applying Lemma 13, we deduce that there exists a linear path scheme L_1 with $|L_1| \leq N^c$ and $|L|_* = 1$ such that $q_{j_{\mathsf{min}}}(0,x_{j_{\mathsf{min}}}) \xrightarrow{L_1} q_{j_{\mathsf{max}}}(0,x_{j_{\mathsf{max}}})$, It follows that the linear path scheme L defined as $L_{\mathsf{min}} L_1 L_{\mathsf{max}}$ satisfies the lemma. ◀

▶ **Corollary 16.** *Every 2-TVASS is flattenable. Furthermore, for every configurations $p(\mathbf{x})$ and $q(\mathbf{y})$ of a 2-TVASS $\mathcal{V} = (Q, \Sigma, \Delta)$ such that $p(\mathbf{x}) \xrightarrow{*} q(\mathbf{y})$, there exists a linear path scheme L with $|L| \leq (|Q| + \|\Sigma\|)^{O(1)}$ and $|L|_* \leq O(|Q|^3)$ such that $p(\mathbf{x}) \xrightarrow{L} q(\mathbf{y})$.*

Proof. The proof is a direct corollary of Lemma 4, Theorem 3, and Lemma 15. ◀

▶ **Example 17.** As an illustration of Corollary 16, let us continue Examples 1 and 2 and provide a finite set Λ of "small" linear path schemes such that $A(\mathbf{x}) \xrightarrow{*} B(\mathbf{y})$ if, and only if, $A(\mathbf{x}) \xrightarrow{L} B(\mathbf{y})$ for some $L \in \Lambda$. First, we observe that for every $x, y \in \mathbb{N}$, if $A(0,x) \xrightarrow{*} A(0,y)$ then $x = y$ or the following condition is satisfied:

$$x \geq 2 \ \wedge \ y \geq x+2 \ \wedge \ (y = x+3 \Rightarrow x \geq 5) \ \wedge \ (y = x+5 \Rightarrow x \geq 3)\,.$$

Second, we introduce the paths $\pi = \delta_{AB}\delta_{BB}\delta_{BB}\delta_{BA}\delta_{AA}$ and $\sigma = \delta_{AB}(\delta_{BB})^5\delta_{BA}(\delta_{AA})^2$. It is routinely checked that $A(0,x) \xrightarrow{\pi} A(0,y)$ if, and only if, $x \geq 2$ and $y = x+2$. Similarly, $A(0,x) \xrightarrow{\sigma} A(0,y)$ if, and only if, $x \geq 5$ and $y = x+3$. We derive that $A(0,x) \xrightarrow{*} A(0,y)$ if, and only if, $A(0,x) \xrightarrow{\pi^* \cdot \{\varepsilon,\sigma\}} A(0,y)$. We are now done by taking $\Lambda = \{L_1, L_2\}$ where L_1 and L_2 are the linear path schemes defined by $L_1 = (\delta_{AA})^* \cdot \pi^* \cdot \delta_{AB} \cdot (\delta_{BB})^*$ and $L_2 = (\delta_{AA})^* \cdot \pi^* \cdot \sigma\delta_{AB} \cdot (\delta_{BB})^*$. ⌟

5 Linear Path Schemes to Systems of Equations

In this section, we associate to a linear path scheme $L = \alpha_0\beta_1^*\alpha_1 \cdots \beta_k^*\alpha_k$ of a d-TVASS $\mathcal{V}$ from a state p to a state q, and to a vectors $\mathbf{x}, \mathbf{y} \in \mathbb{N}^d$, a system of linear inequalities $S_{\mathbf{x},L,\mathbf{y}}$ encoding over the variables $(n_1, \ldots, n_k)$ the following constraint:

$$p(\mathbf{x}) \xrightarrow{\alpha_0\beta_1^{n_1}\alpha_1\cdots\beta_k^{n_k}\alpha_k} q(\mathbf{y})$$

Such a system is classical for d-VASS, but for d-TVASS, the presence of zero-test transitions in the linear path scheme L requires some additional work.

Let us first characterize the binary relation $\xrightarrow{\pi}$ thanks to a system of linear inequalities associated to a path π. We introduce the *displacement* $\mathrm{disp}(\delta)$ of a transition δ as the vector in $\mathbb{Z}^d$ defined by $\mathrm{disp}(\delta) \stackrel{\text{def}}{=} \mathbf{a}$ if δ is of the form $(p, \mathbf{a}, q)$ with $\mathbf{a} \in \mathbb{Z}^d$ and $\mathrm{disp}(\delta) \stackrel{\text{def}}{=} \mathbf{0}$ if δ is of the form $(p, \mathtt{tst}, q)$. The *displacement* of a path $\pi = \delta_1 \dots \delta_n$ is $\mathrm{disp}(\pi) \stackrel{\text{def}}{=} \mathrm{disp}(\delta_1)+\cdots+\mathrm{disp}(\delta_n)$. We also introduce the vector $\mathbf{m}_\pi \in \mathbb{N}^d$ defined component-wise for every $i \in [1, d]$ by $\mathbf{m}_\pi(i) \stackrel{\text{def}}{=} \max_\alpha(-\mathrm{disp}(\alpha)(i))$ where α ranges over the prefixes of π.

A path π from a state p to a state q is said to be *feasible* if $p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$ for some $\mathbf{x}, \mathbf{y} \in \mathbb{N}^d$. We introduce the partial order $\geq_1$ defined over $\mathbb{N}^d$ by $\mathbf{x} \geq_1 \mathbf{y}$ if $\mathbf{x}(1) = \mathbf{y}(1)$ and $\mathbf{x}(i) \geq \mathbf{y}(i)$ for every $i \in [2, d]$. We let $\succeq_\pi$ denote the partial order over $\mathbb{N}^d$ defined as follows: $\succeq_\pi$ is $\geq_1$ if π contains a zero-test transition, and $\succeq_\pi$ is $\geq$ otherwise.

▶ **Lemma 18.** *Let π be a feasible path from a state p to a state q. For every $\mathbf{x}, \mathbf{y} \in \mathbb{N}^d$, we have:*

$$p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y}) \iff \mathbf{x} \succeq_\pi \mathbf{m}_\pi \ \wedge \ \mathbf{y} = \mathbf{x} + \mathrm{disp}(\pi)$$

Let us recall that in Section 2 we introduce the d-TVASS $\overline{\mathcal{V}}$ obtained from $\mathcal{V}$ by reversing the effect of each transition, i.e., $\overline{\Delta} = \{\overline{\delta} \mid \delta \in \Delta\}$ where $\overline{(p, \mathbf{a}, q)} = (q, -\mathbf{a}, p)$ and $\overline{(p, T, q)} = (q, T, p)$. Given a path $\pi = \delta_1 \cdots \delta_n$ from p to q in $\mathcal{V}$, we introduce the path $\overline{\pi}$ from q to p in $\overline{\mathcal{V}}$ defined as $\overline{\pi} \stackrel{\text{def}}{=} \overline{\delta_n} \cdots \overline{\delta_1}$. Observe that $p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$ if, and only if, $q(\mathbf{y}) \xrightarrow{\overline{\pi}} p(\mathbf{x})$.

▶ **Lemma 19.** *We have $\mathbf{m}_{\overline{\pi}} = \mathbf{m}_\pi + \mathrm{disp}(\pi)$.*

Proof. Observe that for any decomposition of π into $\alpha\alpha'$, we have $\mathrm{disp}(\pi) = \mathrm{disp}(\alpha)+\mathrm{disp}(\alpha')$. Hence $-\mathrm{disp}(\alpha) + \mathrm{disp}(\pi) = \mathrm{disp}(\alpha') = -\mathrm{disp}(\overline{\alpha}')$. In particular $\max_\alpha(-\mathrm{disp}(\alpha)(i) + \mathrm{disp}(\pi)(i)) = \max_{\alpha'}(-\mathrm{disp}(\overline{\alpha}')(i))$ for every $i \in [1, d]$ where α ranges over the prefixes of π and α' over the suffixes of π. By observing that $\overline{\alpha}'$ ranges over all the prefixes of $\overline{\pi}$ when α' ranges over the suffixes of π, we get $\mathbf{m}_\pi + \mathrm{disp}(\pi) = \mathbf{m}_{\overline{\pi}}$. ◀

We are now ready to express the relation $\xrightarrow{\beta^n}$ where β is a cycle on a state q and $n \geq 1$ is a positive natural number.

▶ **Lemma 20.** *Let β be a feasible cycle on a state q. For every $\mathbf{x}, \mathbf{y} \in \mathbb{N}^d$ and $n \in \mathbb{N} \setminus \{0\}$, we have:*

$$q(\mathbf{x}) \xrightarrow{\beta^n} q(\mathbf{y}) \iff \mathbf{x} \succeq_\pi \mathbf{m}_\beta \ \wedge \ \mathbf{y} \succeq_\pi \mathbf{m}_{\overline{\beta}} \ \wedge \ \mathbf{y} = \mathbf{x} + n\,\mathrm{disp}(\beta)$$

A linear path scheme $L = \alpha_0\beta_1^*\alpha_1 \cdots \beta_k^*\alpha_k$ is said to be *feasible* if the paths $\alpha_0, \dots, \alpha_k$ and the cycles $\beta_1, \dots, \beta_k$ are feasible. We are now ready to introduce a system of linear inequalities $S_{\mathbf{x},L,\mathbf{y}}$ over the variables $(n_1, \dots, n_k)$ where $\mathbf{x}, \mathbf{y} \in \mathbb{N}^d$, and $n_1, \dots, n_k$ are variables ranging over $\mathbb{N}$ as follows:

$$\bigwedge_{j=0}^{k} \mathbf{x}_j \succeq_{\alpha_j} \mathbf{m}_{\alpha_j} \quad \wedge \quad \bigwedge_{j=1}^{k} \mathbf{y}_{j-1} \succeq_{\beta_j} \mathbf{m}_{\beta_j} \wedge \mathbf{x}_j \succeq_{\beta_j} \mathbf{m}_{\overline{\beta_j}} \quad \wedge \quad \mathbf{y} = \mathbf{y}_k$$

where $\mathbf{x}_0$ is the expression $\mathbf{x}$, and by induction over j, by letting $\mathbf{y}_j$ be the expression $\mathbf{x}_j + \mathrm{disp}(\alpha_j)$ for every $j \in [0, k]$, and $\mathbf{x}_j$ is the expression $\mathbf{y}_{j-1} + n_j\,\mathrm{disp}(\beta_j)$ for every $j \in [1, k]$. From Lemmas 18 and 20, we derive the following corollary.

▶ **Corollary 21.** *Assume that $L = \alpha_0\beta_1^*\alpha_1 \cdots \beta_k^*\alpha_k$ is a feasible linear path scheme from a state p to a state q, and let $\mathbf{x}, \mathbf{y} \in \mathbb{N}^d$. If $(n_1, \ldots, n_k)$ is a solution of $S_{\mathbf{x},L,\mathbf{y}}$ then*

$$p(\mathbf{x}) \xrightarrow{\alpha_0\beta_1^{n_1}\alpha_1\cdots\beta_k^{n_k}\alpha_k} q(\mathbf{y})$$

Conversely, a tuple $(n_1, \ldots, n_k)$ with $n_1, \ldots, n_k \geq 1$ that satisfies the previous relation is a solution of $S_{\mathbf{x},L,\mathbf{y}}$.

▶ Remark 22. We can easily extend the definition of $S_{\mathbf{x},L,\mathbf{y}}$ to encode linear path schemes of extended d-TVASS with zero-test actions on any counter.

6 Complexity Results

This section utilizes the results of Sections 4 and 5 to characterize the complexity of the reachability problem in 2-TVASS. We assume that 2-TVASS and their configurations are encoded in binary, and that sizes are defined as expected (up to a polynomial). Under this binary encoding, the reachability problem in 2-TVASS is shown to be PSPACE-complete. Since the reachability problem for 2-VASS (i.e., without zero-test transitions) is already PSPACE-hard [8], we only need to prove PSPACE-membership.

The PSPACE complexity upper-bound is obtained via small solutions of the system of inequalities $S_{\mathbf{x},L,\mathbf{y}}$ associated to a linear path scheme L and a pair of vectors $\mathbf{x}, \mathbf{y} \in \mathbb{N}^2$. We first recall some results about small solution of systems of equations.

▶ **Theorem 23** ([27])**.** *Let $M = (M_{i,j})$ be a matrix in $\mathbb{Z}^{e\times k}$. Every solution $\mathbf{x} \in \mathbb{N}^k$ of $M\mathbf{x} = \mathbf{0}$ is a finite sum of solutions $\mathbf{y} \in \mathbb{N}^k$ satisfying additionally $\sum_{i=1}^k \mathbf{y}(i) \leq (1+m)^k$ where $m = \max_i \sum_{j=1}^k |M_{i,j}|$.*

We also recall the classical application of the previous theorem to systems of inequalities with constant terms (the vector $\mathbf{b}$ in the following corollary).

▶ **Corollary 24.** *Let $M = (M_{i,j})$ be a matrix in $\mathbb{Z}^{e\times k}$ and let $\mathbf{b} \in \mathbb{Z}^e$. If there exists a solution $\mathbf{x} \in \mathbb{N}^k$ of $M\mathbf{x} \geq \mathbf{b}$ then there exists a solution $\mathbf{y} \in \mathbb{N}^k$ such that $\sum_{i=1}^k \mathbf{y}(i) \leq (2+m)^{k+1+e}$ where $m = \max_i \sum_{j=1}^k |M_{i,j}| + |\mathbf{b}(i)|$.*

Proof. Let $\mathbf{y}$ be the vector in $\mathbb{N}^e$ defined as $\mathbf{y} \stackrel{\text{def}}{=} M\mathbf{x} - \mathbf{b}$ and observe that $(\mathbf{x}, 1, \mathbf{y})$ is a solution of $M\mathbf{x} - t\mathbf{b} - \mathbf{y} = \mathbf{0}$. From Theorem 23 we derive that $(\mathbf{x}, 1, \mathbf{y})$ can be decomposed as a finite sum of "small solutions" $(\mathbf{u}, s, \mathbf{v})$ with $\sum_{j=1}^k \mathbf{u}(j) + s + \sum_{i=1}^e \mathbf{v}(i) \leq (2+m)^{k+1+e}$. Since the sum of those small solutions is 1 on the "s" component, exactly one of them is 1 on that component. This solution $(\mathbf{u}, 1, \mathbf{v})$ provides a vector $\mathbf{u}$ with $\sum_{j=1}^k \mathbf{u}(j) \leq (2+m)^{k+1+e}$ such that $M\mathbf{u} \geq \mathbf{b}$. ◀

We deduce a bound on minimal runs between two configurations.

▶ **Lemma 25.** *For every configurations $p(\mathbf{x})$ and $q(\mathbf{y})$ of a 2-TVASS $\mathcal{V} = (Q, \Sigma, \Delta)$ such that $p(\mathbf{x}) \xrightarrow{*} q(\mathbf{y})$, there exists a path π such that $p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$ and such that:*

$$|\pi| \leq (|Q| + \|\mathbf{x}\| + \|\mathbf{y}\| + \|\Sigma\|)^{O(|Q|^3)}$$

Proof. Let $c \geq 1$ be a constant satisfying Corollary 16, i.e., such that $O(1) \leq c$ and $O(|Q|^3) \leq c|Q|^3$. Consider a 2-TVASS $\mathcal{V}$ and let $p(\mathbf{x})$ and $q(\mathbf{y})$ be two configurations such that $p(\mathbf{x}) \xrightarrow{*} q(\mathbf{y})$. The case where $\Sigma \subseteq \{\mathbf{0}, \mathtt{tst}\}$ is trivial (there is a run of length at most $|Q|$ in that case), so we assume that $\|\Sigma\| \geq 1$ for the remainder of the proof. Let

us introduce $N = |Q| + \|\Sigma\|$. Corollary 16 shows that there exists a linear path scheme $L = \alpha_0\beta_1^*\alpha_1 \cdots \beta_k^*\alpha_k$ with $p(\mathbf{x}) \xrightarrow{L} q(\mathbf{y})$ and such that $|L| \leq N^c$ and $k \leq c|Q|^3$. It follows that there exists $n_1, \ldots, n_k \in \mathbb{N}$ such that:

$$p(\mathbf{x}) \xrightarrow{\alpha_0\beta_1^{n_1}\alpha_1\cdots\beta_k^{n_k}\alpha_k} q(\mathbf{y})$$

By removing from L the cycles β_j such that $n_j = 0$, we can assume, without loss of generality, that $n_j \geq 1$ for every $j \in [1, k]$. It follows that L is feasible. From Corollary 21 we deduce that $(n_1, \ldots, n_k)$ is a solution of $S_{\mathbf{x},L,\mathbf{y}}$. From Corollary 24 we deduce that there exist $m_1, \ldots, m_k \in \mathbb{N}$ such that $(m_1, \ldots, m_k)$ satisfies $S_{\mathbf{x},L,\mathbf{y}}$ and such that $m_1 + \cdots + m_k \leq (2+m)^{k+1+e}$ where $m \leq \|\mathbf{x}\| + \|\mathbf{y}\| + \|\Sigma\| \cdot |L|$ and $e \stackrel{\text{def}}{=} 9k+7$. This expression for e comes from the encoding of $\geq_1$ with 3 inequalities. Let us introduce the path $\pi = \alpha_0\beta_1^{m_1}\alpha_1 \cdots \beta_k^{m_k}\alpha_k$ and observe that $p(\mathbf{x}) \xrightarrow{\pi} q(\mathbf{y})$ from Corollary 21. Moreover $|\pi|$ is bounded by:

$$(2+m)^{10k+8} \cdot |L| \leq (2 + \|\mathbf{x}\| + \|\mathbf{y}\| + \|\Sigma\| N^c)^{10c|Q|^3+8} N^c \leq (|Q| + \|\mathbf{x}\| + \|\mathbf{y}\| + \|\Sigma\|)^{O(|Q|^3)}$$

This concludes the proof of the lemma. ◀

We are now ready to characterize the complexity of the reachability problem in 2-TVASS. This decision problem asks, given a 2-TVASS $\mathcal{V} = (Q, \Sigma, \Delta)$ and two configurations $p(\mathbf{x})$ and $q(\mathbf{y})$, whether $p(\mathbf{x}) \xrightarrow{*} q(\mathbf{y})$. By Lemma 25, if $p(\mathbf{x}) \xrightarrow{*} q(\mathbf{y})$ then there exists a run from $p(\mathbf{x})$ to $q(\mathbf{y})$ of length at most exponential in the sizes of $\mathcal{V}$, $p(\mathbf{x})$ and $q(\mathbf{y})$. Notice that configurations along that run have a polynomial size (with respect to the size of the input problem). It follows that a polynomial-space bounded exploration of the reachability set provides a way to decide the reachability problem. We have shown the following theorem.

▶ **Theorem 26.** *The reachability problem for 2-TVASS is PSPACE-complete.*

▶ Remark 27. Other natural problems on 2-TVASS are PSPACE-complete. In the full version of the paper [21, Appendix D], we derive from the succinct flattenability of 2-TVASS that the boundedness problem and the termination problem are both decidable in polynomial space. These results are obtained by providing a polynomial bound on the size of reachable configurations of a bounded 2-TVASS.

7 Conclusion and Perspectives

We have shown in this paper that extending 2-VASS with zero-tests on the first counter is for free, in the sense that the reachability problem remains PSPACE-complete (and so do the boundedness and termination problems). As in the case of 2-VASS, a crucial step in our approach is what we call *succinct flattenability*, i.e., the existence of small linear path schemes witnessing flattenability. Succinct flattenability of 2-VASS was leveraged by Englert et al. in [7] to show that reachability in 2-VASS is NL-complete when the input integers are encoded in unary. The question whether reachability in unary 2-TVASS remains NL-complete is left open. We conjecture that this question can be answered positively, by leveraging our succinct flattenability result for 2-TVASS and by extending [7] with zero-tests on the first counter.

References

1 Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in two-dimensional vector addition systems with states is PSPACE-complete. In *LICS*, pages 32–43. IEEE, 2015.

2 Rémi Bonnet. The reachability problem for vector addition system with one zero-test. In *MFCS*, volume 6907 of *LNCS*, pages 145–157. Springer, 2011.

3 I. Borosh and L. B. Treybig. A sharp bound on positive solutions of linear diophantine equations. *SIAM J. Matrix Analysis Applications*, 13(2):454–458, 1992.

4 Wojciech Czerwiński, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. In *STOC*, pages 24–33. ACM, 2019.

5 Wojciech Czerwiński, Sławomir Lasota, Christof Löding, and Radoslaw Piórkowski. New pumping technique for 2-dimensional VASS. In *MFCS*, volume 138 of *LIPIcs*, pages 62:1–62:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

6 Catherine Dufourd, Alain Finkel, and Philippe Schnoebelen. Reset nets between decidability and undecidability. In *ICALP*, volume 1443 of *LNCS*, pages 103–115. Springer, 1998.

7 Matthias Englert, Ranko Lazić, and Patrick Totzke. Reachability in two-dimensional unary vector addition systems with states is NL-complete. In *LICS*, pages 477–484. ACM, 2016.

8 John Fearnley and Marcin Jurdzinski. Reachability in two-clock timed automata is PSPACE-complete. *Inform. Comput.*, 243:26–36, 2015.

9 Alain Finkel, Jérôme Leroux, and Grégoire Sutre. Reachability for two-counter machines with one test and one reset. In *FSTTCS*, volume 122 of *LIPIcs*, pages 31:1–31:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

10 Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In *CONCUR*, volume 5710 of *LNCS*, pages 369–383. Springer, 2009.

11 John Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8(2):135–159, 1979.

12 S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *STOC*, pages 267–281. ACM, 1982.

13 Jean-Luc Lambert. A structure to decide reachability in Petri nets. *Theor. Comput. Sci.*, 99(1):79–104, 1992.

14 Michel Latteux. Langages à un compteur. *J. Comput. Syst. Sci.*, 26(1):14–33, 1983.

15 Jérôme Leroux. The general vector addition system reachability problem by Presburger inductive invariants. *Logical Methods in Computer Science*, 6(3), 2010.

16 Jérôme Leroux. Vector addition system reachability problem: a short self-contained proof. In *POPL*, pages 307–316. ACM, 2011.

17 Jérôme Leroux. Vector addition systems reachability problem (A simpler solution). In *Turing-100*, volume 10 of *EPiC Series in Computing*, pages 214–228. EasyChair, 2012.

18 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *LICS*, pages 1–13. IEEE, 2019.

19 Jérôme Leroux and Grégoire Sutre. On flatness for 2-dimensional vector addition systems with states. In *CONCUR*, volume 3170 of *LNCS*, pages 402–416. Springer, 2004.

20 Jérôme Leroux and Grégoire Sutre. Flat counter automata almost everywhere! In *ATVA*, volume 3707 of *LNCS*, pages 489–503. Springer, 2005.

21 Jérôme Leroux and Grégoire Sutre. Reachability in two-dimensional vector addition systems with states: One test is for free, 2020. `arXiv:2007.09096`.

22 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On the coverability problem for pushdown vector addition systems in one dimension. In *ICALP (2)*, volume 9135 of *LNCS*, pages 324–336. Springer, 2015.

23 Richard J. Lipton. The reachability problem requires exponential space. Technical Report 62, Yale University, 1976. URL: `http://cpsc.yale.edu/sites/default/files/files/tr63.pdf`.

24 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *STOC*, pages 238–246. ACM, 1981.

25 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.

26 Marvin L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.

27 Loic Pottier. Minimal solutions of linear diophantine systems: Bounds and algorithms. In *RTA*, volume 488 of *LNCS*, pages 162–173. Springer, 1991.

28 Klaus Reinhardt. Reachability in Petri nets with inhibitor arcs. *Electr. Notes Theor. Comput. Sci.*, 223:239–264, 2008.

29 Philippe Schnoebelen. Revisiting Ackermann-hardness for lossy counter machines and reset Petri nets. In *MFCS*, volume 6281 of *LNCS*, pages 616–628. Springer, 2010.

30 Leslie G. Valiant and Mike Paterson. Deterministic one-counter automata. *J. Comput. Syst. Sci.*, 10(3):340–350, 1975.

Coverability in 1-VASS with Disequality Tests

Shaull Almagor
Technion – Israel Institute of Technology, Haifa, Israel

Nathann Cohen
CNRS & LRI, Gif-sur-Yvette, France

Guillermo A. Pérez
University of Antwerp, Belgium

Mahsa Shirmohammadi
CNRS & IRIF, Université de Paris, France

James Worrell
University of Oxford, UK

Abstract

We study a class of reachability problems in weighted graphs with constraints on the accumulated weight of paths. The problems we study can equivalently be formulated in the model of vector addition systems with states (VASS). We consider a version of the vertex-to-vertex reachability problem in which the accumulated weight of a path is required always to be non-negative. This is equivalent to the so-called control-state reachability problem (also called the coverability problem) for 1-dimensional VASS. We show that this problem lies in NC: the class of problems solvable in polylogarithmic parallel time. In our main result we generalise the problem to allow disequality constraints on edges (i.e., we allow edges to be disabled if the accumulated weight is equal to a specific value). We show that in this case the vertex-to-vertex reachability problem is solvable in polynomial time even though a shortest path may have exponential length. In the language of VASS this means that control-state reachability is in polynomial time for 1-dimensional VASS with disequality tests.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Reachability, Vector addition systems with states, Weighted graphs

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.38

Funding *Shaull Almagor*: has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 837327.
James Worrell: Supported by EPSRC Fellowship EP/N008197/1.

Acknowledgements We thank P. Offtermatt for pointing us to literature on NC-algorithms.

1 Introduction

In this paper we study reachability problems in weighted graphs with constraints on the accumulated weight along a path. We show that the vertex-to-vertex reachability problem is in NC if the constraint is that the accumulated weight must always be non-negative, and the problem is in polynomial time if we additonally allow disequality constraints on edges (i.e., constraints that prevent an edge from being taken in a path if the accumulated weight prior to taking the edge is equal to a specific value). In both cases a shortest path satisfying the constraints may have length exponential in the problem description. Several related problems have been studied in the literature, including the problem of finding a path from a source vertex to target vertex that has a specific total weight [12].

The problems we study can naturally be formalised as reachability problems for types of one-counter machines, and the majority of the related work has been presented in this context. Under this correspondence, the value of the counter represents the accumulated weight along a path, and tests on the counter encode constraints on allowable paths. Algorithmic properties of one-counter machines have been studied by many authors over several decades [2, 4, 6, 7, 8, 9, 10, 11]. The above references are a small subset of the extensive literature on one-counter machines, but they well illustrate that there are many variations on the basic model and that these variations can lead to the model having substantially different algorithmic properties. Particular features mentioned in the references above, driven by applications to automated verification and program analysis, include equality tests, disequality tests, inequality tests, parametric tests, binary updates, polynomial updates, and parametric updates.

Analysing the complexity of reachability in the presence of the features listed above leads to a rich complexity landscape. It is shown in [11] that control-state reachability is decidable in NL for a "plain vanilla" model of one-counters machine – namely with a counter taking values in the nonnegative integers with operations increment, decrement, and zero testing. Thinking of one-counter machines as one-dimensional vector addition systems with states (1-VASS), it is natural to allow the counter to be updated by adding integer constants in binary. In this case, still with equality tests, control-state reachability becomes NP-complete [10]. The NP upper bound here is non-trivial since, due to the binary encoding of integers, a computation that reaches the goal state may have length exponential in the size of the machine. If one enriches the model further by introducing inequality tests (comparing the counter with an integer constant) then control-state reachability becomes PSPACE-complete [7]. A model of intermediate complexity is one with equality and disequality tests (introduced in [6], with applications to temporal-logic model checking). In this case the complexity of control-state reachability is open (between NP and PSPACE).

In this paper we consider 1-VASS with disequality tests, but no equality tests. In terms of 1-VASS, our main result states that the control-state reachability problem is solvable in polynomial time for 1-VASS with disequality tests. This result confirms the intuition that disequality tests are weaker than equality tests. The main technical challenge to obtaining a polynomial-time bound is that a run witnessing that a given control state is reachable may have length exponential in the description of the counter machine. A standard way to overcome this obstacle in related settings is to show that one may restrict attention to computations that fit a regular pattern (usually in terms of iterating a "small" number of cycles). Here the presence of disequality tests proves to be surprisingly disruptive: it destroys the monotonicity of the transition relation and prevents from freely iterating positive-weight cycles. (For example, the lack of monotonicity means that it is coNP hard to determine whether, given a control state s_0, for all counter values $u \in \mathbb{N}$ the configuration (s_0, u) is unbounded, i.e., can reach infinitely many configurations – see Figure 1 – whereas the same problem for 1-VASS without tests is easily seen to be decidable in polynomial time.) Resolving the complexity of reachability for 1-VASS with both equality and disequality tests remains open. We hope that the techniques developed here can help solve this challenging problem.

To complement our main result, we show that for 1-VASS without tests control-state reachability (and hence also boundedness) is decidable in NC, i.e., the subclass of P consisting of problems solvable in polylogarithmic parallel time. Problems in NC are in particular solvable in polylogarithmic space. Related to this, Rosier and Yen [16] have shown that boundedness for VASS is NL-complete in case there are absolute bounds on the dimension and bit-size of integer vectors.

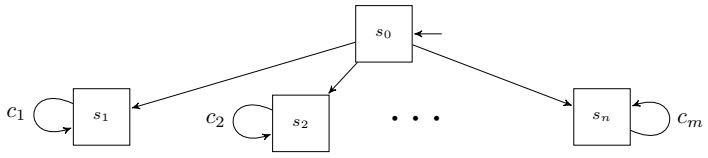


Figure 1 A 1-VASS with disequality tests, derived from a 3-CNF formula φ having propositional variables $X_1, \ldots, X_m$ and clauses $C_1, \ldots, C_n$. We have states $s_1, \ldots, s_n$ – one state for each clause – and an initial state s_0. The reduction is such that (s_0, u) is unbounded for all $u \in \mathbb{N}$ iff φ is unsatisfiable. Let $p_1, \ldots, p_m$ be the first m primes and write $P := p_1 \cdots p_m$ for their product. For all $u \in \mathbb{N}$, define the propositional assignment $\mathrm{val}_u : \{X_1, \ldots, X_m\} \to \{0, 1\}$ by $\mathrm{val}_u(X_i) = 1$ if and only if $p_i \mid u$. Suppose that state s corresponds to a clause C that mentions variables $X_{i_1}, X_{i_2}, X_{i_3}$. Then we place a self-loop on s with increment $c_i := p_{i_1} p_{i_2} p_{i_3}$ and add disequality tests on s (or equivalently on the self-loop on s) for all those values $u \in \{P, P+1, \ldots, P + p_{i_1} p_{i_2} p_{i_3} - 1\}$ where the assignment val_u satisfies the clause C. Given $u \in \{0, 1, \ldots, P-1\}$, observe that the configuration (s_0, u) is bounded iff val_u satisfies φ (see Appendix A for a complete proof).

2 Definitions

We write $\mathbb{N}$ to denote the set of all nonnegative integers $0, 1, 2, \ldots$ In presenting our results we assume familiarity of the reader with basic graph theory and computational complexity.

One-Dimensional Vector Addition Systems with States and Tests. A *1-VASS with disequality tests* is a tuple $\mathcal{V} = (Q, D, \Delta, w)$, where Q is a set of *states*, $D = \{D_q\}_{q \in Q}$ is a collection of cofinite subsets $D_q \subseteq \mathbb{N}$, $\Delta \subseteq Q \times Q$ is a set of *transitions*, and $w : \Delta \to \mathbb{Z}$ is a function that assigns an integer weight to each transition. In the special case that each D_q equals $\mathbb{N}$, we simply call $\mathcal{V}$ a 1-VASS (and we omit the collection D).

A *configuration* of $\mathcal{V}$ is a pair (q, z) comprising a state $q \in Q$ and a nonnegative integer $z \in \mathbb{N}$ referred to as the *counter value*. We write *Conf* for the set $Q \times \mathbb{N}$ of all configurations. We define a partial order on *Conf* by $(q, z) \leq (q', z')$ if and only if $q = q'$ and $z \leq z'$. A configuration (q, z) is *valid* if $z \in D_q$.

A *path* in $\mathcal{V}$ is a sequence of states $\pi = q_1, \ldots, q_n$ such that $(q_i, q_{i+1}) \in \Delta$ for all $i \in \{1, \ldots, n-1\}$. We sometimes refer to such a path as a q_1-q_n path. Let $\pi' = p_1, p_2, \ldots, p_m$ be another path such that $q_n = p_1$, we define $\pi_1 \cdot \pi_2 := q_1, \ldots, q_n, p_2, \ldots, p_m$. Given states p, q, r, a set P of p-q paths, and a set R of q-r paths, we define $P \cdot R := \{\pi \cdot \pi' \mid \pi \in P, \pi' \in R\}$. The weight of π is defined to be $\mathrm{weight}(\pi) := \sum_{i=1}^{n-1} w(q_i, q_{i+1})$. A (possibly empty) prefix of π is said to be *minimal* if it has minimal weight among all prefixes of π. Define $\mathrm{pmin}(\pi)$ to be the weight of a minimal prefix of π.

A *run* is a sequence $(q_1, z_1), \ldots, (q_n, z_n)$ of configurations of $\mathcal{V}$ such that there is a path $\pi = q_1, \ldots, q_n$ with $z_{i+1} = z_i + w(q_i, q_{i+1})$ for $i = 1, \ldots, n-1$. We write $(q_1, z_1) \xrightarrow{\pi} (q_n, z_n)$ to denote such a run. Observe that runs are not allowed to reach negative counter values. A *valid run* is a run whose configurations are all valid. Intuitively, a valid run through q can proceed if and only if the current counter value is in D_q. We say that a configuration (q', z') is *reachable* from (q, z) if there is a valid run π such that $(q, z) \xrightarrow{\pi} (q', z')$.

In computational problems all numbers in the description of $\mathcal{V}$ are given in binary. Given a state q we represent the cofinite set D_q as the complement of an explicitly given subset of $\mathbb{N}$. Given this convention, we can assume without loss of generality that for all states q the set D_q is either $\mathbb{N}$ or $\mathbb{N} \setminus \{g\}$ for some $g \in \mathbb{N}$; see Appendix B. For states q with $D_q = \mathbb{N} \setminus \{g\}$, we refer to the single missing value g in the domain as the *disequality guard* on q.

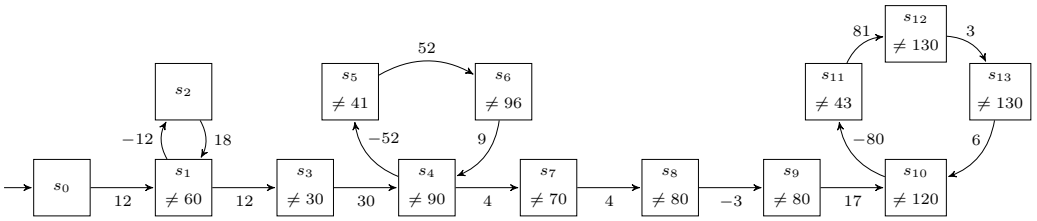


Figure 2 A 1-VASS with disequality tests. Disequality guards are denoted by $\neq$. For example, in state s_1 the set D_{s_1} is $\mathbb{N} \setminus \{60\}$, and no run goes through s_1 if its current counter value is 60.

The Coverability and Unboundedness Problems. Let $\mathcal{V} = (Q, \Delta, D, w)$ be a 1-VASS with disequality tests, and let s and t be two distinguished states of $\mathcal{V}$. The *Coverability Problem* asks whether there exists a valid run in $\mathcal{V}$ from $(s, 0)$ to (t, z) for some $z \in \mathbb{N}$ (in which case we say that $(s, 0)$ can *cover* t). The *Unboundedness Problem* asks whether the set of configurations reachable from $(s, 0)$ is infinite (in which case we say that $(s, 0)$ is *unbounded*).

The Coverability problem reduces to the Unboundedness problem by, intuitively, forcing $(t, 0)$ to be unbounded using a positive cycle, and removing all states that cannot reach t in the underlying graph of $\mathcal{V}$. In fact, the following holds.

▶ **Lemma 1.** *There is an* NC^2*-computable many-one reduction from the Coverability Problem to the Unboundedness Problem.*

Henceforth, we focus on the complexity of deciding the Unboundedness Problem. In Section 3 we prove that the Unboundedness Problem for 1-VASS with disequality tests is decidable in polynomial time. Since $\mathsf{NC}^2 \subseteq \mathsf{P}$, by Lemma 1 we also have that the Coverability Problem in this setting is decidable in polynomial time. In Section 4 we prove that the Unboundedness Problem for 1-VASS (without disequality tests) is in NC^2, and we deduce that the Coverability Problem for 1-VASS is decidable in NC^2.

3 Unboundedness for 1-VASS with Disequality Tests

Fix a 1-VASS $\mathcal{V} = (Q, D, \Delta, w)$ with disequality tests and a distinguished state $s \in Q$. We are interested in determining whether the configuration $(s, 0)$ is unbounded.

For a (possibly infinite) path $\pi = q_1, q_2, \ldots$, denote by $\mathrm{blocked}(\pi)$ the set of $z \in \mathbb{N}$ such that the unique induced run from (q, z) either contains a negative counter value or violates a disequality guard. That is, π does not lift to a valid run from the configuration (q_1, z).

▶ **Example 2.** In Figure 2, since 41 is the guard on s_5 the run $(s_4, 93), (s_5, 41), (s_6, 93)$ is not valid and $93 \in \mathrm{blocked}(s_4, s_5, s_6)$. Observe that $\mathrm{blocked}(s_4, s_5, s_6) = [0, 52) \cup \{90, 93, 96\}$ and $\mathrm{blocked}((s_4, s_5, s_6)^\omega) = [0, 52) \cup \{52 \leq z \leq 96 \mid z \equiv 0, 3, 6 \pmod 9\}$.

Recall that for a path π, $\mathrm{pmin}(\pi)$ is the weight of a minimum-weight prefix of π. Let $Q_+ \subseteq Q$ be the set of states $q \in Q$ such that there is a positive-weight simple cycle on q in the underlying graph of $\mathcal{V}$. For $q \in Q_+$ we pick a simple cycle γ_q such that $\mathrm{pmin}(\gamma_q) \geq \mathrm{pmin}(\gamma)$ for any other positive-weight simple cycle γ on q; write W_q for $\mathrm{weight}(\gamma_q)$.[1] Define $\mathit{Conf}_+ := \{(q, z) \in \mathit{Conf} \mid q \in Q_+,\ z + \mathrm{pmin}(\gamma_q) \geq 0\}$.

Define a path to be *primitive* if no proper infix is a positive cycle (note though that a primitive path may itself be a positive cycle). We say that a run is primitive if the underlying path is primitive. Observe that if ρ is a valid run, none of whose internal configurations (i.e. excluding the first and last configurations) lies in Conf_+, then ρ is primitive.

[1] Note that γ_q does not necessarily have maximal weight W_q among the positive simple cycles on q.

▶ **Example 3.** In Figure 2, for $s_1 \in Q_+$ we pick the simple cycle $\gamma_{s_1} = s_1, s_2, s_1$ with $W_{s_1} = 6$. Since $\mathrm{pmin}(\gamma_{s_1}) = -12$, we have that $\{z \mid (s_1, z) \in \mathit{Conf}_+\} = [12, \infty)$. Moreover, the path s_4, s_5, s_6, s_4 is primitive, but s_1, s_2, s_1, s_3 is not primitive.

▶ **Proposition 4.** *A configuration $(s, 0)$ is unbounded if, and only if, $(s, 0)$ can reach an unbounded configuration in Conf_+.*

In order to decide whether $(s, 0)$ is unbounded, by Proposition 4, it suffices to compute the set of unbounded configurations in Conf_+ and determine whether $(s, 0)$ can reach this set. Define $\mathit{Conf}_\infty \subseteq \mathit{Conf}_+$ to be the set of all unbounded configurations in Conf_+. Observe that every configuration $(q, z) \in \mathit{Conf}_+$ with $z \notin \mathrm{blocked}(\gamma_q^\omega)$ can take the cycle γ_q arbitrarily many times and is thus included in Conf_∞. However, even if $z \in \mathrm{blocked}(\gamma_q^\omega)$, it may still be the case that (q, z) is unbounded, by traversing more complicated paths.

▶ **Example 5.** In Figure 2, all configurations (s_4, z) with z in $\mathbb{N} \setminus \mathrm{blocked}((s_4, s_5, s_6)^\omega) = \{52 \leq z \leq 96 \mid z \not\equiv 0, 3, 6 \pmod 9\} \cup (96, \infty)$ are trivially unbounded and thus included in Conf_∞. It will transpire that $\{s_4\} \times \{54, 60, 63, 69\} \subseteq \mathit{Conf}_\infty$ even though $\{54, 60, 63, 69\} \in \mathrm{blocked}((s_4, s_5, s_6)^\omega)$.

In order to reason about the aforementioned complicated paths, we proceed as follows. In Section 3.1 we introduce residue classes and chains, which form a partition of Conf_+, and are the building blocks of our analysis. In Section 3.2 we characterize Conf_∞ as the limit of an inductive construction. This enables us to reason about the structure of Conf_∞ in Section 3.3. Finally, in Section 3.4 we show how to compute Conf_∞ and decide unboundedness.

3.1 Residue Classes and Chains

Given $q \in Q_+$ and $0 \leq r < W_q$, we call the set of configurations $\{(q, z) \in \mathit{Conf}_+ \mid z \equiv r \pmod{W_q}\}$ a *q-residue class*. We simply speak of a *residue class* if we do not want to specify the state q. Given a q-residue class R, a set $C \subseteq R$ is called a *q-chain* if it is a maximal subset of R for the property that every pair of configurations $(q, z), (q, z') \in C$ with $z < z'$ is connected by a valid run obtained by iterating the cycle γ_q. Again, we speak of a *chain* if we do not want to specify the state q.

We draw a distinction between *bounded chains* and *unbounded chains*, where a chain is bounded if and only if the associated set of counter values is bounded. An unbounded q-chain C is contained in Conf_∞ since the cycle γ_q can be taken arbitrarily many times from any configuration in C to yield a valid run.

▶ Remark 6. Let us write $\gamma_q = q_1, q_2, \ldots$ For each q_1-residue class R, every z such that $z + \mathrm{weight}(q_1, \ldots, q_i) \notin D_{q_i}$, for some q_i, induces at most two bounded chains. Namely, the set of configurations below (q, z) form a chain; and the singleton $\{(q, z)\}$ is also (vacuously) a chain. Note that every residue class also has one unbounded chain. That is, the set of configurations above (q, z) with z the maximal "induced guard" on q. Since there are at most $|Q|$ guards, each residue class decomposes as a disjoint union of at most $2|Q|$ bounded chains and a single unbounded chain.

Intuitively, within each bounded chain we can iterate the cycle γ_q until hitting a guard. We call a residue class R *trivial* if it consists solely of a single unbounded chain. Note that the union of all bounded q-chains is equal to $\mathit{Conf}_+ \cap \{q\} \times \mathrm{blocked}(\gamma_q^\omega)$.

▶ **Example 7.** As indicated in Figure 3 for the running example, the residue classes $\{s_4\} \times (52 + i + 9\mathbb{N})$ with $i \in \{0, 1, 3, 4, 6, 7\}$ are indeed trivial, while each residue class $\{s_4\} \times (52 + i + 9\mathbb{N})$ with $i \in \{2, 5, 8\}$ consists of two bounded chains $\{s_4\} \times \{52 \leq z < 88 + i \mid z \equiv i \pmod 9\}$ and $\{s_4\} \times \{88 + i\}$, and a single unbounded chain $\{s_4\} \times (88 + i + 9\mathbb{N})$.

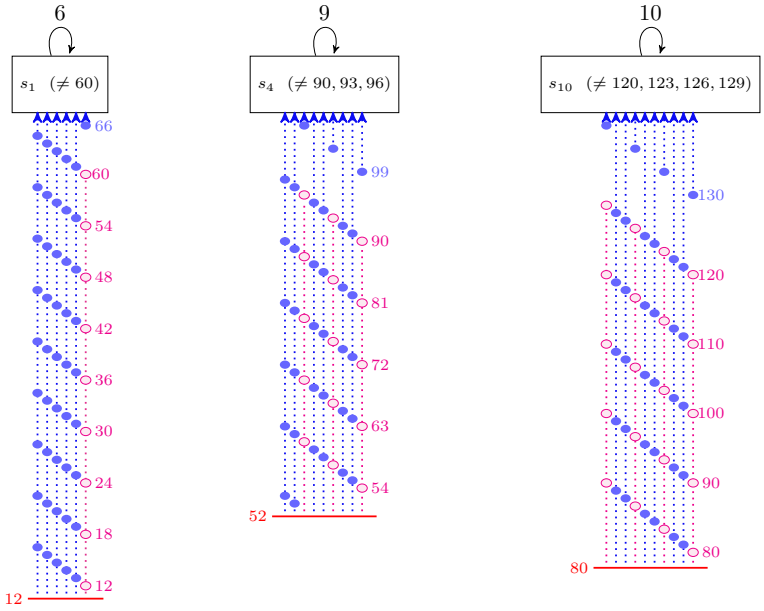


Figure 3 We focus on states s_1, s_4, and s_{10} in the 1-VASS in Figure 2, each of which lies on a simple positive cycle. We also indicate which counter values prevent taking the associated positive cycle. For example, state s_4 has the simple cycle γ_{s_4} with $W_{s_4} = 9$ and taking γ_{s_4} from $\{s_4\} \times \{90, 93, 96\}$ is not allowed due to disequality guards along γ_{s_4}. The columns underneath each state represent residue classes of that state in $Conf_+$. We colour all unbounded chains in blue and all bounded chains in pink; thus all blue configurations form the set U_0.

One of the main ideas in this section is to show that a configuration is unbounded if and only if it can reach an unbounded chain via a valid run whose underlying path π has the form

$$\pi = \pi_0 \cdot \gamma_{q_1}^{n_1} \cdot \pi_1 \cdots \pi_{k-1} \cdot \gamma_{q_k}^{n_k} \cdot \pi_k \,,$$

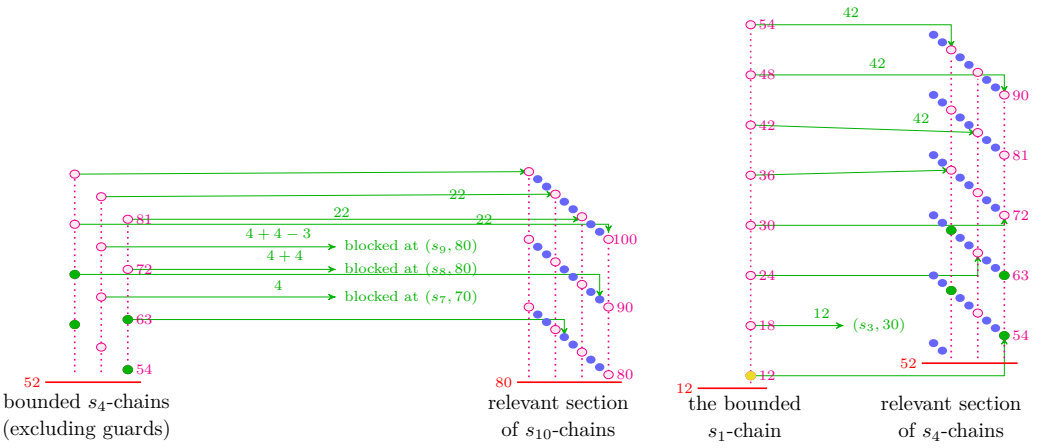
where $\pi_0, \ldots, \pi_k$ are primitive paths and $n_1, \ldots, n_k$ are non-negative integers. Moreover, we give a polynomial bound on the length of the π_i and the magnitude of k in terms of the size of the underlying 1-VASS (in general, the exponents n_i may be exponential in the size of the 1-VASS). We also show how to detect the existence of such a path in polynomial time.

Recall the structure of $Conf$ as a partially ordered set. We will use standard order-theoretic terminology and notation to refer to sets of configurations: in particular given sets of configurations $S, S' \subseteq Conf$, we say that S is *downward closed in* S' if for all $(q, z) \in S \cap S'$ and $(q, z') \in S'$ with $z' \leq z$, we have $(q, z') \in S$.

3.2 Inductive Characterization of $Conf_\infty$

We now give an inductive backward-reachability construction of the set of all configurations in $Conf_+$ that can reach an unbounded chain. Since unbounded configurations can, in particular, reach unbounded chains (as above the maximal disequality guard, all chains are unbounded), this set is exactly $Conf_\infty$.

In order for our inductive construction to converge in a polynomial number of steps, we essentially consider meta-transitions of the form $\gamma_q^k \cdot \pi$ for γ_q a simple cycle, $k \in \mathbb{N}$, and π a primitive path. Formally, we define an increasing sequence $U_0 \subseteq U_1 \subseteq U_2 \subseteq \cdots$ of subsets of $Conf_+$ such that $\bigcup_{n \in \mathbb{N}} U_n = Conf_\infty$. Define U_0 to be the union of the collection



(a) The set U_1 is obtained from U_0 in Figure 3.

(b) The set U_2.

Figure 4 The sets U_1 and U_2 of the running example. The blue configurations are in U_0; green ones are in $U_1 \setminus U_0$; yellow one is in $U_2 \setminus U_1$. The pink configurations are in $Conf_+ \setminus U_1$ and $Conf_+ \setminus U_2$, respectively. While computing U_1, the green configurations $(s_4, 63)$ and $(s_4, 69)$ take the primitive path $\pi = s_4, s_7, s_8, s_9, s_{10}$ to U_0. In all other pink configurations in s_4-chains, although enabled, the path π either hits a guard or ends in $(s_{10}, z) \in Conf_+ \setminus U_1$.

of unbounded chains. Given $n \in \mathbb{N}$ we inductively construct U_{n+1} as follows. First, define $U'_n \subseteq Conf_+$ as the set of configurations $(q, z) \notin U_n$ whose distance to U_n is minimal among all configurations in $Conf_+ \setminus U_n$ (here the distance of a configuration (q, z) to U_n is the length of the shortest valid run from (q, z) to U_n). Now define $U_{n+1} \subseteq Conf_+$ to be the smallest set such that $U_n, U'_n \subseteq U_{n+1}$ and $U_{n+1} \cap C$ is downward closed in every chain C. Then $\bigcup_{n \in \mathbb{N}} U_n$ is the set of configurations in $Conf_+$ that can reach an unbounded chain which, as noted above, is equal to $Conf_\infty$.

▶ Remark 8. By definition, a shortest run from a configuration $(q, z) \in U'_{n+1} \setminus U_n$ to U_n has no internal configurations in $Conf_+$, and is therefore primitive.

▶ **Example 9.** Figure 3 indicates the set U_0 for the running example. Note that U_0 contains all trivial residue classes. Observe that $U'_0 = \{(s_4, 63), (s_4, 69)\}$; see Figure 4a. These two configurations belong to two distinct chains. The downward closure of $\{(s_4, 63)\}$ in its chain is $\{s_4\} \times \{54, 63\}$, and the downward closure of $\{(s_4, 69)\}$ in its chain is $\{s_4\} \times \{60, 69\}$. We have that $U_1 = U_0 \cup (\{s_4\} \times \{54, 60, 63, 69\})$. The second iteration to compute U_2 only adds the configuration $(s_1, 12)$ to U_1; see Figure 4b. The sequence stabilizes in this iteration.

3.3 The Structure of $Conf_\infty$

In this section we analyze the structure of $Conf_\infty$, based on its inductive characterization. This analysis will be key in obtaining a polynomial-time algorithm to compute $Conf_\infty$.

The guiding intuition is that for all n the set U_n is *almost upward closed* in each residue class R. By this we mean that if (q, z) is the least configuration in $R \cap U_n$, then all but polynomially many configurations of R above (q, z) are also in U_n. More specifically, we show that for any bounded chain C in R that lies above (q, z), although the number of configurations in C may be exponential in $|Q|$, the size of $C \setminus U_n$ is bounded by a polynomial in $|Q|$. (Note here that the unique unbounded chain in R is contained in U_0 and hence is contained in U_n for all $n \in \mathbb{N}$.) Using this observation, we provide a polynomial bound on the

number of iterations until the inductive construction converges. Indeed, in every iteration, unless a fixed point has been reached, there must exist some bounded chain C such that the size of $C \setminus U_n$ strictly decreases. After showing that $C \setminus U_n$ is of polynomial size, we obtain a polynomial bound on the number of iterations until U_n converges by Remark 6.

We start by characterizing the paths between chains.

▶ **Proposition 10.** *Let $(q,z),(q',z') \in Conf_+$ and let $(q,z) \xrightarrow{\pi} (q',z')$ be a (not necessarily valid) run such that π is a primitive path. Then there exists a run $(q,z) \xrightarrow{\pi'} (q',z'')$ of length at most $|Q|^2+2$ such that*

1. $\mathrm{pmin}(\pi') \geq \mathrm{pmin}(\pi)$,
2. $z'' \geq z'$, *and*
3. *the q'-residue class of (q',z'') is either trivial or identical to that of (q',z').*

Given a q-residue class R, in general U_n is not an upward closed subset of R. The following definitions are intended to measure the defect of U_n in this regard.

We say that a bounded chain C that is contained in a residue class R is *n-active* if there exists a configuration in $U_n \cap R$ that lies below some configuration in C. Let C be an n-active chain. Recall that U_n is downward closed in C and hence $C \setminus U_n$ is upward closed in C. Suppose that $C \setminus U_n$ is non-empty, write $m_1 := \min\{x : (q,x) \in C \setminus U_n\}$ and $m_2 := \max\{x : (q,x) \in C \setminus U_n\}$, and define[2]

$$\delta_n(C) := \{(q,x) \in Conf_+ : m_1 \leq x \leq m_2 \text{ and } (q,x) \notin U_n\}.$$

Thus $\delta_n(C)$ contains all configurations in $C \setminus U_n$, as well as all configurations "between" elements of $C \setminus U_n$, apart from those that are themselves in U_n. If $C \setminus U_n = \emptyset$ then we define $\delta_n(C) := \emptyset$. Finally for a residue class R we write

$$\delta_n(R) := \bigcup \{\delta_n(C) : C \subseteq R \text{ an } n\text{-active chain}\} \ . \tag{1}$$

For $(q, x_{\min})$ the least element in $R \cap U_n$ we have that $|\{(q,x) \in R \setminus U_n : x_{\min} \leq x\}| \leq |\delta_n(R)|$.

▶ **Example 11.** In Figure 4a consider the chain $C := \{s_4\} \times \{54, 63, 72, 81\}$, which is 1-active as $(s_4, 54) \in U_1$. Since $C \setminus U_1 = \{s_4\} \times \{72, 81\}$ we have that $\delta_1(C) = \{s_4\} \times \{72, 75, 78, 81\}$.

▶ **Lemma 12.** *For all $n \in \mathbb{N}$ and every chain C we have that $|\delta_n(C)| \leq |Q| \cdot |C \setminus U_n|$.*

We now come to the central technical part of the paper, controlling the growth of $\delta_n(R)$ as a function of n:

▶ **Lemma 13.** *There exists a polynomial poly_2 such that for each residue class R and all $n \in \mathbb{N}$ we have $|\delta_{n+1}(R)| \leq \max\{|\delta_n(R')| : R' \text{ a residue class}\} + \mathrm{poly}_2(|Q|)$ if R contains a chain that is $(n+1)$-active but not n-active.*

Before proceeding to prove Lemma 13, we demonstrate the underlying intuition. Consider a configuration $(q,z) \in R \cap U'_{n+1}$ that has a primitive path π to a configuration $(q',z') \in U_n$. To prove Lemma 13, we argue that π lifts to a valid run from a "dense" subset of configurations in $\{(q,z'') \in R : z'' \geq z\}$. There are two main cases in this argument based on whether one of the larger configurations in the chain induces a valid run ending in a trivial residue class.

[2] We omit q from the definition of $\delta_n(C)$ for brevity.

▶ **Example 14.** The first case occurs in obtaining U_1 from U_0 in the running example; see Figure 4a. Consider the chain $C := \{s_4\} \times \{54, 63, 72, 81\}$. The primitive path $s_4, s_7, s_8, s_9, s_{10}$ from the largest configuration $(s_4, 81)$ in C leads to a non-trivial s_{10}-residue class (out of U_0). However, one among the n-next largest configurations in C, for $n = |\text{blocked}(s_4, s_7, s_8, s_9, s_{10})| \cdot |Q|$, lifts to a valid run to a trivial s_{10}-residue class. In the example, this is the case for $(s_4, 63)$. The second case occurs in obtaining U_2 from U_1 in the running example; see Figure 4b. Consider the chain $C' := \{s_1\} \times \{12, 18, 24, \cdots, 54\}$. The primitive path s_1, s_3, s_4, from none of the configurations in this chain, ends in a trivial s_4-residue class. However, we provide a subtle argument to bound $|C' \setminus U_2|$ with $|\delta_1(C)| + \text{poly}_2(|Q|)$.

Proof of Lemma 13. Pick the minimal element $(q, z_0) \in R \cap U'_{n+1}$. Moreover, let $(q', z') \in U_n$ and $(q, z_0) \xrightarrow{\pi} (q', z')$ be such that π is a shortest run from (q, z_0) to U_n. By Remark 8, π is a primitive path.

By Proposition 10 there is a run $(q, z_0) \xrightarrow{\pi'} (q', z'')$, for some $z'' \geq z'$, such that π' has length at most $|Q|^2 + 2$, and the residue class R' of (q', z'') is either *trivial* or the same as the residue class of (q', z').

Note that we do not claim that $(q', z'') \in U_n$, nor that π' lifts to a valid run. In what follows we will argue that if there are more than some polynomial number of configurations above (q, z_0) in $C \setminus U'_{n+1}$, where C is an $(n+1)$-active chain of R, then π' does lift to a valid run from one of them. Moreover, the run leads to some configuration in the same residue class as (q', z') or to a trivial residue class. Observe that, intuitively, this means we "pump" γ_q before taking π' so if we wanted to reach the same residue class as (q', z') we would need some nonnegative integer c such that

$$z_0 + W_q \cdot c + \text{weight}(\pi') \equiv z_0 + \text{weight}(\pi') \pmod{W_{q'}}.$$

Based on this intuition, we now identify two cases according to the order of W_q in the group $\mathbb{Z}/\mathbb{Z}W_{q'}$ of integers modulo $W_{q'}$, which is $\frac{W_{q'}}{\gcd(W_q, W_{q'})}$. Recall that this quantity is the smallest integer $c \geq 1$ such that $W_q \cdot c \equiv 0 \pmod{W_{q'}}$.

Case (i): $\frac{W_{q'}}{\gcd(W_q, W_{q'})} > |Q|$. We first show that $|C \setminus U_{n+1}| \leq (|Q|^2 + 2)(|Q| + 1)$ for every $(n+1)$-active chain C in R.

Let C be an $(n+1)$-active chain of R and suppose for a contradiction that $|C \setminus U_{n+1}| > (|Q|^2 + 2)(|Q| + 1)$. Since C is $(n+1)$-active, for every configuration $(q, z) \in C \setminus U_{n+1}$ we have $z \geq z_0$. Further, since $\text{pmin}(\pi') + z_0 \geq 0$, π' can only be blocked on a configuration due to a violation of a disequality guard. Since the length of π' is at most $|Q|^2 + 2$, it follows that at most $|Q|^2 + 2$ elements of $C \setminus U_{n+1}$ lie in $\{q\} \times \text{blocked}(\pi')$.

Recall that $C \setminus U_{n+1}$ is upward closed in C, so by the assumption that $|C \setminus U_{n+1}| > (|Q|^2 + 2)(|Q| + 1)$, there exists a set $S := \{(q, z_1 + iW_q) : 0 \leq i \leq |Q|\}$ of $|Q| + 1$ "consecutive" elements of $C \setminus U_{n+1}$, for some z_1, such that no element of S lies in $\{q\} \times \text{blocked}(\pi')$. Then π' lifts to a valid run from each element of S. Moreover, since the order of W_q in $\mathbb{Z}/\mathbb{Z}W_{q'}$ is assumed to be greater than $|Q|$, the images of the elements of S, after following π', lie in pairwise distinct q'-residue classes. But the number of non-trivial q'-residue classes is at most $|Q|$ and hence some configuration in S has a run over π' to a trivial q'-residue class and hence to U_n. But then such a configuration lies in U_{n+1}, which is a contradiction.

We conclude that $|C \setminus U_{n+1}| \leq (|Q|^2 + 2)(|Q| + 1)$ for every $(n+1)$-active chain C in R. But then $|\delta_{n+1}(C)| \leq |Q|(|Q|^2 + 2)(|Q| + 1)$ by Lemma 12. Finally, since R comprises at most $2|Q|$ bounded chains by Remark 6, we have that $|\delta_{n+1}(R)| \leq 2|Q|^2(|Q|^2 + 2)(|Q| + 1)$.

Case (ii): $\frac{W_{q'}}{\gcd(W_q, W_{q'})} \leq |Q|$. For the residue classes R and R' as above, define an injective partial mapping $\Phi : \delta_{n+1}(R) \to \delta_n(R')$ by $\Phi(q,x) = (q',x')$ if and only if $x' = x + \text{weight}(\pi')$ and $(q',x') \in \delta_n(R')$. We will prove that Φ is defined on all but $\text{poly}_3(|Q|)$ many configurations in $\delta_{n+1}(R)$, for some polynomial poly_3, thereby showing that $|\delta_{n+1}(R)| \leq |\delta_n(R')| + \text{poly}_3(|Q|)$. To this end, it suffices to show that Φ is defined on all but $\text{poly}_4(|Q|)$ many configurations in $\delta_{n+1}(C)$ for every $(n+1)$-active chain C in R, for some polynomial poly_4.

Let C be an $(n+1)$-active chain in R and let $C_1, \ldots, C_s$ be a list, given in increasing order, of the chains in R' that are mapped into by Φ from some configuration in $\delta_{n+1}(C)$. Then $C_1, \ldots, C_s$ are all n-active (as they are above $(q',z') \in U_n$). For $i \in \{1, \ldots, s\}$, write $(q, x^{(i)}_{\min})$ for the minimum configuration in $\delta_{n+1}(C)$ that is mapped by Φ to C_i and write $(q, x^{(i)}_{\max})$ for the maximum configuration in $\delta_{n+1}(C)$ that is mapped to C_i. Then for each $i = 1, \ldots, s$, every configuration $(q,x) \in \delta_{n+1}(C)$ such that $x^{(i)}_{\min} \leq x \leq x^{(i)}_{\max}$ and $x \notin \times\text{blocked}(\pi')$ is mapped by Φ to $\delta_n(R')$. Thus, writing $(q, x_{\max})$ and $(q, x_{\min})$ respectively for maximum and minimum configurations in $\delta_{n+1}(C)$, we have that Φ is defined on all non-blocked elements of $\delta_{n+1}(C)$ lying outside the set below.

$$\left\{ (q,x) \in \delta_{n+1}(C) \;\middle|\; x \in \left(x^{(s)}_{\max}, x_{\max}\right] \cup \left[x_{\min}, x^{(1)}_{\min}\right) \cup \bigcup_{i=1}^{s-1} \left(x^{(i)}_{\max}, x^{(i+1)}_{\min}\right) \right\} \tag{2}$$

Since $\text{blocked}(\pi')$ contains at most $|Q|^2 + 2$ elements, it remains to prove that the set (2) has polynomial cardinality. We claim its size is at most $(2|Q|+1) \cdot \text{poly}_5(|Q|)$, for some polynomial poly_5. For this it will suffice to show that any sub-interval I of $\delta_{n+1}(C)$ of the form $\{(q,x) \in \delta_{n+1}(C) : a \leq x \leq b\}$, where $a, b \geq x_{\min}$, and such that it does not meet the domain of Φ, has cardinality at most $\text{poly}_5(|Q|)$. (Indeed, note that (2) is a union of at most $2|Q|+1$ such intervals since there are at most $2|Q|$ chains in R by Remark 6.)

Let $\text{poly}_6(x) := (x^2+2)(x+1)+1$. Since $\text{blocked}(\pi')$ has cardinality at most $|Q|^2+2$, if we take $\text{poly}_6(|Q|)$ consecutive elements of $C \setminus U_{n+1}$ then there are at least $|Q|+1$ consecutive elements that lie outside $\{q\} \times \text{blocked}(\pi')$ and at least one of these elements – say (q,x) – has a valid run over π' to the residue class R' by the assumption that $\frac{W_{q'}}{\gcd(W_q, W_{q'})} \leq |Q|$. Since $(q,x) \notin U_{n+1}$ we have that $(q', x + \text{weight}(\pi')) \notin U_n$ and hence (q,x) is in the domain of Φ. We conclude that any sequence of at least $\text{poly}_6(|Q|)$ consecutive elements of $C \setminus U_{n+1}$ meets the domain of Φ. Hence any sub-interval I, as defined above, contains at most $\text{poly}_6(|Q|)$ elements of $C \setminus U_{n+1}$ and, by Lemma 12, contains at most $|Q| \cdot \text{poly}_6(|Q|)$ elements in total. ◀

Proposition 15 follows from Lemma 13 by induction, as follows.

▶ **Proposition 15.** *There exists a polynomial* poly_1 *such that for each residue class* R *and all* $n \in \mathbb{N}$ *we have* $|\delta_n(R)| \leq \text{poly}_1(|Q|)$.

Proof. Let α_n be the number of chains in $Conf_+$ that are n-active. Since n-active chains are by definition bounded, we have that $\alpha_n \leq 2|Q|^2$ for all $n \in \mathbb{N}$ (see Remark 6). We argue by induction on n that $|\delta_n(R)| \leq \alpha_n \cdot \text{poly}_2(|Q|)$ for all $n \in \mathbb{N}$ and all residue classes R. We conclude that $|\delta_n(R)| \leq 2|Q|^2 \cdot \text{poly}_2(|Q|)$.

The base case is trivial as there are no 0-active chains and $\delta_0(R)$ is empty for all residue classes. The induction step has two cases. First, suppose that $\alpha_{n+1} = \alpha_n$, i.e., all chains in $Conf_+$ that are $(n+1)$-active were already n-active. Since $U_n \subseteq U_{n+1}$, we have that $\delta_{n+1}(C) \subseteq \delta_n(C)$ for all chains C in R. We conclude that $\delta_{n+1}(R) \subseteq \delta_n(R)$ and so $|\delta_{n+1}(R)| \leq |\delta_n(R)|$. Since $|\delta_n(R)| \leq \alpha_n \cdot \text{poly}_2(|Q|)$ by induction hypothesis, and $\alpha_n = \alpha_{n+1}$ we get that $|\delta_{n+1}(R)| \leq \alpha_{n+1} \cdot \text{poly}_2(|Q|)$.

The second case is that $\alpha_{n+1} > \alpha_n$. Then by Lemma 13 we have $|\delta_{n+1}(R)| \leq \max\{|\delta_n(R')| : R' \text{ a residue class}\} + \text{poly}_2(|Q|)$. Since the right-hand side of the latter is at most $\leq \alpha_n \cdot \text{poly}_2(|Q|) + \text{poly}_2(|Q|)$, by induction hypothesis, and $\alpha_{n+1} > \alpha_n$ we get that $|\delta_{n+1}(R)| \leq \alpha_{n+1} \cdot \text{poly}_2(|Q|)$. ◀

Recall that $(U_n)_{n\in\mathbb{N}}$ is a monotone sequence. Furthermore, observe that by the proof of Lemma 13 the sequence $|\delta_n(R)|$ either strictly decreases, or possible increases if R contains a chain that is $(n+1)$-active but not n-active. Since the latter can only take place $|Q|$ times, then $|\delta_n(R)|$ can take a polynomial number of distinct values before converging. Thus, as a consequence of Proposition 15 we have:

▶ **Corollary 16.** *The sequence $(U_n)_{n\in\mathbb{N}}$ stabilizes in at most $\text{poly}_1(|Q|)$ steps.*

3.4 Computing $Conf_\infty$ and Deciding Unboundedness

In this section we show how to compute $Conf_\infty$ in polynomial time and how to decide in polynomial time whether the initial configuration $(s, 0)$ can reach $Conf_\infty$.

We start by showing that if a configuration can reach U_n via a primitive run, then it can also reach U_n via a polynomial-length run (see Appendix G for the proof).

▶ **Proposition 17.** *There exists a polynomial poly_7 such that the following holds. Let $(q, z), (q', z') \in Conf_+$ and let $(q, z) \xrightarrow{\pi} (q', z')$ be a valid run such that $(q', z') \in U_n$ and π is primitive. Then there is a valid run $(q, z) \xrightarrow{\pi'} (q', z'')$ such that $(q', z'') \in U_n$ and π' has length at most $\text{poly}_7(|Q|)$.*

Recall that U'_{n+1} consists of all configurations in $Conf_+$ with minimal distance to U_n. Combining Remark 8 and Proposition 17, we have that the minimal distance from a configuration $(q, z) \in U'_{n+1} \setminus U_n$ to U_n is at most $\text{poly}_7(|Q|)$. It follows that we can restrict the search for configurations that can reach U_n, to those within a polynomially-bounded distance to U_n. By itself this is not sufficient to obtain a polynomial-time algorithm to decide whether U_n is reachable. However, using our analysis of the structure of U_n in Section 3.3, we are able to formulate the bounded reachability problem above in a form that admits a polynomial-time algorithm.

Specifically, we consider the *Bounded Coverability problem with a Disequality Objective*: Given as input a 1-VASS $\mathcal{V} = (Q, D, \Delta, w)$ with a distinguished state q_f, a positive integer L (written in unary), an initial configuration (q_0, x_0), and a coverability objective of the form

$$O = \left\{ (q_f, x) \mid x \geq \ell \wedge \bigwedge_{i=1}^{m} (x \not\equiv a_i \bmod W) \wedge \bigwedge_{i=1}^{n} (x \neq b_i) \right\}, \tag{3}$$

where ℓ, W and the a_i and b_i are non-negative integers given in binary, decide whether O is reachable from (q_0, x_0) via a valid run of length at most L.

▶ **Proposition 18.** *The Bounded Coverability problem with a Disequality Objective is decidable in polynomial time.*

We now show how to compute $Conf_\infty$ in polynomial time. By Corollary 16, the sequence $\{U_n\}_{n\in\mathbb{N}}$ converges in at most $\text{poly}_1(|Q|)$ steps. It remains to show how to compute U_{n+1} from U_n in polynomial time for each n.

Recall that all unbounded chains are contained in U_0 and hence are contained in U_n for all n. Recall also that the total number of bounded chains is at most $2|Q|$ and that U_n is downward closed in each bounded chain. Thus U_n is determined by giving, for every bounded chain C such that $U_n \cap C \neq \emptyset$, the maximum configuration in $U_n \cap C$. In particular, U_n can be described in space polynomial in the description of the given 1-VASS.

Recall that U_{n+1} is obtained from U_n by adding the configurations in $\mathrm{Conf}_+ \setminus U_n$ that have minimum distance to U_n and then closing downward in each bounded chain. By Remark 8 and Proposition 17, a configuration in $\mathrm{Conf}_+ \setminus U_n$ that has minimum distance to U_n has distance at most $\mathrm{poly}_7(|Q|)$. The idea to compute U_{n+1} from U_n is as follows:

For each bounded chain C, and each configuration $(q, x) \in C \setminus U_n$ that is among the top $\mathrm{poly}_1(|Q|)$ configurations in C, we determine the distance of (q, x) to U_n up to a bound of $\mathrm{poly}_7(|Q|)$. To do this we use the procedure described in Proposition 18, having first written U_n as a polynomial-size union of sets of the form (3) – see below for details. The reason that it suffices to look only among the top $\mathrm{poly}_1(|Q|)$ configurations in each bounded chain is because we know from Proposition 15 that $|C \setminus U_{n+1}| \leq \mathrm{poly}_1(|Q|)$ for every $(n+1)$-active chain C.

We next show how to decompose U_n into a polynomial union of sets of the form (3) in order to apply Proposition 18. Fixing $q \in Q_+$, let $R_1, \ldots, R_m$ be a list of the non-trivial q-residue classes and for each $i \in \{1, \ldots, m\}$, write a_i for the corresponding residue modulo W_q and define $\ell_i := \min(R_i \cap U_n)$. Moreover, let $b_1, \ldots, b_k$ be a list of the counter values such that for all $1 \leq j \leq k$ we have $b_j \geq \ell_i$ and $(q, b_j) \in R_i \setminus U_n$ for some i. Note that $m \leq |Q|$ and $k \leq m\,\mathrm{poly}_1(|Q|)$, and the corresponding classes and numbers can be enumerated in polynomial time. We decompose the set of configurations $\{(q, z) \in U_n\}$ into the following two components:

1. $\{(q, z) : z \geq \mathrm{pmin}(\gamma_q) \wedge \bigwedge_{i=1}^{m} z \not\equiv a_i \pmod{W_q}\}$, i.e., all configurations in trivial q-residue classes,
2. for all $j \in \{1, \ldots, m\}$, the set $\{(q, z) : z \geq \ell_j \wedge \bigwedge_{i:i\neq j} z \not\equiv a_i \pmod{W_q} \wedge \bigwedge_{i=1}^{k} z \neq b_i\}$, which includes $R_j \cap U_n$ for the non-trivial residue class R_j.

Finally, it remains to decide whether the configuration $(s, 0)$ is unbounded. By Proposition 4, $(s, 0)$ is unbounded if and only if it can reach Conf_∞. Now a shortest run from $(s, 0)$ to Conf_∞ is necessarily primitive: if an internal configuration in such a run lies in Conf_+ then it is also in Conf_∞ – a contradiction. By Proposition 17, a shortest run from $(s, 0)$ to Conf_∞ has length at most $\mathrm{poly}_7(|Q|)$. Thus we can decide whether such a run exists in polynomial time using Proposition 18. In conclusion we have

▶ **Theorem 19.** *The Unboundedness Problem and the Coverability Problem for 1-VASS with disequality tests are decidable in polynomial time.*

4 Unboundedness for 1-VASS

In this section we show that the Unboundedness Problem for 1-VASS (i.e., with no disequality tests) is in NC^2. Recall that NC^i is the class of decision problems solvable in time $O(\log^i n)$, with n the size of the input, on a parallel computer with a polynomial number of processors [13, 1].

Let $\mathcal{V} = (Q, \Delta, w)$ be a 1-VASS with a distinguished state $s \in Q$. We want to decide whether the configuration $(s, 0)$ is unbounded. Since $\mathcal{V}$ has no disequality tests, deleting a negative-weight or zero-weight cycle that appears as an infix of a valid run yields another valid run. It follows that $(s, 0)$ is unbounded if and only if there is a valid run from $(s, 0)$ consisting of a simple path (of length at most $|Q|$) followed by a positive-weight simple cycle (again, of length at most $|Q|$). We call such a run a *lasso*.

Let $\mathcal{V} = (Q, \Delta, w)$ be a 1-VASS and let $\pi = q_1, \ldots, q_n$ be a path in $\mathcal{V}$. Recall that a (possibly empty) prefix of π is said to be *minimal* if it has minimal weight among all prefixes of π. Likewise a (possibly empty) suffix of π is said to be *maximal* if it has maximal weight among all suffixes. It is clear that $q_1, \ldots, q_m$ is a minimal prefix of π if and only if $q_m, \ldots, q_n$ is a maximal suffix. In such a case let us call q_m a *nadir* of π (the nadir is the lowest point reached in any run over π). Recall that $\mathrm{pmin}(\pi)$ is the weight of a minimal prefix of π; correspondingly we define $\mathrm{smax}(\pi)$ to be the weight of a maximal suffix.

Given paths π and π', say that π is *dominated* by π' if $\mathrm{pmin}(\pi) \leq \mathrm{pmin}(\pi')$ and $\mathrm{smax}(\pi) \leq \mathrm{smax}(\pi')$. Observe that if π is dominated by π' then $\mathrm{weight}(\pi) \leq \mathrm{weight}(\pi')$.

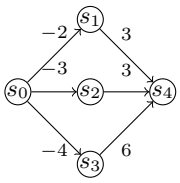


Figure 5 The topmost path dominates the middle one; the bottom path dominates no other path.

▶ **Example 20.** In Figure 5, the path s_0, s_1, s_4 dominates s_0, s_2, s_4. However, despite it being the case that $\mathrm{weight}(s_0, s_3, s_4) > \mathrm{weight}(s_0, s_2, s_4)$, s_0, s_3, s_4 does not dominate s_0, s_2, s_4 since the weight of a minimal prefix of the former is smaller than that of the latter.

Fix two states $p, q \in Q$ and let P be a set of p-q paths. We say that a set P' of p-q paths is a *Pareto set* for P if for every $\pi \in P$ there exists $\pi' \in P'$ such that π is dominated by π'.

We observe some simple properties of Pareto sets:

▶ **Lemma 21.** *Let $p, q, r \in Q$. Then all of the following statements hold:*

1. *If P_1, P_2, P_3 are sets of p-q paths such that P_1 is a Pareto set of P_2 and P_2 is a Pareto set of P_3, then P_1 is a Pareto set of P_3.*
2. *If P, R are sets of p-q paths with respective Pareto sets P', R', then $P' \cup R'$ is a Pareto set for $P \cup R$*
3. *If P is a set of p-q paths and R is a set of q-r paths with respective Pareto sets P', R', then $P' \cdot R'$ is a Pareto set of $P \cdot R$.*

▶ **Proposition 22.** *Let $p, q \in Q$. Then every set P of p-q paths of length at most k has a Pareto set P' of cardinality at most $|Q|$ such that each path in P' has length at most $2k$. Moreover such a set P' can be computed from P in NC^1.*

An NC^2 Upper Bound

▶ **Theorem 23.** *The Unboundedness Problem and the Coverability Problem for 1-VASS are decidable in NC^2.*

Proof. By Lemma 1, it will suffice to show that Unboundedness is in NC^2.

Let $\mathcal{V} = (Q, \Delta, w)$ be a 1-VASS. Given $p, q \in Q$ and $m \in \mathbb{N}$, denote by $\mathrm{Paths}_{p,q,m}$ the set of all p-q paths in $\mathcal{V}$ of length at most m.

Given a state $s \in Q$, recall that $(s, 0)$ is unbounded if and only if there exists a lasso run that starts at $(s, 0)$. To determine the existence of such a run we compute a Pareto set P_q for $\mathrm{Paths}_{s,q,|Q|}$ and a Pareto set P'_q for $\mathrm{Paths}_{q,q,|Q|}$ for every state $q \in Q$. Having done this we look for $q \in Q$ and paths $\pi \in P_q$ and $\pi' \in P'_q$ such that $\pi \cdot \pi'$ induces a valid run from $(s, 0)$ and π' has positive weight.

It remains to show how to compute a Pareto set of $\text{Paths}_{p,q,|Q|}$ for all pairs of states $p, q \in Q$ (together with the values $\text{weight}(\pi)$ and $\text{pmin}(\pi)$ for every path π in the Pareto set) in NC^2.

For $k = 1, \ldots, \lceil \log |Q| \rceil$, we show how to compute a family $\mathcal{P}_k = \{P_{p,q,k}\}_{p,q \in Q}$ such that for all $p, q \in Q$:

1. $P_{p,q,k}$ is a Pareto set for $\text{Paths}_{p,q,2^k}$;
2. $P_{p,q,k} \subseteq \text{Paths}_{p,q,4^k}$;
3. $|P_{p,q,k}| \leq |Q|$.

By Item 1, if $k = \lceil \log |Q| \rceil$ then $P_{p,q,k}$ is a Pareto set for $\text{Paths}_{p,q,|Q|}$. (Note that for $k = \lceil \log |Q| \rceil$, $\mathcal{P}_k$ consists of paths of length at most $|Q|^2$.)

The construction of $\mathcal{P}_k$ is by induction on k. Suppose we have computed $\mathcal{P}_k$ with Properties 1-3 above. Fix $p, q \in Q$. In order to compute $P_{p,q,k+1}$ we observe that

$$P := \{\pi_1 \cdot \pi_2 : \exists r \in Q(\pi_1 \in P_{p,r,k} \wedge \pi_2 \in P_{r,q,k})\} \tag{4}$$

is a Pareto set for $\text{Paths}_{p,q,2^{k+1}}$ by Items 2 and 3 of Lemma 21. Applying Proposition 22, we obtain a Pareto set P' for P of cardinality at most $|Q|$. By Item 1 of Lemma 21, P' is a Pareto set for $\text{Paths}_{p,q,2^{k+1}}$. Finally, it is clear from the length bound in Proposition 22 that all paths in P' have length at most 4^{k+1}. Thus we define $P_{p,q,k+1} := P'$.

It remains to establish the NC^2 complexity bound for computing $\mathcal{P}_{\lceil \log |Q| \rceil}$. For this it suffices to show that for all k the computation of $\mathcal{P}_{k+1}$ from $\mathcal{P}_k$ can be carried out in NC^1. But we may compute each set $P_{p,q,k+1}$ in parallel (over $p, q \in Q$), and the computation of each such set can be done in NC^1 by Proposition 22. ◀

5 Conclusion

We have shown that control-state reachability for 1-VASS with disequality tests can be solved in polynomial time. The complexity of reaching a given *configuration* in this model is open (being equivalent to control-state reachability in the presence of both equality and disequality tests), lying between NP and PSPACE. For multi-dimensional VASS with disequality tests, the classical argument of Rackoff [15] easily generalises to show that control-state reachability remains in EXPSPACE. By contrast, decidability of reachability is open to the best of our knowledge. For comparison, recall that without disequality tests reachability is decidable but non-elementary [5].

References

1 Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009. URL: `http://www.cambridge.org/catalogue/catalogue.asp?isbn=9780521424264`.

2 Benedikt Bollig, Karin Quaas, and Arnaud Sangnier. The complexity of flat freeze LTL. In *28th International Conference on Concurrency Theory, CONCUR*, volume 85 of *LIPIcs*, pages 33:1–33:16, 2017.

3 Daniel P. Bovet and Pierluigi Crescenzi. *Introduction to the theory of complexity*. Prentice Hall international series in computer science. Prentice Hall, 1994.

4 Daniel Bundala and Joël Ouaknine. On parametric timed automata and one-counter machines. *Inf. Comput.*, 253:272–303, 2017.

5 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for petri nets is not elementary. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 24–33. ACM, 2019.

6 S. Demri, R. Lazic, and A. Sangnier. Model checking memoryful linear-time logics over one-counter automata. *Theor. Comput. Sci.*, 411(22-24):2298–2316, 2010.

7 John Fearnley and Marcin Jurdzinski. Reachability in two-clock timed automata is pspace-complete. In *Automata, Languages, and Programming – 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings*, volume 7966 of *Lecture Notes in Computer Science*, pages 212–223. Springer, 2013.

8 Alain Finkel, Stefan Göller, and Christoph Haase. Reachability in register machines with polynomial updates. In *Mathematical Foundations of Computer Science 2013 – 38th International Symposium, MFCS*, volume 8087 of *Lecture Notes in Computer Science*, pages 409–420. Springer, 2013.

9 Stefan Göller, Christoph Haase, Joël Ouaknine, and James Worrell. Model checking succinct and parametric one-counter automata. In *Automata, Languages and Programming, 37th International Colloquium, ICALP*, volume 6199 of *Lecture Notes in Computer Science*, pages 575–586. Springer, 2010.

10 C. Haase, S. Kreutzer, J. Ouaknine, and J. Worrell. Reachability in succinct and parametric one-counter automata. In *Proceedings of CONCUR*, volume 5710 of *LNCS*, pages 369–383. Springer, 2009.

11 P. Lafourcade, D. Lugiez, and R. Treinen. Intruder deduction for *AC*-like equational theories with homomorphisms. In *Research Report LSV-04-16, LSV, ENS de Cachan*, 2004.

12 Matti Nykänen and Esko Ukkonen. The exact path length problem. *J. Algorithms*, 42(1):41–53, 2002.

13 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.

14 Franco P. Preparata. New parallel-sorting schemes. *IEEE Trans. Computers*, 27(7):669–673, 1978. doi:10.1109/TC.1978.1675167.

15 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978.

16 Louis E. Rosier and Hsu-Chun Yen. A multiparameter analysis of the boundedness problem for vector addition systems. *J. Comput. Syst. Sci.*, 32(1):105–135, 1986.

17 Heribert Vollmer. *Introduction to Circuit Complexity – A Uniform Approach*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1999. doi:10.1007/978-3-662-03927-4.

A Proof of the reduction in Figure 1

Proof. Let us recall that for every value $u \in \mathbb{N}$, the assignment $\mathrm{val}_u : \{X_1, \ldots, X_m\} \to \{0, 1\}$ is defined by $\mathrm{val}_u(X_i) = 1$ if and only if $p_i \mid u$. For convenience, define the domain $D_s \subseteq \mathbb{N}$ containing all allowable counter values in state s (exclude all disequality guards on s).

The key observation is the following: let $u \in \{0, \ldots, P-1\}$, and consider a clause $C_i = \ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3}$, where ℓ_{i_j} is a literal of variable X_{i_j}, then val_u satisfies C_i iff there exists some $k \in \mathbb{N}$ such that $u + k p_{i_1} p_{i_2} p_{i_3} \notin D_i$.

Indeed, note that for every $j \in \{1, 2, 3\}$ and every $k \in \mathbb{N}$ we have that $p_{i_j} | u$ iff $p_{i_j} | u + k p_{i_1} p_{i_2} p_{i_3}$. Recall that $\mathrm{val}_u(X_{i_j}) = 1$ iff $p_{i_j} | u$, and observe that since $u < P$, there exists $k \in \mathbb{N}$ such that $u + k p_{i_1} p_{i_2} p_{i_3} \in \{P, P+1, \ldots, P + p_{i_1} p_{i_2} p_{i_3} - 1\}$. We thus have that val_u satisfies C_i iff $\mathrm{val}_{u + k p_{i_1} p_{i_2} p_{i_3}}$ satisfies C_i, iff $u + k p_{i_1} p_{i_2} p_{i_3} \notin D_i$.

Now, assume φ is satisfiable, and let π be a satisfying assignment. We associate with π the number $u = \prod_{j : \pi(X_j) = 1} p_j \pmod P$ (note that taking modulo P simply means that if the product is exactly P, we take $u = 0$). Clearly $\pi = \mathrm{val}_u$. We claim that (s_0, u) is bounded. Indeed, the only paths possible from (s_0, u) start by choosing a state s_i, and then repeatedly applying the cycle of cost c_i. However, since val_u satisfies all clauses, then by the above, all such paths are blocked by a disequality guard after taking the c_i for k times, for some $k \in \mathbb{N}$ (which depends on i). Thus, (s_0, u) is bounded.

Conversely, assume (s_0, u) is bounded for some value u, we claim that val_u satisfies φ. Indeed, by the same reasoning above, it follows that for every cycle of cost c_i, we have $u + kc_i \notin D_i$ for some $k \in \mathbb{N}$, so val_u satisfies C_i. Since this is true for all clauses, we have that val_u satisfies φ.

We conclude that φ is satisfiable iff some configuration (s_0, u) is bounded, which completes the reduction.

Finally, we note that the reduction indeed takes polynomial time – indeed, the construction clearly has polynomially many states. Also, the first m primes $p_1, \ldots, p_m$ can be listed in time polynomial in m, and are representable in polynomially many bits. Therefore, the binary representation of the transition values and the amount of missing elements in the domain of each state are both polynomial. ◀

B Single disequality guards suffice

Given a 1-VASS $\mathcal{V} = (Q, \Delta, D, w)$ with disequality tests, we can assume that for all states q the set D_q is either $\mathbb{N}$ or $\mathbb{N} \setminus \{g\}$ for some $g \in \mathbb{N}$. This assumption is without loss of generality, as a state q with $D_q = \mathbb{N} \setminus \{a_1, \ldots, a_n\}$ can be replaced with a sequence of new states $q_1, \cdots, q_n$, connected with 0-weight transitions, such that $D_{q_i} = \mathbb{N} \setminus \{a_i\}$ for $i \in \{1, \ldots, n\}$. The transformation yields only a polynomial blow-up in the size of the 1-VASS, and there is a natural correspondence between runs in the original 1-VASS and the modified one.

C Proof of Lemma 1

Proof. Consider a 1-VASS $\mathcal{V} = (Q, \Delta, D, w)$ with disequality tests, and let $s, t \in Q$. We reduce the Coverability problem to the Unboundedness problem as follows.

We obtain from $\mathcal{V}$ a new 1-VASS $\mathcal{V}'$ as follows. First, we remove from $\mathcal{V}$ all the states that cannot reach t in the underlying graph. Second, we introduce a new state t' with a self-loop of weight $+1$, that is reachable from t with a transition of weight 0. The output of the reduction is $\mathcal{V}'$ with the distinguished state s.

Recall that reachability in directed graphs can be decided in $\mathsf{NL} \subseteq \mathsf{NC}^2$, and hence this reduction is NC^2-computable.

Henceforth assume that s can reach t in the underlying graph of $\mathcal{V}$ (otherwise s cannot cover t, and the reduction can output a trivial negative instance). We proceed to prove the correctness of the reduction.

First, if $(s, 0)$ can cover t in $\mathcal{V}$, then in particular it can only cover t using states in $\mathcal{V}'$. We now have that $(s, 0)$ is unbounded in $\mathcal{V}'$, by covering t, and then taking the transition to t' and repeating the self loop unboundedly. Note that crucially, there are no disequality guards on t', and therefore once t is reached, we can take the transition to t and repeat the self loop unboundedly.

Conversely, suppose $(s, 0)$ is unbounded in $\mathcal{V}'$, then either there is a valid run in $\mathcal{V}$ from $(s, 0)$ to (t', z) for some z, in which case $(s, 0)$ can cover t in $\mathcal{V}$, or $(s, 0)$ is unbounded already in $\mathcal{V}$ and, moreover, it is unbounded in $\mathcal{V}$ using only states that can reach t in the underlying graph. We claim that in the latter case, $(s, 0)$ can cover t in $\mathcal{V}$. Indeed, from $(s, 0)$ there is a valid run to a configuration (q, z) with z that is large enough, such that a simple path from q to t in the underlying graph lifts to a valid run from (q, z) to (t, z') for some z'. Specifically, taking $z > |Q| \cdot W \cdot G$ where W is the maximal absolute value of the weight of a transition in $\mathcal{V}$, and G is the maximal disequality guard, suffices for such a run. ◀

D Proof of Proposition 4

Proof. Clearly if $(s,0)$ can reach an unbounded configuration in $Conf_+$ then it is unbounded.

Conversely, if $(s,0)$ is unbounded, then there is a state q such that for all $z_0 \in \mathbb{N}$, there exist $z, z' \geq z_0$ and a valid run π starting in $(s,0)$ that visits (q,z) and ends in (q,z'). Thus, there is a positive cycle γ on q. The positive cycle γ on q may not be simple, but it certainly visits a state p with a simple positive cycle γ_p on it. Pick z_0 such that $z_0 > \mathrm{pmin}(\gamma) + x$. for all $x \in \mathrm{blocked}(\gamma_p^\omega)$ (Note that $\mathrm{blocked}(\gamma_p^\omega)$ is finite since γ_p is a positive cycle. The maximum is thus well-defined.) Hence, there is a valid run from $(s,0)$ to (p,y) where $y > \max(\mathrm{blocked}(\gamma_p^\omega))$. Observe that $(p,y) \in Conf_+$ and it is unbounded. ◀

E Proof of Proposition 10

Proof. Suppose that π has length strictly greater than $|Q|^2+2$. By the Pigeonhole principle, we can find $|Q|+1$ distinct proper prefixes (i.e. prefixes that are not just the initial state, or the entire path) of π that end in the same state. That is, $|Q|$ proper cycles on the same state. Let $\pi_1, \ldots, \pi_{|Q|+1}$ be a list of these prefixes, given in order of increasing length, and let the corresponding suffixes be $\pi'_1, \ldots, \pi'_{|Q|+1}$. We now consider two cases.

First, suppose that there exist $i < j$ such that $\mathrm{weight}(\pi_i)$ and $\mathrm{weight}(\pi_j)$ have the same residue modulo $W_{q'}$. Then define $\pi' := \pi_i \cdot \pi'_j$. In this case path π' lifts to a run from (q,z) to (q',z'') such that (q',z'') lies in the same q'-residue class as (q',z'). The second case is that the respective residue classes of $\mathrm{weight}(\pi_1), \ldots, \mathrm{weight}(\pi_{|Q|+1})$ modulo $W_{q'}$ are all distinct. Then there exists $i > 1$ such that, defining $\pi' := \pi_1 \cdot \pi'_i$, the path π' lifts to a run from (q,z) to (q',z'') such that (q',z'') lies in a trivial q'-residue class (as there are at most $|Q|$ non-trivial residue classes).

Continuing in this fashion we can recursively remove cycles from the original path π to eventually obtain a path π' that has length at most $|Q|^2+2$ and such that Item 3 is satisfied. Consider all maximal infixes that were removed from π to obtain π'. Note that each such infix must necessarily be a cycle as they arise from iteratively removing cycles. Since π was primitive, all of them must have non-positive weight. Hence, Items 1 and 2 also hold[3]. ◀

F Proof of Lemma 12

Proof. Consider two "consecutive" configurations $(q,z), (q,z+W_q) \in C \setminus U_n$, then all configurations (q,z') for $z \leq z' < z + W_q$ lie in pairwise-distinct q-residue classes. In particular, since there are at most $|Q|$ non-trivial residue classes, and since trivial residue classes are contained in U_0, we have that at most $|Q|$ such elements are in $\delta_n(C)$. ◀

G Proof of Proposition 17

Proof. By Proposition 15 we can find a polynomial $\mathrm{poly'}_7$ such that

$$\mathrm{poly'}_7(|Q|) \geq |Q|^2 + |Q| + 3 + \sum_{R \text{ non-trivial}} |\delta_n(R)| \tag{5}$$

for all $n \in \mathbb{N}$.

[3] Note that we do not claim that the intermediate paths obtained in the procedure are primitive nor that the individual cycles removed in this process are negative. Rather the observation is that π' can equivalently be obtained from π in one step by simultaneously removing a disjoint family of infixes, where each infix is a cycle (necessarily non-positive).

Set $\text{poly}_7(|Q|) := |Q| \cdot (\text{poly}'_7(|Q|))^2 + |Q|^2 + 4$, and consider a valid, primitive path π such that $\text{length}(\pi) > \text{poly}_7(|Q|)$ and $(q, z) \xrightarrow{\pi} (q', z')$.

Since π has length greater than $|Q| \cdot (\text{poly}'_7(|Q|))^2 + 2$, there exists a state $q'' \in Q$ that occurs at least $(\text{poly}'_7(|Q|))^2$ times in internal configurations within the first $|Q| \cdot (\text{poly}'_7(|Q|))^2 + 2$ configurations of π. Thus, there exists a sequence of proper prefixes $\pi_1 < \ldots < \pi_{\text{poly}'_7(|Q|)}$ of π that all end in q'' and such that one of the following two cases holds.

(i) The numbers $\text{weight}(\pi_i)$ all have the same residue modulo $W_{q'}$.

(ii) The numbers $\text{weight}(\pi_i)$ have pairwise distinct residues modulo $W_{q'}$.

Indeed, since there are $(\text{poly}'_7(|Q|))^2$ prefixes to choose from, either Case (i) holds, or there are strictly less than $\text{poly}'_7(|Q|)$ prefixes per residue class. If the latter holds then there must be least $\text{poly}'_7(|Q|)$ such distinct residue classes, so Case (ii) holds.

In either case, we decompose the computation π as $\pi = \pi_{\text{poly}'_7(|Q|)} \cdot \pi'$. Observe that since π is primitive, then so is π'. Applying Proposition 10 to π' we obtain a path π'' of length at most $|Q|^2 + 1$ such that $\pi_{\text{poly}'_7(|Q|)} \cdot \pi''$ leads from (q, x) to either the same residue class as (q', z') or to a trivial q'-residue class.

It is important to note that we cannot assume π'' is not blocked after the prefix $\pi_{\text{poly}'_7(|Q|)}$. However, since $|\text{blocked}(\pi'')| \leq |Q|^2$, we can remove from the list of prefixes at most $|Q|^2$ prefixes such that the remaining prefixes do not cause π'' to block. (Indeed, we will not modify the path by literally removing prefixes but rather cycles which correspond to the path from a prefix to a longer prefix. For now, we are only speaking about removing elements from the collection of prefixes we can choose from.) W.l.o.g, let $\pi_1, \ldots \pi_d$ be the remaining prefixes.

Consider the family of paths $\theta_i := \pi_i \cdot \pi''$ for $i \in \{1, \ldots, d\}$. Note that every θ_i is of length at most $\text{poly}_7(|Q|)$, and since the θ_i are obtained by removing q''-cycles, and since π is primitive, the configurations reached by θ_i are above (q', z'). We claim that one of the θ_i is a valid run from (q, z) to U_n.

We separate the analysis according to the cases above.

- In Case (i), if π'' leads to a trivial residue class, then all the θ_i reach U_n, and we are done. Otherwise, π'' leads to the same residue class as (q', z'). By our choice of $\text{poly}'_7(|Q|)$ in (5), we have that $d > \sum_{R \text{ non-trivial}} |\delta_n(R)|$. That is, there are more prefixes that do not cause π'' to block than there are missing elements above (q', z') in U_n. We conclude that some θ_i reaches U_n.
- In Case (ii), the paths θ_i all reach distinct residue classes. In particular, since there are more than $|Q|$ such prefixes – i.e. $d > |Q|$ by our choice of $\text{poly}'_7(|Q|)$ – then some θ_i reach trivial residue classes, and thus reach U_n. ◀

H Proof of Proposition 18

Proof. We carry out a forward reachability analysis starting from the initial configuration (q_0, x_0). The algorithm runs for $L + 1$ rounds. In the k-th round, we maintain for each state q a set $S_{q,k}$ of configurations (q, x) that are reachable from (q_0, x_0) by valid runs of length k. Let $R_{q,k}$ denote the set of all configurations (q, x) that are reachable from (q_0, x_0) by valid runs of length k. We maintain the invariant that if some configuration $(q, x) \in R_{q,k}$ can reach the objective O in $L - k$ steps via a path π then some configuration $(q, x') \in S_{q,k}$ can also reach O via the same path π. We output that the objective is reachable if and only if one of the sets $S_{q_f,k}$ for some $k \in \{0, \ldots, L\}$ intersects O. This last step is clearly sound, given the invariant.

The key to obtaining a polynomial-time runtime bound is to suitably prune the sets $S_{q,k}$ to keep them of polynomial size. In order to compute $\{S_{q,k+1}\}_{q\in Q}$ from $\{S_{q,k}\}_{q\in Q}$ we proceed as follows. First define $\{S'_{q,k}\}_{q\in Q}$ to be the indexed set of all valid configurations reachable in one step from $\{S_{q,k}\}_{q\in Q}$. Now we obtain $S_{q,k+1}$ from $S'_{q,k}$ by the following two steps:

- First, we delete from $S'_{q,k}$ all configurations (q,x) such that there are $(n+L)$ configurations (q,x') in $S'_{q,k}$ with $x' > x$ and $x' \equiv x \pmod W$.
- Secondly, we delete from $S'_{q,k}$ all configurations (q,x) such that there are $(n+L)(m+1)$ configurations (q,x') in $S'_{q,k}$ with $x' > x$.

Clearly each set $S_{q,k}$ has cardinality at most $(n+L)(m+1)$, and moreover, it can be computed from the collection of sets $\{S_{q',k-1} \mid q' \in Q\}$ in polynomial time.

It remains to argue that the invariant is maintained between rounds. To this end, suppose some state $(q,x) \in R_{q,k+1}$ can reach the objective in $L-k-1$ steps via a path π. Then there exists a state $(q',x') \in R_{q',k}$ that can reach the objective in $L-k$ steps via the path $q'\pi$. By the loop invariant there exists a state $(q',x'') \in S_{q',k}$ that can also reach the objective via the path $q'\pi$. Hence there is a state $(q,y) \in S'_{q',k}$ that can reach the objective via the path π. Now if (q,y) is deleted in the first stage of pruning then there is some configuration (q,y') such that $y' > y$, $y' \equiv y \pmod W$, and π yields a valid computation from (q,y') to the objective O. After the first stage of pruning, each residue class in $S'_{q,k}$ contains at most $n+L$ elements. Hence if (q,y') is deleted in the second stage of pruning, there are at least $n+L$ configurations (q,y'') in $S_{q,k+1}$ that are above (q,y') and are such that the run over π from (q,y') leads to a configuration (q_f,z) with $\bigwedge_{i=1}^{m} z \not\equiv a_i \bmod W$. Now from one of these configurations π yields a valid run that reaches O since one of $n+L$ choices of (q,y'') will avoid blocked(π) and lead to a configuration (q_f,z) such that $\bigwedge_{i=1}^{n} z \neq b_i$. ◀

I Proof of Lemma 21

Proof. Items 1 and 2 are obvious. Item 3 follows from the fact that if $\pi_1 \in P$ is dominated by $\pi'_1 \in P'$ and $\pi_2 \in R$ is dominated by $\pi'_2 \in R'$ then $\pi_1 \cdot \pi_2$ is dominated by $\pi'_1 \cdot \pi'_2$. Indeed,

$$\begin{aligned} \mathrm{pmin}(\pi_1 \cdot \pi_2) &= \min(\mathrm{pmin}(\pi_1), \mathrm{weight}(\pi_1) + \mathrm{pmin}(\pi_2)) \\ &\leq \min(\mathrm{pmin}(\pi'_1), \mathrm{weight}(\pi'_1) + \mathrm{pmin}(\pi'_2)) \\ &= \mathrm{pmin}(\pi'_1 \cdot \pi'_2)\,. \end{aligned}$$

We can similarly argue that $\mathrm{smax}(\pi_1 \cdot \pi_2) \leq \mathrm{smax}(\pi'_1 \cdot \pi'_2)$. ◀

J Proof of Proposition 22

Proof. Fix a state $r \in Q$. Consider all p-r paths that appear as a minimal prefix of some path in P. Pick a single such prefix π_1 of maximum weight. Likewise consider all r-q paths that appear as a maximal suffix of some path in P and pick a single such suffix π_2 of maximum weight. Now form the path $\pi := \pi_1 \cdot \pi_2$. This path dominates any path in P with nadir r. We define P' to be the set of paths π formed in this way as r runs through Q. By taking k large enough, we can suppose without loss of generality, that the absolute weight of all paths in P' is at most 2^k. That is, it can be encoded in binary using $k+1$ bits.

The NC^1 bound on computing P' relies on the well-known fact that the sum of a list of binary integers can be computed in NC^1 [17, Chapter 1]. To obtain P' we compute the weight of each prefix and suffix of every path in P in parallel. According to [17], this can be done in time $O(\log k)$ on a parallel computer with $|P|k$ processors: one for each element

of P and each midpoint $0 \leq m \leq k$. Finally, for each state $r \in Q$ in parallel, we find a maximum-weight prefix of a path in P that connects p and r and a maximum-weight suffix of a path in P that connects r and q. It is straightforward to prove the latter is also in NC^1 since sorting a list of numbers can be done in NC^1, [14, 3] thus completing the proof. ◀

Strategy Complexity of Parity Objectives in Countable MDPs

Stefan Kiefer
Department of Computer Science, University of Oxford, UK

Richard Mayr
School of Informatics, University of Edinburgh, UK

Mahsa Shirmohammadi
CNRS & IRIF, Université de Paris, FR

Patrick Totzke
Department of Computer Science, University of Liverpool, UK

Abstract

We study countably infinite MDPs with parity objectives. Unlike in finite MDPs, optimal strategies need not exist, and may require infinite memory if they do. We provide a complete picture of the exact strategy complexity of ε-optimal strategies (and optimal strategies, where they exist) for all subclasses of parity objectives in the Mostowski hierarchy. Either MD-strategies, Markov strategies, or 1-bit Markov strategies are necessary and sufficient, depending on the number of colors, the branching degree of the MDP, and whether one considers ε-optimal or optimal strategies. In particular, 1-bit Markov strategies are necessary and sufficient for ε-optimal (resp. optimal) strategies for general parity objectives.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains; Mathematics of computing → Probability and statistics

Keywords and phrases Markov decision processes, Parity objectives, Levy's zero-one law

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.39

Related Version A full version of the paper is [13], available at `http://arxiv.org/abs/2007.05065`.

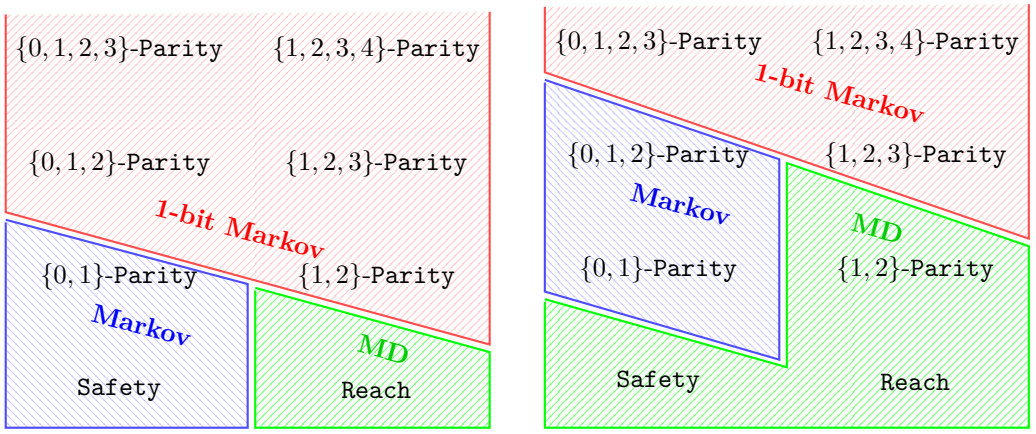
Funding *Stefan Kiefer*: Supported by a Royal Society University Fellowship.

1 Introduction

Background. Markov decision processes (MDPs) are a standard model for dynamic systems that exhibit both stochastic and controlled behavior [17]. MDPs play a prominent role in numerous domains, including artificial intelligence and machine learning [20, 19], control theory [4, 1], operations research and finance [5, 18], and formal verification [7, 2].

An MDP is a directed graph where states are either random or controlled. Its observed behavior is described by runs, which are infinite paths that are, in part, determined by the choices of a controller. If the current state is random then the next state is chosen according to a fixed probability distribution. Otherwise, if the current state is controlled, the controller can choose a distribution over all possible successor states. By fixing a strategy for the controller (and initial state), one obtains a probability space of runs of the MDP. The goal of the controller is to optimize the expected value of some objective function on the runs.

The type of strategy necessary to achieve an optimal (resp. ε-optimal) value for a given objective is called its *strategy complexity*. There are different types of strategies, depending on whether one can take the whole history of the run into account (history-dependent; (H)), or whether one is limited to a finite amount of memory (finite memory; (F)) or whether decisions are based only on the current state (memoryless; (M)). Moreover, the strategy type depends on whether the controller can randomize (R) or is limited to deterministic



(a) ε-optimal strategies for infinitely branching MDPs.

(b) Optimal strategies for infinitely branching MDPs.

Figure 1 These diagrams show the strategy complexity of ε-optimal strategies and optimal strategies (where they exist) for parity objectives. Depending on the position in the Mostowski hierarchy, either MD-strategies (green), deterministic Markov-strategies (blue) or deterministic 1-bit Markov strategies (red) are necessary and sufficient (and randomization does not help [12]). If the MDPs are finitely branching then the Markov strategies can be replaced by MD-strategies (i.e., the blue parts turn green), but the deterministic 1-bit Markov part (red) remains unchanged.

choices (D). The simplest type, MD, refers to memoryless deterministic strategies. *Markov strategies* are strategies that base their decisions only on the current state and the number of steps in the history of the run. Thus they do use infinite memory, but only in a very restricted form by maintaining an unbounded step-counter. Slightly more general are *1-bit Markov strategies* that use 1 bit of extra memory in addition to a step-counter.

Parity objectives. We study countably infinite MDPs with parity objectives. Parity conditions are widely used in temporal logic and formal verification, e.g., they can express ω-regular languages and modal μ-calculus [9]. Every state has a *color*, out of a finite set of colors encoded as natural numbers. A run is winning iff the highest color that is seen infinitely often is even. The controller wants to maximize the probability of winning runs. The Mostowski hierarchy [15] is a classification of parity conditions based on restricting the set of allowed colors. For instance, $\{1, 2, 3\}$-`Parity` objectives only use colors 1, 2, and 3. This includes Büchi ($\{1, 2\}$-`Parity`) and co-Büchi objectives ($\{0, 1\}$-`Parity`), both of which further subsume reachability and safety objectives.

Related work. In *finite* MDPs, there always exist optimal MD-strategies for parity objectives. In fact, this holds even for finite turn-based 2-player stochastic parity games [6, 23]. Similarly, there always exist optimal MD-strategies in countably infinite *non-stochastic* turn-based 2-player parity games [22].

The picture is more complex for countably infinite MDPs. Optimal strategies need not exist (not even for reachability objectives [17, 16]), and ε-optimal strategies for Büchi objectives [10] and optimal strategies for parity objectives [14] require infinite memory.

The paper [14] gave a complete classification whether MD-strategies suffice or whether infinite memory is required for ε-optimal (resp. optimal) strategies for all subclasses of parity objectives in the Mostowski-hierarchy.

However, the mere fact that infinite memory is required for (a subclass of) parity does not establish the precise strategy complexity. E.g., are Markov strategies (or Markov strategies with finite extra memory) sufficient?

In [12] we showed that deterministic 1-bit Markov strategies are both necessary and sufficient for ε-optimal strategies for Büchi objectives. I.e., deterministic 1-bit Markov strategies are sufficient, but neither randomized Markov strategies nor randomized finite-memory strategies are sufficient. This solved a 40-year old problem in gambling theory from [10, 11]. The same paper [12] showed that even for finitely branching MDPs with $\{1,2,3\}$-`Parity` objectives, optimal strategies (where they exist) need to be *at least* deterministic 1-bit Markov in general, i.e., neither randomized Markov nor randomized finite-memory strategies are sufficient.

While the lower bounds for ε-optimal strategies for Büchi objectives (resp. for optimal strategies for $\{1,2,3\}$-`Parity` objectives) carry over to general parity objectives, the upper bounds on the strategy complexity of ε-optimal (resp. optimal) parity remained open.

A basic upper bound and related conjecture. A basic upper bound on the complexity of ε-optimal strategies for parity can be obtained by using a combination of the results of [12] on Büchi objectives (1-bit Markov) and Lévy's zero-one law as follows. (However, note that the following argument does not work directly for optimal strategies.)

Informally speaking, Lévy's zero-one law implies that, for a tail objective (like parity) and any strategy, the level of attainment from the current state almost surely converges to either zero or one. I.e., the runs that always stay in states where the strategy attains something in $(0,1)$ is a null-set. A consequence for parity is that almost all winning runs must eventually, with ever higher probability, commit to winning by some particular color. Thus, with minimal losses (e.g., $\varepsilon/2$), after a sufficiently long finite prefix (depending on ε), one can switch to a strategy that aims to visit some *particular* color x infinitely often. The latter objective is like a Büchi objective where the states of color x are accepting and states of color $> x$ are considered losing sinks. By [12], an $\varepsilon/2$-optimal strategy for such a Büchi objective can be chosen 1-bit Markov. However, one would also need to remember which color x one is supposed to win by and *stick to that color*. The latter is critical, since strategies that switch focus between winning colors infinitely often (e.g., if they follow some local criteria based on the value of the current state wrt. various colors) can end up losing. Overall, the memory needed for such an ε-optimal strategy for parity is: $\lceil \log_2(c) \rceil$ bits for c even colors to remember which color x one is supposed to win by and Markov plus 1 bit for the Büchi strategy (see above), where the Markov step-counter also determines whether one still plays in the prefix. Thus Markov plus $(1 + \lceil \log_2(c) \rceil)$ bits are sufficient. This argument would suggest that more memory is required for more colors. However, our result shows that this is *not* the case.

Our contributions. We show *tight* upper bounds on the strategy complexity of ε-optimal (resp. optimal) strategies for parity objectives: They can be chosen as deterministic 1-bit Markov, regardless of the number of colors. I.e., we provide matching upper bounds to the lower bounds from [12].

In Section 3 we prove Theorem 1. An iterative plastering construction (i.e., fixing player choices on larger and larger subspaces) builds an ε-optimal 1-bit Markov strategy where the probability of never switching between winning even colors is $\geq 1 - \varepsilon$. Its correctness relies heavily on Lévy's zero-one law. The number of iterations is finite and proportional to the number of even colors. It eliminates the need to remember the winning color x and the $\lceil \log_2(c) \rceil$ part of the memory.

▶ **Theorem 1.** *Consider an MDP $\mathcal{M}$, a parity objective and a finite set S_0 of initial states.*
For every $\varepsilon > 0$ there exists a deterministic 1-bit Markov strategy that is ε-optimal from every state $s \in S_0$.

In Section 4 we prove Theorem 2. If an optimal strategy exists, then an optimal 1-bit Markov strategy can be constructed by the so-called *sea urchin* construction. It is a very complex plastering construction with infinitely many iterations that uses the results of Theorem 1 and Lévy's zero-one law as building blocks. Its name comes from the shape of the subspace in which player choices get fixed: a growing finite body (around a start set S_0) with a finite, but increasing, number of spikes, where each spike is of infinite size; cf. Figure 4. E.g., if the initial states are almost surely winning then, at the stage with i spikes, this strategy attains parity with some probability $\geq 1 - 2^{-i}$ already *inside* this subspace, and in the limit of $i \to \infty$ it attains parity almost surely. A further step even yields a single deterministic 1-bit Markov strategy that is optimal from every state that has an optimal strategy.

▶ **Theorem 2.** *Consider an MDP $\mathcal{M}$ with a parity objective and let S_{opt} be the subset of states that have an optimal strategy.*
There exists a deterministic 1-bit Markov strategy that is optimal from every $s \in S_{opt}$.

In Theorem 1 and Theorem 2 the initial content of the 1-bit memory is irrelevant (cf. Lemma 9, Lemma 18 and Remark 8).

Moreover, we show in Section 5 and Section 6 that in certain subcases deterministic Markov strategies are necessary and sufficient (i.e., these require a Markov step-counter, but not the extra bit): optimal strategies for co-Büchi and $\{0,1,2\}$-`Parity`, and ε-optimal strategies for safety and co-Büchi. In the special case of finitely branching MDPs, these Markov strategies (but not the 1-bit Markov strategies) can be replaced by MD-strategies.

Together with the previously established lower bounds, this yields a complete picture of the *exact* strategy complexity of parity objectives at all levels of the Mostowski hierarchy, for countable MDPs. Figure 1 gives a complete overview.

2 Preliminaries

A *probability distribution* over a countable set S is a function $f : S \to [0,1]$ with $\sum_{s \in S} f(s) = 1$. We write $\mathcal{D}(S)$ for the set of all probability distributions over S.

We study *Markov decision processes* (MDPs) over countably infinite state spaces. Formally, an MDP $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ consists of a countable set S of *states*, which is partitioned into a set $S_\Box$ of *controlled states* and a set $S_\bigcirc$ of *random states*, a *transition relation* $\longrightarrow \subseteq S \times S$, and a *probability function* $P : S_\bigcirc \to \mathcal{D}(S)$. We write $s \longrightarrow s'$ if $(s, s') \in \longrightarrow$, and refer to s' as a *successor* of s. We assume that every state has at least one successor. The probability function P assigns to each random state $s \in S_\bigcirc$ a probability distribution $P(s)$ over its set of successors. A *sink* is a subset $T \subseteq S$ closed under the $\longrightarrow$ relation. An MDP is *acyclic* if the underlying graph $(S, \longrightarrow)$ is acyclic. It is *finitely branching* if every state has finitely many successors and *infinitely branching* otherwise. An MDP without controlled states ($S_\Box = \emptyset$) is a *Markov chain*.

Strategies and Probability Measures. A *run* ρ is an infinite sequence $s_0 s_1 \cdots$ of states such that $s_i \longrightarrow s_{i+1}$ for all $i \in \mathbb{N}$; write $\rho(i) \stackrel{\text{def}}{=} s_i$ for the i-th state along ρ. A *partial run* is a finite prefix of a run. We say that (partial) run ρ *visits* s if $s = \rho(i)$ for some i, and that ρ *starts in* s if $s = \rho(0)$.

A *strategy* is a function $\sigma : S^*S_\Box \to \mathcal{D}(S)$ that assigns to partial runs $\rho s \in S^*S_\Box$ a distribution over the successors of s. A (partial) run $s_0 s_1 \cdots$ is *induced by* strategy σ if for all i either $s_i \in S_\Box$ and $\sigma(s_0 s_1 \cdots s_i)(s_{i+1}) > 0$, or $s_i \in S_\bigcirc$ and $P(s_i)(s_{i+1}) > 0$.

A strategy σ and an initial state $s_0 \in S$ induce a standard probability measure on sets of infinite plays. We write $\mathbb{P}_{\mathcal{M},s_0,\sigma}(\mathcal{R})$ for the probability of a measurable set $\mathcal{R} \subseteq s_0 S^\omega$ of runs starting from s_0. As usual, it is first defined on the *cylinders* $s_0 s_1 \dots s_n S^\omega$, where $s_1, \dots, s_n \in S$: if $s_0 s_1 \dots s_n$ is not a partial run induced by σ then $\mathbb{P}_{\mathcal{M},s_0,\sigma}(s_0 s_1 \dots s_n S^\omega) \stackrel{\text{def}}{=} 0$. Otherwise, $\mathbb{P}_{\mathcal{M},s_0,\sigma}(s_0 s_1 \dots s_n S^\omega) \stackrel{\text{def}}{=} \prod_{i=0}^{n-1} \bar{\sigma}(s_0 s_1 \dots s_i)(s_{i+1})$, where $\bar{\sigma}$ is the map that extends σ by $\bar{\sigma}(ws) = P(s)$ for all $ws \in S^*S_\bigcirc$. By Carathéodory's theorem [3], this extends uniquely to a probability measure $\mathbb{P}_{\mathcal{M},s_0,\sigma}$ on measurable subsets of $s_0 S^\omega$. We will write $\mathbb{E}_{\mathcal{M},s_0,\sigma}$ for the expectation w.r.t. $\mathbb{P}_{\mathcal{M},s_0,\sigma}$. We may drop the subscripts from notations, if it is understood.

Objectives. The objective of the player is determined by a predicate on infinite plays. We assume familiarity with the syntax and semantics of the temporal logic LTL [8]. Formulas are interpreted on the structure $(S, \longrightarrow)$. We use $[\![\varphi]\!]^s \subseteq sS^\omega$ to denote the set of runs starting from s that satisfy the LTL formula φ, which is a measurable set [21]. We also write $[\![\varphi]\!]$ for $\bigcup_{s\in S}[\![\varphi]\!]^s$. Where it does not cause confusion we will identify φ and $[\![\varphi]\!]$ and just write $\mathbb{P}_{\mathcal{M},s,\sigma}(\varphi)$ instead of $\mathbb{P}_{\mathcal{M},s,\sigma}([\![\varphi]\!]^s)$.

Given a set $T \subseteq S$ of states, the *reachability* objective $\texttt{Reach}(T)$ is the set of runs that visit T at least once; and the *safety objective* $\texttt{Safety}(T)$ is the set of runs that never visit T.

Let $\mathcal{C} \subseteq \mathbb{N}$ be a finite set of colors. A *color function* $Col : S \to \mathcal{C}$ assigns to each state s its color $Col(s)$. The parity objective, written as $\texttt{Parity}(Col)$, is the set of infinite runs such that the largest color that occurs infinitely often along the run is even. To define this formally, let $even(\mathcal{C}) = \{i \in \mathcal{C} \mid i \equiv 0 \mod 2\}$. For $\rhd \in \{<, \leq, =, \geq, >\}$, $n \in \mathbb{N}$, and $Q \subseteq S$, let $[Q]^{Col \rhd n} \stackrel{\text{def}}{=} \{s \in Q \mid Col(s) \rhd n\}$ be the set of states in Q with color $\rhd n$. Then

$$\texttt{Parity}(Col) \stackrel{\text{def}}{=} \bigvee_{i \in even(\mathcal{C})} \left(\mathsf{GF}[S]^{Col=i} \wedge \mathsf{FG}[S]^{Col\leq i}\right).$$

The Mostowski hierarchy [15] classifies parity objectives by restricting the range of Col to a set of colors $\mathcal{C} \subseteq \mathbb{N}$. We write $\mathcal{C}$-$\texttt{Parity}$ for such restricted parity objectives. In particular, the classical Büchi and co-Büchi objectives correspond to $\{1,2\}$-$\texttt{Parity}$ and $\{0,1\}$-$\texttt{Parity}$, respectively. These two classes are incomparable but both subsume the reachability and safety objectives. Assuming that T is a sink, $\texttt{Reach}(T) = \texttt{Parity}(Col)$ for the coloring with $Col(s) = 1 \iff s \notin T$ and $\texttt{Safety}(T) = \texttt{Parity}(Col)$ for the coloring with $Col(s) = 1 \iff s \in T$. Similarly, $\{0,1,2\}$-$\texttt{Parity}$ and $\{1,2,3\}$-$\texttt{Parity}$ are incomparable, but they both subsume (modulo renaming of colors) Büchi and co-Büchi objectives.

An objective φ is called a *tail objective* (resp. *suffix-closed*) iff for every run $\rho'\rho$ with some finite prefix ρ' we have $\rho'\rho \in \varphi \Leftrightarrow \rho \in \varphi$ (resp. $\rho'\rho \in \varphi \Rightarrow \rho \in \varphi$). In particular, $\texttt{Parity}(Col)$ is tail for every coloring Col. Moreover, if φ is suffix-closed then $\mathsf{F}\varphi$ is tail.

Strategy Classes. Strategies $\sigma : S^*S_\Box \to \mathcal{D}(S)$ are in general *randomized* (R) in the sense that they take values in $\mathcal{D}(S)$. A strategy σ is *deterministic* (D) if $\sigma(\rho)$ is a Dirac distribution for all partial runs $\rho \in S^*S_\Box$.

We formalize the amount of *memory* needed to implement strategies. Let M be a countable set of memory modes. An *update function* is a function $u : \mathsf{M} \times S \to \mathcal{D}(\mathsf{M} \times S)$ that meets the following two conditions, for all modes $\mathsf{m} \in \mathsf{M}$:

- for all controlled states $s \in S_\Box$, the distribution $u((\mathsf{m}, s))$ is over $\mathsf{M} \times \{s' \mid s \longrightarrow s'\}$.
- for all random states $s \in S_\bigcirc$, we have that $\sum_{\mathsf{m}' \in \mathsf{M}} u((\mathsf{m}, s))(\mathsf{m}', s') = P(s)(s')$.

An update function u together with an initial memory m_0 induce a strategy $u[\mathsf{m}_0] : S^*S_\Box \to \mathcal{D}(S)$ as follows. Consider the Markov chain with states set $\mathsf{M} \times S$, transition relation $(\mathsf{M} \times S)^2$ and probability function u. Any partial run $\rho = s_0 \cdots s_i$ in $\mathcal{M}$ gives rise to a set $H(\rho) = \{(\mathsf{m}_0, s_0) \cdots (\mathsf{m}_i, s_i) \mid \mathsf{m}_0, \ldots, \mathsf{m}_i \in \mathsf{M}\}$ of partial runs in this Markov chain. Each $\rho s \in s_0 S^* S_\Box$ induces a probability distribution $\mu_{\rho s} \in \mathcal{D}(\mathsf{M})$, the probability of being in state (m, s) conditioned on having taken some partial run from $H(\rho s)$. We define $u[\mathsf{m}_0]$ such that $u[\mathsf{m}_0](\rho s)(s') \stackrel{\text{def}}{=} \sum_{\mathsf{m},\mathsf{m}' \in \mathsf{M}} \mu_{\rho s}(\mathsf{m}) u((\mathsf{m}, s))(\mathsf{m}', s')$ for all $\rho s \in S^* S_\Box$ and $s' \in S$.

We say that a strategy σ can be *implemented* with memory M (and initial memory m_0) if there exists an update function u such that $\sigma = u[\mathsf{m}_0]$. In this case we may also write $\sigma[\mathsf{m}_0]$ to explicitly specify the initial memory mode m_0. Based on this, we can define several classes of strategies:

- A strategy σ is *memoryless* (M) (also called *positional*) if it can be implemented with a memory of size 1. We may view M-strategies as functions $\sigma : S_\Box \to \mathcal{D}(S)$.
- A strategy σ is *finite memory* (F) if there exists a finite memory M implementing σ. More specifically, a strategy is *k-bit* if it can be implemented with a memory of size 2^k. Such a strategy is then determined by a function $u : \{0,1\}^k \times S \to \mathcal{D}(\{0,1\}^k \times S)$.
- A strategy σ is *Markov* if it can be implemented with the natural numbers $\mathsf{M} = \mathbb{N}$ as the memory, initial memory mode $\mathsf{m}_0 = 0$ and a function u such that the distribution $u(\mathsf{m}, s)$ is over $\{\mathsf{m}+1\} \times S$ for all $\mathsf{m} \in \mathsf{M}$ and $s \in S$. Intuitively, such a strategy depends only on the current state and the number of steps taken so far.
- A strategy σ is *k-bit Markov* if it can be implemented with memory $\mathsf{M} = \mathbb{N} \times \{0,1\}^k$, $\mathsf{m}_0 \in \{0\} \times \{0,1\}^k$ and a function u such that the distribution $u((n, b, s))$ is over $\{n+1\} \times \{0,1\}^k \times S$ for all $(n, b) \in \mathsf{M}$ and $s \in S$.

Deterministic 1-bit strategies are central in this paper; by this we mean strategies that are both deterministic and 1-bit.

Optimal and ε-optimal Strategies. Given an objective φ, the value of state s in an MDP $\mathcal{M}$, denoted by $\mathtt{val}_{\mathcal{M}}(s)$, is the supremum probability of achieving φ. Formally, we have $\mathtt{val}_{\mathcal{M}}(s) \stackrel{\text{def}}{=} \sup_{\sigma \in \Sigma} \mathbb{P}_{\mathcal{M},s,\sigma}(\varphi)$ where Σ is the set of all strategies. For $\varepsilon \geq 0$ and state $s \in S$, we say that a strategy is ε-optimal from s iff $\mathbb{P}_{\mathcal{M},s,\sigma}(\varphi) \geq \mathtt{val}_{\mathcal{M}}(s) - \varepsilon$. A 0-optimal strategy is called optimal. An optimal strategy is almost-surely winning if $\mathtt{val}_{\mathcal{M}}(s) = 1$.

Considering an MD strategy as a function $\sigma : S_\Box \to S$ and $\varepsilon \geq 0$, σ is *uniformly* ε-optimal (resp. uniformly optimal) if it is ε-optimal (resp. optimal) from every $s \in S$.

Fixing and Safe Sets. Let σ be an MD strategy. Given a set $S' \subseteq S$ of states, write $\mathcal{M}[\sigma, S']$ for the MDP obtained from $\mathcal{M}$ by fixing the strategy σ for all states in S', that is, $\mathcal{M}[\sigma, S'] \stackrel{\text{def}}{=} (S, S_\Box \setminus S', S_\bigcirc \cup S', \longrightarrow, P')$ where $P'(s) \stackrel{\text{def}}{=} \sigma(s)$ for all $s \in S'$.

For an objective φ and a threshold $\beta \in [0,1]$, denote by $\mathit{Safe}_{\mathcal{M},\sigma,\varphi}(\beta)$ the set of all states s starting from which σ attains at least probability β; and denote by $\mathit{Safe}_{\mathcal{M},\varphi}(\beta)$ the set of states whose value for φ is at least β. Formally,

$$\mathit{Safe}_{\mathcal{M},\sigma,\varphi}(\beta) \stackrel{\text{def}}{=} \{s \in S \mid \mathbb{P}_{\mathcal{M},s,\sigma}(\varphi) \geq \beta\}, \quad \mathit{Safe}_{\mathcal{M},\varphi}(\beta) \stackrel{\text{def}}{=} \{s \in S \mid \mathtt{val}_{\mathcal{M},\varphi}(s) \geq \beta\}. \quad (1)$$

3 ε-Optimal Strategies for Parity

In this section we prove Theorem 1, stating that ε-optimal strategies for parity objectives can be chosen 1-bit Markov. Given an MDP we convert it by three successive reductions to a structurally simpler MDP where strategies require less sophistication to achieve parity.

First reduction (Finitely Branching). This reduction converts an infinitely branching MDP $\mathcal{M}$ to a finitely branching one $\mathcal{M}'$, with a clear bijection between the strategies in $\mathcal{M}$ and $\mathcal{M}'$. The construction, first presented in our previous work [12], replaces each controlled state s, that has infinitely many successors $(s_i)_{i\in\mathbb{N}}$, with a "ladder" of controlled states $(q_i)_{i\in\mathbb{N}}$, where each q_i has only two successors: q_{i+1} and s_i. Roughly speaking, the controller choice of successor s_n at s in $\mathcal{M}$, is simulated by a series of choices q_{i+1} at q_i, $0 \leq i < n$, followed by a choice of successor s_n in state q_n in $\mathcal{M}'$, and vice versa.

To prevent scenarios when the controller in $\mathcal{M}'$ stays on a ladder and never commits to a decision, we assign color 1 to all states $(q_i)_{i\geq 1}$ on the ladder (q_0 inherits the color of s). Hence, a hesitant run on the ladder is losing for parity. So w.l.o.g. we can assume that the given $\mathcal{M}$ is finitely branching.

▶ **Lemma 3.**

1. *Suppose that for every finitely branching acyclic MDP with a finite set S_0 of initial states, and a parity objective, there exist ε-optimal deterministic 1-bit strategies from S_0.*
   Then even for every infinitely branching acyclic MDP with a finite set S_0 of initial states and a parity objective, there exist ε-optimal deterministic 1-bit strategies from S_0.
2. *Suppose that for every finitely branching acyclic MDP with a parity objective, there exists a deterministic 1-bit strategy that is optimal from all states that have an optimal strategy.*
   Then even for every infinitely branching acyclic MDP with a parity objective, there exists a deterministic 1-bit strategy that is optimal from all states that have an optimal strategy.

Second reduction (Acyclicity). A deterministic 1-bit Markov strategy can be seen as a function $\sigma : \mathbb{N} \times \{0,1\} \times S \to \{0,1\} \times S$, where σ has access to an internal bit $b \in \{0,1\}$, which can be updated freely, and a step counter $k \in \mathbb{N}$, which increments by one in each step. Having b and k, σ produces a decision based on the current state of the MDP.

Following [12], we encode the step-counter from strategies into MDPs s.t. the current state of the system uniquely determines the length of the path taken so far. This translation allows us to focus on acyclic MDPs.

▶ **Lemma 4.** *Consider MDPs with a parity objective and $k \in \mathbb{N}$.*

1. *Suppose that for every acyclic MDP $\mathcal{M}'$ and every* finite *set of initial states S'_0 and $\varepsilon > 0$, there exists a deterministic k-bit strategy that is ε-optimal from all states $s \in S'_0$.*
   Then for every MDP $\mathcal{M}$ and every finite *set of initial states S_0 and $\varepsilon > 0$, there exists a deterministic k-bit Markov strategy that is ε-optimal from all states $s \in S_0$.*
2. *Suppose that for every acyclic MDP $\mathcal{M}'$ and $\varepsilon > 0$, there exists a deterministic k-bit strategy that is ε-optimal from all states. Then for every MDP $\mathcal{M}$ and $\varepsilon > 0$, there exists a deterministic k-bit Markov strategy that is ε-optimal from all states.*
3. *Suppose that for every acyclic MDP $\mathcal{M}'$, where S'_{opt} is the subset of states that have an optimal strategy, there exists a deterministic k-bit strategy that is optimal from all states $s \in S'_{opt}$. Then for every MDP $\mathcal{M}$, where S_{opt} is the subset of states that have an optimal strategy, there exists a deterministic k-bit Markov strategy that is optimal from all states $s \in S_{opt}$.*

By Lemma 4, the sufficiency of deterministic 1-bit strategies in acyclic MDPs implies the sufficiency of deterministic 1-bit Markov strategies in general MDPs. Thus to prove Theorem 1, it suffices to prove the following:

▶ **Theorem 5.** *Consider an acyclic MDP $\mathcal{M}$, a parity objective and a finite set S_0 of states. For every $\varepsilon > 0$ there exists a deterministic 1-bit strategy that is ε-optimal from every $s \in S_0$.*

Third reduction (Layered MDP). This reduction is in the same spirit of the previous one, in which the bit $b \in \{0,1\}$ is transferred from strategies to MDPs. Given an MDP $\mathcal{M}$, the corresponding *layered* MDP $\mathcal{L}(\mathcal{M})$ has two copies of each state $s \in S$ and each transition $t \in \longrightarrow_1$ of $\mathcal{M}$, one augmented with bit 0 and another with bit 1: (s,i) and (t,j) with $i,j \in \{0,1\}$. The states (s,i) are random if $s \in S_\bigcirc$ and controlled if $s \in S_\Box$. All the (t,j) are controlled. If there is a transition $t = (a,b)$ from state a to b in $\mathcal{M}$, there will be two transitions from (a,i) to (t,i), and four transitions from (t,i) to (b,j) in $\mathcal{L}(\mathcal{M})$; see Figure 2.

A 1-bit deterministic strategy in $\mathcal{M}$ at a state a picks a single successor b and may flip the bit from i to j; this is simulated in $\mathcal{L}(\mathcal{M})$ with an MD strategy σ within two consecutive steps: σ first chooses the transition $t = (a,b)$ by $\sigma(a,i) = (t,i)$ and then updates the bit by $\sigma(t,i) = (b,j)$ thereby moving from layer i to layer j. The controlled states (t,i) are essential for a correct simulation, since otherwise the controller cannot freely flip the bit (switch between layers) after it observes the successor chosen randomly at a random state.

▶ **Definition 6** (Layered MDP). *Given an MDP $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow_1, P_1)$ with coloring $Col_1 : S \to \mathcal{C}$, we define the corresponding* layered MDP $\mathcal{L}(\mathcal{M}) = (L, L_\Box, L_\bigcirc, \longrightarrow_2, P_2)$ *with coloring $Col_2 : L \to \mathcal{C}$ as follows.*

- $L \stackrel{def}{=} (S \cup \longrightarrow_1) \times \{0,1\}$ *where the set of controlled states is* $L_\Box \stackrel{def}{=} (S_\Box \cup \longrightarrow_1) \times \{0,1\}$.
- *For all $t \in \longrightarrow_1$ such that $t = (s,s')$ and for all $i,j \in \{0,1\}$, we have:*
  1. $(s,i) \longrightarrow_2 (t,i)$ *and* $(t,i) \longrightarrow_2 (s',j)$,
  2. $P(s,i)((t,i)) \stackrel{def}{=} P(s)(s')$ *iff* $s \in S_\bigcirc$, *and*
  3. $Col_2((s,i)) \stackrel{def}{=} Col_1(s)$ *and* $Col_2((t,i)) \stackrel{def}{=} Col_1(s')$.

The layered MDP of an acyclic MDP is acyclic. For $q \in S \cup \longrightarrow_1$, we refer to the copies of q in layer 0 and layer 1 as *siblings*: $(q,0)$ and $(q,1)$. A set $B \subseteq L$ is *closed* if for each state $(q,i) \in B$ its sibling is also in B. Denote by $Cl(B)$ the minimal closed superset of B.

▶ **Lemma 7.** *Consider an acyclic MDP $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ with a parity objective $\varphi = \mathtt{Parity}(Col)$ and let $\mathcal{L}(\mathcal{M})$ be the corresponding layered MDP.*

For every deterministic 1-bit strategy $u[\mathsf{m}_0]$ in $\mathcal{M}$ there is a corresponding MD strategy τ in $\mathcal{L}(\mathcal{M})$, and vice-versa, such that for every $s_0 \in S$, $\mathbb{P}_{\mathcal{L}(\mathcal{M}),(s_0,\mathsf{m}_0),\tau}(\varphi) = \mathbb{P}_{\mathcal{M},s_0,u[\mathsf{m}_0]}(\varphi)$.

▶ Remark 8. We note that, in a layered system $\mathcal{L}(\mathcal{M})$, any two siblings have the same value w.r.t. a parity objective φ. Moreover, any state s in $\mathcal{M}$ has an optimal strategy iff $(s,0) \in \mathcal{L}(\mathcal{M})$ has an optimal strategy iff its sibling $(s,1)$ has an optimal strategy.

Suppose τ is an MD strategy in $\mathcal{L}(\mathcal{M})$ that is optimal for all states that have an optimal strategy. Let u be the update function of a corresponding 1-bit strategy in $\mathcal{M}$, derived as described in Lemma 7. Then for every state s in $\mathcal{M}$ that has an optimal strategy we have $\mathbb{P}_{\mathcal{M},s,u[0]}(\varphi) = \mathbb{P}_{\mathcal{L}(\mathcal{M}),(s,0),\tau}(\varphi) = \mathbb{P}_{\mathcal{L}(\mathcal{M}),(s,1),\tau}(\varphi) = \mathbb{P}_{\mathcal{M},s,u[1]}(\varphi)$. That is, both $u[0]$ and $u[1]$ are optimal from s, so the initial memory mode is irrelevant. ◀

To prove Theorem 5, given an acyclic MDP, a set of initial states S_0 and $\varepsilon > 0$, we consider the layered MDP $\mathcal{L}(\mathcal{M})$ and set $L_0 = S_0 \times \{0\}$ of initial states. In the following lemma, we prove that there exists a single MD strategy that is ε-optimal starting from every state $\ell_0 \in L_0$ in $\mathcal{L}(\mathcal{M})$. This and Lemma 7 will directly lead to Theorem 5.

▶ **Lemma 9.** *Consider an acyclic MDP $\mathcal{M}$ and parity objective $\varphi = \mathtt{Parity}(Col)$. Let $\mathcal{L}(\mathcal{M})$ be the layered MDP of $\mathcal{M}$ and Col. For all finite sets L_0 of states in $\mathcal{L}(\mathcal{M})$ and all $\varepsilon > 0$ there exists a single MD strategy that is ε-optimal for φ from every state $\ell_0 \in L_0$.*

In the rest of this section, we prove Lemma 9. We fix a layered MDP $\mathcal{L}(\mathcal{M})$ (or simply $\mathcal{L}$) obtained from a given acyclic and finitely branching MDP $\mathcal{M}$ and a coloring $Col : S \to \mathcal{C}$, where the set of states is L and the finite set of initial states is $L_0 \subseteq L$. Let φ be the resulting parity objective in $\mathcal{L}$.

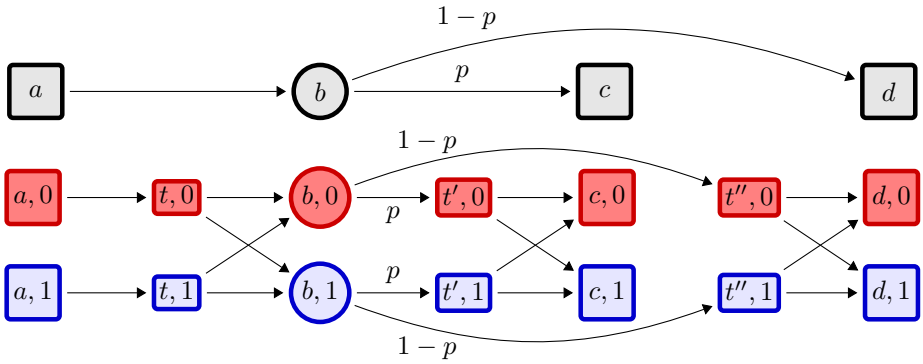


Figure 2 An MDP $\mathcal{M}$ (in grey) and the corresponding layered MDP $\mathcal{L}(\mathcal{M})$ with states of layer 0 and 1 in red and blue, respectively. Here, $t = (a, b)$, $t' = (b, c)$ and $t'' = (b, d)$ are transitions of $\mathcal{M}$.

Recall that $even(\mathcal{C}) = 2\mathbb{N} \cap \mathcal{C}$ denotes the set of even colors. We denote by $e_{\max}$ the largest even color in $even(\mathcal{C})$ and assume w.l.o.g., that $even(\mathcal{C})$ contains all even numbers from 2 to $e_{\max}$ inclusive. We have:

$$\begin{aligned}
\varphi &\stackrel{\text{def}}{=} \bigvee_{e \in even(\mathcal{C})} \left(\mathsf{GF}[L]^{Col=e} \wedge \mathsf{FG}[L]^{Col\leq e}\right) \\
&= \bigvee_{e \in even(\mathcal{C})} \left(\mathsf{FGF}[L]^{Col=e} \wedge \mathsf{FG}[L]^{Col\leq e}\right) \qquad \text{since } \mathsf{GF}[L]^{Col=e} \text{ is a tail objective} \\
&= \bigvee_{e \in even(\mathcal{C})} \mathsf{F}\left(\mathsf{GF}[L]^{Col=e} \wedge \mathsf{G}[L]^{Col\leq e}\right) \qquad \text{since } \mathsf{FG}A \wedge \mathsf{FG}B = \mathsf{F}(\mathsf{G}A \wedge \mathsf{G}B) \\
&= \bigvee_{e \in even(\mathcal{C})} \mathsf{F}\varphi_e \,,
\end{aligned}$$

where $\varphi_e \stackrel{\text{def}}{=} \left(\mathsf{GF}[L]^{Col=e} \wedge \mathsf{G}[L]^{Col\leq e}\right)$. Indeed, φ_e is the set of runs that win through color e (i.e., by visiting color e infinitely often and never visiting larger colors). Since the $\mathsf{F}\varphi_e$ are disjoint, for all states ℓ and strategies σ, we have:

$$\mathbb{P}_{\mathcal{L},\ell,\sigma}(\varphi) = \sum_{e \in even(\mathcal{C})} \mathbb{P}_{\mathcal{L},\ell,\sigma}(\mathsf{F}\varphi_e). \tag{2}$$

Fix $\varepsilon > 0$ and define $\gamma \stackrel{\text{def}}{=} \frac{\varepsilon}{e_{\max}+2}$. To construct an MD strategy $\hat{\sigma}$ that is ε-optimal starting from every state in L_0 we have an iterative procedure. In each iteration, we define $\hat{\sigma}$ at states in some carefully chosen region; and continuing in this fashion, we gradually fix all choices of $\hat{\sigma}$. In an iteration, in order to fix "good" choices in the "right" region we need to carefully observe the behavior of finitely many $\frac{\gamma}{2}$-optimal strategies σ_{ℓ_0}, one for each $\ell_0 \in L_0$, which must respect the choices already fixed in previous iterations. We thus view these strategies σ_{ℓ_0} to be $\frac{\gamma}{2}$-optimal not in $\mathcal{L}$ but in another layered MDP that is derived from $\mathcal{L}$ after fixing the choices of partially defined $\hat{\sigma}$.

In more detail, the proof consists of exactly $\frac{e_{\max}}{2} + 1$ iterations: one iteration for each even color e and a final "reach" iteration. Starting from color 2 and $\mathcal{L}_0 \stackrel{\text{def}}{=} \mathcal{L}$, in the iteration $e \in \{2, \cdots, e_{\max}\}$, we obtain a layered MDP $\mathcal{L}_e$ from $\mathcal{L}_{e-2}$ by fixing a single choice for each controlled state in a set fix_e. Roughly speaking, a run that falls in the set fix_e is likely going to win through φ_e (win through color e). We identify a certain subspace of fix_e, referred to as core_e, such that the following crucial fact holds: Once core_e is visited the run

remains in fix_e with probability at least $1-\gamma$. At the final iteration, we fix the choices of all remaining states to maximize the probability of falling into the union of core_e sets. As mentioned, the majority of such runs that visit core_e, for some color e, will stay in fix_e forever and thus win parity through color e. After all the iterations, all choices of all controlled states are fixed, and this prescribes the MD strategy $\hat{\sigma}$ from L_0 in $\mathcal{L}$.

In order to define the sets fix_e we heavily use Lévy's zero-one law and follow an inductive transformation on objectives. Lévy's zero-one states that, for a given set of (infinite) runs of a Markov chain, if we gradually observe a random run of the chain, we will become more and more certain whether the random run belongs to that set. This law has a strong implications for tail objectives. It asserts that on almost all runs $s_0 s_1 s_2 \cdots$ the limit of the value of s_i w.r.t. a tail objective tends to either 0 or 1.

In each iteration $e \in \{2, \cdots, e_{\max}\}$, we transform an objective ψ_{e-2} to a next objective ψ_e where $\psi_0 \stackrel{\text{def}}{=} \varphi$ is the parity objective and the result of the last transformation is $\psi_{e_{\max}} = \bigvee_{e \in even(\mathcal{C})} \mathsf{F}\mathit{core}_e$. We will also move from the MDP $\mathcal{L}_{e-2}$ to $\mathcal{L}_e$ after the fixings so as to maintain the following **invariant**: For all $\ell_0 \in L_0$, the value of ℓ_0 for ψ_e in $\mathcal{L}_e$ is almost as high as its value for φ in $\mathcal{L}$, that is

$$\mathtt{val}_{\mathcal{L}_e,\psi_e}(\ell_0) \geq \mathtt{val}_{\mathcal{L},\varphi}(\ell_0) - e \cdot \gamma. \tag{3}$$

Recall that $\varphi = \bigvee_{e \in even(\mathcal{C})} \mathsf{F}\varphi_e$. Let $\mathsf{Fix}_0 \stackrel{\text{def}}{=} \emptyset$ and write $\mathsf{Fix}_e \stackrel{\text{def}}{=} \bigcup_{e' \leq e} \mathit{Cl}(\mathit{fix}_{e'})$ for $e \in \{2, 4, \cdots, e_{\max}\}$. We define:

$$\psi_0 \stackrel{\text{def}}{=} \bigvee_{e'>0} \mathsf{F}\varphi_{e'} \wedge \mathsf{G}\,\neg\mathsf{Fix}_0 = \varphi \qquad\qquad \psi_e \stackrel{\text{def}}{=} \bigvee_{e' \leq e} \mathsf{F}\mathit{core}_{e'} \vee \bigvee_{e'>e} (\mathsf{F}\varphi_{e'} \wedge \mathsf{G}\,\neg\mathsf{Fix}_e). \tag{4}$$

At each transformation, we examine the disjunct $\chi_e \stackrel{\text{def}}{=} \mathsf{F}\varphi_e \wedge \mathsf{G}\neg\mathsf{Fix}_{e-2}$ in ψ_{e-2}. The set of runs satisfying this objective χ_e not only win through color e but also avoid the previously fixed regions. Roughly speaking, the aim is to transform χ_e to $\mathsf{F}\mathit{core}_e$, to move from ψ_{e-2} to ψ_e. We apply Lévy's zero-one law to deduce that the runs satisfying the χ_e are likely to enter a region that has a high value for a slightly simpler objective, namely

$$\theta_e \stackrel{\text{def}}{=} \varphi_e \wedge \mathsf{G}\,\neg\mathsf{Fix}_{e-2}. \tag{5}$$

To do so, we observe in $\mathcal{L}_{e-2}$ the behavior of several arbitrary $\frac{\gamma}{2}$-optimal strategies σ_{ℓ_0} for ψ_{e-2}, one for each $\ell_0 \in L_0$. Then, for each σ_{ℓ_0}, we apply Lévy's zero-one law separately; this provides that there exists a finite set R_e of states that have a high value for θ_e, and is reached by one of the σ_{ℓ_0} with probability as high as the probability of satisfying the disjunct χ_e. Now we use our previous results [12] on the strategy complexity of Büchi objectives and prove the existence of an MD strategy τ_e that is almost optimal for θ_e (error less than γ), starting from every state in R_e. We define sets fix_e and core_e to be the set of states from which τ_e attains a high probability for θ_e in $\mathcal{L}_{e-2}$; see Figure 3. Define $\beta \stackrel{\text{def}}{=} 1-\gamma$ and $\alpha \stackrel{\text{def}}{=} 1-\gamma^2$, and

$$\mathit{fix}_e \stackrel{\text{def}}{=} \mathit{Safe}_{\mathcal{L}_{e-2},\tau_e,\theta_e}(\beta) \qquad\qquad \mathit{core}_e \stackrel{\text{def}}{=} \mathit{Safe}_{\mathcal{L}_{e-2},\tau_e,\theta_e}(\alpha). \tag{6}$$

We fix the strategy τ_e in the fix_e-region to derive the MDP $\mathcal{L}_e$ from $\mathcal{L}_{e-2}$. Formally,

$$\mathcal{L}_e \stackrel{\text{def}}{=} \mathcal{L}_{e-2}[\tau_e, \mathit{fix}_e]. \tag{7}$$

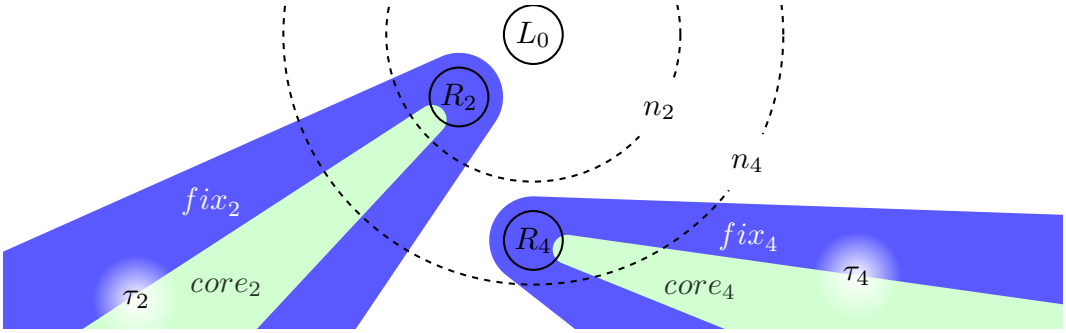


Figure 3 The construction for Lemma 9. In the first iteration, for color 2, we fix the MD strategy τ_2 in the fix_2-region. In the second iteration, for color 4, we fix τ_4 in fix_4, and so on for all even colors. Everywhere else we fix an γ-optimal reachability strategy towards $\bigcup_{e=2}^{e_{\max}} core_e$ (in green).

Iteration $e \in \{2, \cdots, e_{\max}\}$. For all states $\ell_0 \in L_0$, let σ_{ℓ_0} be a general (not necessarily MD) $\frac{\gamma}{2}$-optimal strategy w.r.t. ψ_{e-2} in the layered MDP $\mathcal{L}_{e-2}$. Consider the Markov chain $\mathcal{C}_{\ell_0}$ induced by $\mathcal{L}_{e-2}$, the fixed initial state ℓ_0 and strategy σ_{ℓ_0}.

By definition (Equation 5), θ_e is suffix-closed and $\mathsf{F}\theta_e$ is tail. The strategy σ_{ℓ_0} attains $\mathsf{F}\theta_e$ with probability at least as large as it achieves disjunct χ_e in ψ_{e-2}. We apply Lévy's zero-one law to deduce that the winning runs of $\mathsf{F}\theta_e$ likely reach a finite set R_e of states that have a high value for θ_e. In other words, most runs that eventually win through color e, while eventually avoiding Fix_{e-2}, will reach R_e within a bounded number of steps.

▶ **Lemma 10.** *Let $s_0 \in S$ and $\mathcal{E}$ be a suffix-closed objective. For all $\varepsilon, \varepsilon' > 0$, there exist n and a finite set $F \subseteq Safe_{\mathcal{E}}(1-\varepsilon)$ such that $\mathbb{P}_{s_0}(\mathsf{F}\mathcal{E} \wedge \mathsf{F}^{\leq n} F) \geq \mathbb{P}_{s_0}(\mathsf{F}\mathcal{E}) - \varepsilon'$.*

By Lemma 10, there exist n_{ℓ_0} and a finite set $R_{\ell_0} \subseteq Safe_{\mathcal{L}_{e-2},\theta_e}(\alpha)$ such that

$$\mathbb{P}_{\mathcal{L}_{e-2},\ell_0,\sigma_{\ell_0}}(\mathsf{F}\theta_e \wedge \mathsf{F}^{\leq n_{\ell_0}} R_{\ell_0}) \geq \mathbb{P}_{\mathcal{L}_{e-2},\ell_0,\sigma_{\ell_0}}(\mathsf{F}\theta_e) - \frac{\gamma}{2}. \tag{8}$$

Define $n_e \stackrel{\text{def}}{=} \max_{\ell_0 \in L_0}(n_{\ell_0})$ and $R \stackrel{\text{def}}{=} \bigcup_{\ell_0 \in L_0} R_{\ell_0}$. Write $R_e \stackrel{\text{def}}{=} \{(s,0) \mid \exists b \cdot (s,b) \in R\}$ for the projection of R_e on the layer 0.

▶ Remark 11. Suppose $\mathcal{E}' \subseteq \mathcal{E}$ and $\varepsilon > 0$ are such that $\mathbb{P}(\mathcal{E}') \geq \mathbb{P}(\mathcal{E}) - \varepsilon$. Then, for any $\mathcal{R}$, we have $\mathbb{P}(\mathcal{E}' \cap \mathcal{R}) \geq \mathbb{P}(\mathcal{E} \cap \mathcal{R}) - \varepsilon$.

Proof. We have:

$$\mathbb{P}(\mathcal{E}' \cap \mathcal{R}) = \mathbb{P}(\mathcal{E}') - \mathbb{P}(\mathcal{E}' \setminus \mathcal{R}) \geq \mathbb{P}(\mathcal{E}) - \varepsilon - \mathbb{P}(\mathcal{E}' \setminus \mathcal{R}) \geq \mathbb{P}(\mathcal{E}) - \varepsilon - \mathbb{P}(\mathcal{E} \setminus \mathcal{R}) = \mathbb{P}(\mathcal{E} \cap \mathcal{R}) - \varepsilon.$$

◀

We apply Remark 11 to Equation (8) to get

$$\mathbb{P}_{\mathcal{L}_{e-2},\ell_0,\sigma_{\ell_0}}(\mathsf{F}\theta_e \wedge \mathsf{G}\,\neg\mathsf{Fix}_{e-2} \wedge \mathsf{F}\,Cl(R_e)) \geq \mathbb{P}_{\mathcal{L}_{e-2},\ell_0,\sigma_{\ell_0}}(\mathsf{F}\theta_e \wedge \mathsf{G}\,\neg\mathsf{Fix}_{e-2}) - \frac{\gamma}{2}.$$

Since $\mathsf{FG}\,\neg\mathsf{Fix}_{e-2} \wedge \mathsf{G}\,\neg\mathsf{Fix}_{e-2} = \mathsf{G}\,\neg\mathsf{Fix}_{e-2}$ and $\chi_e = \mathsf{F}\varphi_e \wedge \mathsf{G}\neg\mathsf{Fix}_{e-2}$,

$$\mathbb{P}_{\mathcal{L}_{e-2},\ell_0,\sigma_{\ell_0}}(\chi_e \wedge \mathsf{F}\,Cl(R_e)) \geq \mathbb{P}_{\mathcal{L}_{e-2},\ell_0,\sigma_{\ell_0}}(\chi_e) - \frac{\gamma}{2}. \tag{9}$$

We think of $\mathsf{GF}[S]^{Col=e}$ as a Büchi condition on a slightly modified MDP. This allows us to apply the following theorem from [12] about the strategy complexity of Büchi objectives.

▶ **Theorem 12** (Theorem 5 in [12]). *For every acyclic countable MDP $\mathcal{M}$, a Büchi objective φ, finite set I of initial states and $\varepsilon > 0$, there exists a deterministic 1-bit strategy that is ε-optimal from every $s \in I$.*

Using Theorem 12, we prove the following.

▷ Claim 13. In MDP $\mathcal{L}_{e-2}$, there is an MD strategy τ_e, that is $(\alpha - \beta)$-optimal for θ_e from R_e.

Notice that τ_e is used to define regions $\mathit{core}_e \subseteq \mathit{fix}_e$; see Equation (6) and Figure 3. Since $\mathtt{val}_{\mathcal{L}_{e-2},\theta_e}(\ell) = \mathtt{val}_{\mathcal{L}_{e-2},\theta_e}(\ell')$ holds for all siblings ℓ and ℓ', all states in R_e have value $\geq \alpha$ w.r.t. θ_e. We have chosen τ_e to be $(\alpha - \beta)$-optimal, which implies $\mathbb{P}_{\mathcal{L}_{e-2},\ell,\tau_e}(\theta_e) \geq \beta$ for all $\ell \in R_e$. This shows that $R_e \subseteq \mathit{fix}_e$. Strategy τ_e is also used to obtain $\mathcal{L}_e$ from $\mathcal{L}_{e-2}$: for all controlled states $\ell \in \mathit{fix}_e$, the successor is fixed to be $\tau_e(\ell)$ in $\mathcal{L}_e$, see Equation (7).

Invariant (3)**.** Given a state $\ell_0 \in L_0$, this invariant states that, for all colors e, $\mathtt{val}_{\mathcal{L}_e,\psi_e}(\ell_0) \geq \mathtt{val}_{\mathcal{L},\varphi}(\ell_0) - e \cdot \gamma$ holds. Recall that $\psi_0 = \varphi$ and $\mathcal{L}_0 = \mathcal{L}$. To prove the invariant, by an induction on even colors e, it suffices to prove the following:

$$\mathtt{val}_{\mathcal{L}_e,\psi_e}(\ell_0) \geq \mathtt{val}_{\mathcal{L}_{e-2},\psi_{e-2}}(\ell_0) - 2\gamma.$$

We construct a strategy π for ψ_e in $\mathcal{L}_e$ such that $\mathbb{P}_{\mathcal{L}_e,\ell_0,\pi}(\psi_e) \geq \mathtt{val}_{\mathcal{L}_{e-2},\psi_{e-2}}(\ell_0) - 2\gamma$. Intuitively speaking, π enforces that most runs that win through colors e', with $e' \leq e$, eventually reach the $\mathit{core}_{e'}$-region and most remaining winning runs always avoid the Fix_e-region.

The strategy π is defined by combining σ_{ℓ_0} and τ_e; recall that the strategy σ_{ℓ_0} is $\frac{\gamma}{2}$-optimal w.r.t. ψ_{e-2} starting from ℓ_0 in $\mathcal{L}_{e-2}$. We define π such that it starts by following σ_{ℓ_0}. If it ever enters $\mathit{Cl}(\mathit{fix}_e)$ then we ensure that it enters fix_e as well (in at most one more step). Then π continues by playing as τ_e does forever.

The following claim concludes the proof of **Invariant** (3).

▷ Claim 14. $\mathbb{P}_{\mathcal{L}_e,\ell_0,\pi}(\psi_e) \geq \mathtt{val}_{\mathcal{L}_{e-2},\psi_{e-2}}(\ell_0) - 2\gamma$.

We summarize the main steps in the proof of Claim 14 here. We first prove the claim that if π ever enters $\mathit{Cl}(\mathit{fix}_e)$ then it is possible to define it in such a way that it actually enters fix_e.

Comparing ψ_e with ψ_{e-2}, one notices that two significant terms in the symmetric difference of these two objectives are χ_e and $\mathsf{F}\mathit{core}_e$. Roughly speaking, we use Equation (9) to move from χ_e to $\mathsf{F}\mathit{Cl}(\mathit{fix}_e)$. Then we move from $\mathsf{F}\mathit{Cl}(\mathit{fix}_e)$ to $\mathsf{F}\mathit{core}_e$ by proving that $\mathbb{P}_{\mathcal{L}_e,\ell_0,\pi}(\mathsf{F}\mathit{core}_e)$ is almost as high as $\mathbb{P}_{\mathcal{L}_{e-2},\ell_0,\pi}(\mathsf{F}\mathit{Cl}(\mathit{fix}_e))$, modulo small errors. To derive the latter, we rely on two facts: another application of Lévy's zero-one law that guarantees $\mathbb{P}_{\mathcal{L}_e,\ell_0,\pi}(\theta_e \wedge \mathsf{F}\mathit{core}_e)$ is equal to $\mathbb{P}_{\mathcal{L}_e,\ell_0,\pi}(\theta_e)$; and the fact that, as soon as π visits the first state $\ell \in \mathit{fix}_e$, it switches to τ_e forever, and thus attains θ_e with probability at least β.

Reach iteration. After all $\frac{e_{\max}}{2}$-iterations for even colors and the fixing, by **Invariant** (3), for all $\ell_0 \in L_0$, we have:

$$\mathtt{val}_{\mathcal{L}_{e_{\max}},\psi_{e_{\max}}}(\ell_0) \geq \mathtt{val}_{\mathcal{L},\varphi}(\ell_0) - e_{\max}\gamma. \tag{10}$$

Recall that $\psi_{e_{\max}} = \bigvee_{e \in \mathit{even}(\mathcal{C})} \mathsf{F}\mathit{core}_e$. At this last iteration, we fix the choice of all remaining states in $\mathcal{L}_{e_{\max}}$ such that the probability of $\psi_{e_{\max}}$ is maximized. Recall that there are uniformly ε-optimal MD strategies for reachability objectives [16]. Hence, there is a single MD strategy τ_{reach} in $\mathcal{L}_{e_{\max}}$ that is uniformly γ-optimal w.r.t. $\psi_{e_{\max}}$; in particular, τ_{reach} is γ-optimal from every state $\ell_0 \in L_0$.

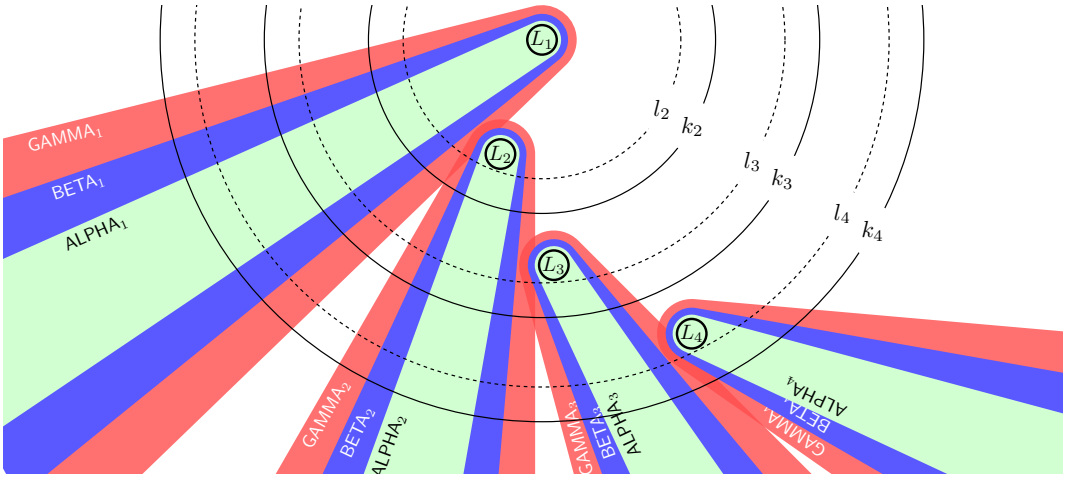


Figure 4 Initial segment of the sea urchin construction. $\mathcal{L}_i$ is the result of fixing τ_i inside BETA_i and then ρ_i inside the k_i-bubble (the set of states reachable from the initial state(s) in $\leq k_i$ steps). Drawn here for $i = 1, 2, 3, 4$.

Let $\mathcal{L}' \stackrel{\text{def}}{=} \mathcal{L}_{e_{\max}}[\tau_{\text{reach}}, L]$. Let $\hat{\sigma}$ be the MD strategy in $\mathcal{L}$ that plays from L_0 as prescribed by all the fixings in $\mathcal{L}'$. Since all choices in all the fix_e-region are resolved according to τ_e, $e \in \{2, \cdots, e_{\max}\}$, we can apply Lévy's zero-one law another time.

▶ **Lemma 15.** *Let $0 < \beta_1 < \beta_2 \leq 1$ and $\mathcal{E}$ a tail objective. For $s \in \mathit{Safe}_{\mathcal{E}}(\beta_2)$, the following holds: $\mathbb{P}_s(\mathsf{G}\,\mathit{Safe}_{\mathcal{E}}(\beta_1)) \geq \frac{\beta_2 - \beta_1}{1 - \beta_1}$.*

By Lemma 15, for all states $\ell \in \mathit{core}_e$,

$$\mathbb{P}_{\mathcal{L}_{e_{\max}},\ell,\tau_e}(\mathsf{G}\mathit{fix}_e) \geq \frac{\alpha - \beta}{1 - \beta} \geq 1 - \gamma. \tag{11}$$

States in fix_e have a high value for θ_e and thus also for $\mathsf{F}\varphi_e$.

▶ **Lemma 16.** *Let $0 < \beta < 1$ and $\mathcal{E}$ a tail objective. For all states $s \in \mathit{Safe}_{\mathcal{E}}(\beta)$:*

1. $\mathbb{P}_s(\mathit{FGSafe}_{\mathcal{E}}(\beta) \setminus \mathcal{E}) = 0$*; and*
2. $\mathbb{P}_s(\mathcal{E} \setminus \mathit{FGSafe}_{\mathcal{E}}(\beta)) = 0$.

By Lemma 16.2, we satisfy $\mathsf{F}\varphi_e$ almost surely:

$$\mathbb{P}_{\mathcal{L}_{e_{\max}},\ell,\tau_e}(\mathsf{F}\varphi_e \mid \mathsf{G}\mathit{fix}_e) = 1. \tag{12}$$

Using Equations (10) and (11), we prove the following.

▷ Claim 17. The MD strategy $\hat{\sigma}$ is ε-optimal for parity objective φ, from every state $\ell_0 \in L_0$.

This concludes the proof of Lemma 9.

4 Optimal Strategies for Parity

In this section we show Theorem 2, i.e., that optimal strategies for parity, where they exist, can be chosen deterministic 1-bit Markov.

First we show the main technical result of this section.

▶ **Lemma 18.** *Let $\mathcal{L}(\mathcal{M})$ be the layered MDP obtained from an acyclic and finitely branching MDP $\mathcal{M}$ and a coloring Col such that all states are almost surely winning for $\varphi = \texttt{Parity}(Col)$ (i.e., every state s has a strategy σ_s such that $\mathbb{P}_{\mathcal{L}(\mathcal{M}),s,\sigma_s}(\varphi) = 1$).*

For every initial state s_0 there exists an MD strategy σ that almost surely wins, i.e., $\mathbb{P}_{\mathcal{L}(\mathcal{M}),s_0,\sigma}(\varphi) = 1$.

Proof sketch. For a complete proof we refer the reader to the technical report [13].

For some intuition consider Figure 4. The sea urchin construction is a plastering construction with infinitely many iterations where MD strategies are fixed in larger and larger subspaces. Its name comes from the shape of the subspace in which player choices are fixed up-to iteration i: A growing finite body of states that are reachable from the initial state s_0 within $\leq k_i$ steps, plus i different spikes of infinite size. Each spike is composed of nested subsets $\mathsf{ALPHA}_i \subseteq \mathsf{BETA}_i$ (and $\subseteq \mathsf{GAMMA}_i$, which is used only in the correctness argument) that correspond to different levels of attainment of certain ε-optimal MD strategies τ_i, obtained from Lemma 9. Strategy τ_i is then fixed in BETA_i (and thus in ALPHA_i). Other MD strategies ρ_i are fixed elsewhere in the finite body, up-to horizon k_i. Using Lévy's zero-one law, we prove that, once inside ALPHA_i, there is a high chance of never leaving the i-th spike BETA_i. Moreover, almost all runs that stay in the i-th spike satisfy parity. Finally, the strategies ρ_i ensure that at least $1/2$ (by probability mass) of the runs from s_0 that don't stay in one of the first i spikes will eventually stay in the $(i+1)$-th spike and satisfy parity there. Thus, at the stage with i spikes, the fixed MD strategy attains parity with some probability $\geq 1 - 2^{-i}$ already *inside* this fixed subspace. In the limit of $i \to \infty$, the resulting MD strategy attains parity almost surely. ◀

▶ **Definition 19.** *For a tail objective φ and an MDP $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$, we define the* conditioned version *of $\mathcal{M}$ w.r.t. φ to be the MDP $\mathcal{M}_* = (S_*, S_{*\Box}, S_{*\bigcirc}, \longrightarrow_*, P_*)$ with $S_* = \{s \in S \mid \exists \sigma.\, \mathbb{P}_{\mathcal{M},s,\sigma}(\varphi) = \texttt{val}_{\mathcal{M}}(s) > 0\}$ and $S_{*\Box} = S_* \cap S_\Box$ and $S_{*\bigcirc} = S_* \cap S_\bigcirc$ and*

$$\longrightarrow_* = \{(s,t) \in S_* \times S_* \mid s \longrightarrow t \text{ and if } s \in S_{*\Box} \text{ then } \texttt{val}_{\mathcal{M}}(s) = \texttt{val}_{\mathcal{M}}(t)\}$$

and $P_* : S_{*\bigcirc} \to \mathcal{D}(S_*)$ so that $P_*(s)(t) = P(s)(t) \cdot \frac{\texttt{val}_{\mathcal{M}}(t)}{\texttt{val}_{\mathcal{M}}(s)}$ for all $s \in S_{*\bigcirc}$ and $t \in S_*$ with $s \longrightarrow_* t$.

A proof that $P_*(s)$ is a probability distribution for all $s \in S_{*\bigcirc}$ and therefore that $\mathcal{M}_*$ is well-defined, can be found in the full paper [13], Appendix C. The name "conditional MDP" stems from a useful property that for all strategies that are optimal for φ in $\mathcal{M}$, the probability in $\mathcal{M}_*$ of any event is the same as that of its probability in $\mathcal{M}$ conditioned under φ.

The following theorem is a very slight generalization of [14, Theorem 5]. It gives a sufficient condition under which we can conclude the existence of MD optimal strategies from the existence of MD almost-sure winning strategies.

▶ **Theorem 20.** *Let φ be a tail objective. Let $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ be an MDP and $\mathcal{M}_* = (S_*, S_{*\Box}, S_{*\bigcirc}, \longrightarrow_*, P_*)$ its conditioned version wrt. φ. Then:*

1. *For all $s \in S_*$ there exists a strategy σ with $\mathbb{P}_{\mathcal{M}_*,s,\sigma}(\varphi) = 1$.*
2. *Suppose that for every $s \in S_*$ there exists an MD strategy σ'' with $\mathbb{P}_{\mathcal{M}_*,s,\sigma''}(\varphi) = 1$. Then there is an MD strategy σ' such that for all $s \in S$:*

$$\big(\exists \sigma \in \Sigma.\, \mathbb{P}_{\mathcal{M},s,\sigma}(\varphi) = \texttt{val}_{\mathcal{M}}(s)\big) \implies \mathbb{P}_{\mathcal{M},s,\sigma'}(\varphi) = \texttt{val}_{\mathcal{M}}(s)$$

▶ **Theorem 21.** *Consider an acyclic MDP $\mathcal{M}$ and a parity objective.*

There exists a deterministic 1-bit strategy that is optimal from all states that have an optimal strategy.

Proof. Consider the corresponding layered system $\mathcal{L}(\mathcal{M})$ (cf. Definition 6), which is also acyclic. Let S_{opt} be the subset of states that have an optimal strategy in $\mathcal{M}$. Thus all states in $S_{opt} \times \{0,1\}$ have an optimal strategy in $\mathcal{L}(\mathcal{M})$ by Lemma 7.

We now use Theorem 20 to obtain an MD strategy σ' in $\mathcal{L}(\mathcal{M})$ that is optimal for all states in $\mathcal{L}(\mathcal{M})$ that have an optimal strategy. First, the parity objective is tail. Second, in $\mathcal{L}(\mathcal{M})$, any two siblings have the same value w.r.t. parity by Remark 8. Therefore the changes from $\mathcal{L}(\mathcal{M})$ to its conditioned version $\mathcal{L}(\mathcal{M})_*$ (wrt. the parity objective) are symmetric in the two layers. Thus $\mathcal{L}(\mathcal{M})_*$ is also a layered acyclic MDP (i.e., there exists some acyclic MDP $\mathcal{M}'$ s.t. $\mathcal{L}(\mathcal{M})_* = \mathcal{L}(\mathcal{M}')$), and by Theorem 20.1 all states in $\mathcal{L}(\mathcal{M})_*$ are almost surely winning. Now we can apply Lemma 18 (generalized to infinitely branching acyclic layered MDPs by Lemma 3) to $\mathcal{L}(\mathcal{M})_*$ and obtain that for every state in $\mathcal{L}(\mathcal{M})_*$ there is an MD strategy that almost surely wins. By Theorem 20.2 there is an MD strategy σ' in $\mathcal{L}(\mathcal{M})$ that is optimal for all states that have an optimal strategy. In particular, σ' is optimal for the states in $S_{opt} \times \{0,1\}$ in $\mathcal{L}(\mathcal{M})$. By Lemma 7, this yields a deterministic 1-bit strategy in $\mathcal{M}$ that is optimal for all states in S_{opt}. ◀

In Theorem 21 the initial memory mode of the 1-bit strategy is irrelevant (recall Remark 8).

Theorem 2 now follows directly from Theorem 21 and Lemma 4(3).

5 Optimal Strategies for $\{0,1,2\}$-Parity

▶ **Theorem 22.** *Let $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ be an MDP, φ a $\{0,1,2\}$-Parity objective and $\mathcal{M}_* = (S_*, S_{*\Box}, S_{*\bigcirc}, \longrightarrow_*, P_*)$ its conditioned version wrt. φ. Assume that in $\mathcal{M}_*$ for every safety objective (given by some target $T \subseteq S_*$) and $\varepsilon > 0$ there exists a uniformly ε-optimal MD strategy. Let S_{opt} be the subset of states that have an optimal strategy for φ in $\mathcal{M}$.*

Then there exists an MD strategy in $\mathcal{M}$ that is optimal for φ from every state in S_{opt}.

The above result generalizes [14, Theorem 16], which considers only finitely-branching MDPs and uses the fact that for every safety objective, an MD strategy exists that is uniformly *optimal*. This is not generally true for infinitely-branching acyclic MDPs [14]. To prove Theorem 22, we adjust the construction so that it only requires uniformly ε-*optimal* MD strategies for safety objectives (in the conditioned MDP $\mathcal{M}_*$).

In order to apply Theorem 22 to infinitely-branching acyclic MDPs, we now show that acyclicity guarantees the existence of uniformly ε-optimal MD strategies for safety objectives.

▶ **Lemma 23.** *For every acyclic MDP with a safety objective and every $\varepsilon > 0$ there exists an MD strategy that is uniformly ε-optimal.*

While we defined ε-optimality wrt. additive errors (cf. Section 2), our proof of Lemma 23 shows that the claim holds even wrt. multiplicative errors (in the style of [16]).

▶ **Theorem 24.** *Consider an MDP $\mathcal{M}$ with a $\{0,1,2\}$-Parity objective and let S_{opt} be the subset of states that have an optimal strategy.*

1. *If $\mathcal{M}$ is acyclic then there exists an MD strategy that is optimal from every state in S_{opt}.*
2. *There exists a deterministic Markov strategy that is optimal from every state in S_{opt}.*

Proof. Towards item 1, if $\mathcal{M}$ is acyclic then also its conditioned version $\mathcal{M}_*$ (with respect to $\{0,1,2\}$-Parity) is acyclic. Thus, by Lemma 23, in $\mathcal{M}_*$ for every $\varepsilon > 0$ and every safety objective there is a uniformly ε-optimal MD strategy. The result now follows from Theorem 22.

Item 2 follows from Item 1 and Lemma 4 (item 3 with $k = 0$). ◀

6 ε-Optimal Strategies for $\{0,1\}$-Parity (co-Büchi)

▶ **Theorem 25.** *Suppose that $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ is an MDP such that for every safety objective (given by some target $T \subseteq S$) and $\varepsilon > 0$ there exists a uniformly ε-optimal MD strategy.*

Then for every co-Büchi objective (given by some coloring $Col : S \to \{0,1\}$) and $\varepsilon > 0$ there exists a uniformly ε-optimal MD strategy.

The precondition of Theorem 25 is satisfied by many classes of MDPs. Indeed, we obtain the following.

▶ **Corollary 26.** *Consider an MDP $\mathcal{M}$ and a co-Büchi objective.*

1. *If $\mathcal{M}$ is acyclic then, for every $\varepsilon > 0$, there exists a uniformly ε-optimal MD strategy.*
2. *If $\mathcal{M}$ is finitely branching then, for every $\varepsilon > 0$, there exists a uniformly ε-optimal MD strategy.*
3. *For every $\varepsilon > 0$ there exists a deterministic Markov strategy that, from every initial state s, attains at least $\mathtt{val}_{\mathcal{M}}(s) - \varepsilon$.*

Proof. Towards (1), for acyclic MDPs, uniformly ε-optimal strategies for safety can be chosen MD by Lemma 23. Towards (2), for finitely branching MDPs there always exists even a uniformly optimal MD strategy for every safety objective. In both cases the claim then follows from Theorem 25. Claim (3) follows directly from (1) and Lemma 4 (item 2 with $k = 0$). ◀

References

1 P. Abbeel and A. Y. Ng. Learning first-order Markov models for control. In *Advances in Neural Information Processing Systems 17*. MIT Press, 2004. URL: http://papers.nips.cc/paper/2569-learning-first-order-markov-models-for-control.

2 C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

3 P. Billingsley. *Probability and Measure*. Wiley, 1995. Third Edition.

4 V. D. Blondel and J. N. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 2000.

5 N. Bäuerle and U. Rieder. *Markov Decision Processes with Applications to Finance*. Springer-Verlag Berlin Heidelberg, 2011.

6 K. Chatterjee, M. Jurdziński, and T. Henzinger. Quantitative stochastic parity games. In *Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2004. URL: http://dl.acm.org/citation.cfm?id=982792.982808.

7 E. M. Clarke, T. A. Henzinger, H. Veith, and R. Bloem, editors. *Handbook of Model Checking*. Springer, 2018. doi:10.1007/978-3-319-10575-8.

8 E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, December 1999.

9 E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games*, LNCS, 2002.

10 T. P. Hill. On the existence of good Markov strategies. *Transactions of the American Mathematical Society*, 1979. doi:10.1090/S0002-9947-1979-0517690-9.

11 T. P. Hill. Goal problems in gambling theory. *Revista de Matemática: Teoría y Aplicaciones*, 1999.

12 S. Kiefer, R. Mayr, M. Shirmohammadi, and P. Totzke. Büchi objectives in countable MDPs. In *International Colloquium on Automata, Languages and Programming*. LIPIcs, 2019. A technical report is available at arXiv:1904.11573. doi:10.4230/LIPIcs.ICALP.2019.119.

13 S. Kiefer, R. Mayr, M. Shirmohammadi, and P. Totzke. Strategy Complexity of Parity Objectives in Countable MDPs. *CoRR*, 2020. arXiv:2007.05065.

14 S. Kiefer, R. Mayr, M. Shirmohammadi, and D. Wojtczak. Parity objectives in countable MDPs. In *Annual IEEE Symposium on Logic in Computer Science*, 2017.

15 A. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Computation Theory*, LNCS, 1984.

16 D. Ornstein. On the existence of stationary optimal strategies. *Proceedings of the American Mathematical Society*, 1969.

17 M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1st edition, 1994.

18 M. Schäl. Markov decision processes in finance and dynamic options. In *Handbook of Markov Decision Processes*. Springer, 2002.

19 O. Sigaud and O. Buffet. *Markov Decision Processes in Artificial Intelligence*. John Wiley & Sons, 2013.

20 R.S. Sutton and A.G Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, 2018.

21 M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. of FOCS'85*, 1985.

22 W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 1998.

23 W. Zielonka. Perfect-information stochastic parity games. In *Foundations of Software Science and Computation Structures*, LNCS. Springer, 2004.

Monte Carlo Tree Search Guided by Symbolic Advice for MDPs

Damien Busatto-Gaston
Université Libre de Bruxelles, Brussels, Belgium
damien.busatto-gaston@ulb.ac.be

Debraj Chakraborty
Université Libre de Bruxelles, Brussels, Belgium
debraj.chakraborty@ulb.ac.be

Jean-Francois Raskin
Université Libre de Bruxelles, Brussels, Belgium
jraskin@ulb.ac.be

Abstract

In this paper, we consider the online computation of a strategy that aims at optimizing the expected average reward in a Markov decision process. The strategy is computed with a receding horizon and using Monte Carlo tree search (MCTS). We augment the MCTS algorithm with the notion of symbolic advice, and show that its classical theoretical guarantees are maintained. Symbolic advice are used to bias the selection and simulation strategies of MCTS. We describe how to use QBF and SAT solvers to implement symbolic advice in an efficient way. We illustrate our new algorithm using the popular game Pac-Man and show that the performances of our algorithm exceed those of plain MCTS as well as the performances of human players.

2012 ACM Subject Classification Theory of computation → Theory of randomized search heuristics; Theory of computation → Convergence and learning in games

Keywords and phrases Markov decision process, Monte Carlo tree search, symbolic advice, simulation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.40

Supplementary Material `http://di.ulb.ac.be/verif/debraj/pacman/`

Funding This work is partially supported by the ARC project Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond (Fédération Wallonie-Bruxelles), the EOS project Verifying Learning Artificial Intelligence Systems (F.R.S.-FNRS & FWO), and the COST Action 16228 GAMENET (European Cooperation in Science and Technology).

Acknowledgements Computational resources have been provided by the Consortium des Équipements de Calcul Intensif (CÉCI), funded by the Fonds de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under Grant No. 2.5020.11.

1 Introduction

Markov decision processes (MDP) are an important mathematical formalism for modeling and solving sequential decision problems in stochastic environments [23]. The importance of this model has triggered a large number of works in different research communities within computer science, most notably in formal verification, and in artificial intelligence and machine learning. The works done in these research communities have respective weaknesses and complementary strengths. On the one hand, algorithms developed in formal verification are generally complete and provide strong guarantees on the optimality of computed solutions but they tend to be applicable to models of moderate size only. On the other hand, algorithms developed in artificial intelligence and machine learning usually scale to larger models but only provide weaker guarantees. Instead of opposing the two sets of algorithms, there have

been recent works [2, 14, 6, 12, 11, 19, 1] that try to combine the strengths of the two approaches in order to offer new hybrid algorithms that scale better and provide stronger guarantees. The contributions described in this paper are part of this research agenda: we show how to integrate *symbolic advice* defined by formal specifications into Monte Carlo Tree Search algorithms [7] using techniques such as SAT [21] and QBF [24].

When an MDP is too large to be analyzed *offline* using verification algorithms, receding horizon analysis combined with simulation techniques are used *online* [17]. Receding horizon techniques work as follows. In the current state s of the MDP, for a fixed horizon H, the receding horizon algorithm searches for an action a that is the first action of a plan to act (almost) optimally on the finite horizon H. When such an action is identified, then it is played from s and the state evolves stochastically to a new state s' according to the dynamics specified by the MDP. The same process is repeated from s'. The optimization criterion over the H next step depends on the long run measure that needs to be optimised. The tree unfolding from s that needs to be analyzed is often very large (*e.g.* it may be exponential in H). As a consequence, receding horizon techniques are often coupled with *sampling techniques* that avoid the systematic exploration of the entire tree unfolding at the expense of approximation. The Monte Carlo Tree Search (MCTS) algorithm [7] is an increasingly popular tree search algorithm that implements these ideas. It is one of the core building blocks of the AlphaGo algorithm [25].

While MCTS techniques may offer reasonable performances out of the shelf, they usually need substantial adjustments that depend on the application to really perform well. One way to adapt MCTS to a particular application is to *bias* the search towards promising subspaces taking into account properties of the application domain [16, 26]. This is usually done by coding directly handcrafted search and sampling strategies. We show in this paper how to use techniques from formal verification to offer a *flexible* and *rigorous* framework to bias the search performed by MCTS using *symbolic advice*. A symbolic advice is a *formal specification*, that can be expressed for example in your favorite linear temporal logic, and which constrain the search and the sampling phases of the MCTS algorithm using QBF and SAT solvers. Our framework offers in principle the ability to easily experiment with precisely formulated bias expressed declaratively using logic.

Contributions. On the theoretical side, we study the impact of using symbolic advice on the guarantees offered by MCTS. We identify sufficient conditions for the symbolic advice to preserve the convergence guarantees of the MCTS algorithm (Theorem 21). Those results are partly based on an analysis of the incidence of sampling on those guarantees (Theorem 11) which can be of independent interest.

On a more practical side, we show how symbolic advice can be implemented using SAT and QBF techniques. More precisely, we use QBF [22] to force that all the prefixes explored by the MCTS algorithm in the partial tree unfolding have the property suggested by the *selection advice* (whenever possible) and we use SAT-based sampling techniques [9] to achieve uniform sampling among paths of the MDP that satisfy the *sampling advice*. The use of this symbolic exploration techniques is important as the underlying state space that we need to analyze is usually huge (*e.g.* exponential in the receding horizon H).

To demonstrate the practical interest of our techniques, we have applied our new *MCTS with symbolic advice algorithm* to play Pac-Man. Figure 1 shows a grid of the Pac-Man game. In this version of the classical game, the agent Pac-Man has to eat food pills as fast as possible while avoiding being pinched by ghosts. We have chosen this benchmark to evaluate our algorithm for several reasons. First, the state space of the underlying MDP is

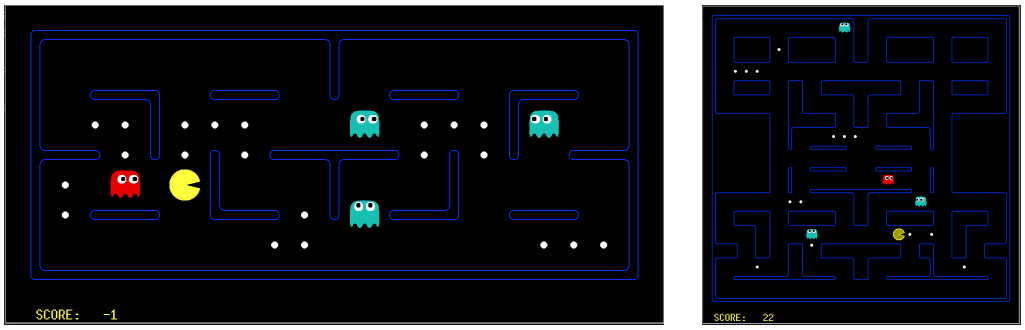


Figure 1 We used two grids of size 9×21 and 27×28 for our experiments. Pac-Man loses if he makes contact with a ghost, and wins if he eats all food pills (in white). The agents can travel in four directions unless they are blocked by the walls in the grid, and ghosts cannot reverse their direction. The score decreases by 1 at each step, and increases by 10 whenever Pac-Man eats a food pill. A win (resp. loss), increases (resp. decreases) the score by 500. The game can be seen as an infinite duration game by saying that whenever Pac-Man wins or loses, the positions of the agents and of the food pills are reset.

way too large for the state-of-the-art implementations of complete algorithms. Indeed, the reachable state space of the small grid shown here has approximately 10^{16} states, while the classical grid has approximately 10^{23} states. Our algorithm can handle both grids. Second, this application not only allows for comparison between performances obtained from several versions of the MCTS algorithm but also with the performances that humans can attain in this game. In the Pac-Man benchmark, we show that advice that instructs Pac-Man on the one hand to avoid ghosts at all costs during the selection phase of the MCTS algorithm (enforced whenever possible by QBF) and on the other hand to bias the search to paths in which ghosts are avoided (using uniform sampling based on SAT) allow to attain or surpass human level performances while the standard MCTS algorithm performs much worse.

Related works. Our analysis of the convergence of the MCTS algorithm with appropriate symbolic advice is based on extensions of analysis results based on bias defined using UCT (bandit algorithms) [18, 3]. Those results are also related to sampling techniques for finite horizon objectives in MDP [17].

Our concept of selection phase advice is related to the MCTS-minimax hybrid algorithm proposed in [4]. There the selection phase advice is not specified declaratively using logic but encoded directly in the code of the search strategy. No use of QBF nor SAT is advocated there and no use of sampling advice either. In [1], the authors provide a general framework to add safety properties to reinforcement learning algorithms via *shielding*. These techniques analyse statically the full state space of the game in order to compute a set of unsafe actions to avoid. This fits our advice framework, so that such a shield could be used as an online selection advice in order to combine their safety guarantees with our formal results for MCTS. More recently, a variation of shielding called *safe padding* has been studied in [14]. Both works are concerned with reinforcement learning and not with MCTS. Note that in general multiple ghosts may prevent the existence of a strategy to enforce safety, *i.e.* always avoid pincer moves.

Our practical handling of symbolic sampling advice relies on symbolic sampling techniques introduced in [8], while our handling of symbolic selection advice relies on natural encodings via QBF that are similar to those defined in [22].

2 Preliminaries

A *probability distribution* on a finite set S is a function $d : S \to [0,1]$ such that $\sum_{s \in S} d(s) = 1$. We denote the set of all probability distributions on set S by $\mathcal{D}(S)$. The support of a distribution $d \in \mathcal{D}(S)$ is $\mathsf{Supp}(d) = \{s \in S \mid d(s) > 0\}$.

2.1 Markov decision process

▶ **Definition 1** (MDP). *A Markov decision process is a tuple $M = (S, A, P, R, R_T)$, where S is a finite set of states, A is a finite set of actions, P is a mapping from $S \times A$ to $\mathcal{D}(S)$ such that $P(s,a)(s')$ denotes the probability that action a in state s leads to state s', $R : S \times A \to \mathbb{R}$ defines the reward obtained for taking a given action at a given state, and $R_T : S \to \mathbb{R}$ assigns a terminal reward to each state in S.*[1]

For a Markov decision process M, a *path* of length $i > 0$ is a sequence of i consecutive states and actions followed by a last state. We say that $p = s_0 a_0 s_1 \dots s_i$ is an i-length path in the MDP M if for all $t \in [0, i-1]$, $a_t \in A$ and $s_{t+1} \in \mathsf{Supp}(P(s_t, a_t))$, and we denote $\mathsf{last}(p) = s_i$ and $\mathsf{first}(p) = s_0$. We also consider states to be paths of length 0. An infinite path is an infinite sequence $p = s_0 a_0 s_1 \dots$ of states and actions such that for all $t \in \mathbb{N}$, $a_t \in A$ and $s_{t+1} \in \mathsf{Supp}(P(s_t, a_t))$. We denote the finite prefix of length t of a finite or infinite path $p = s_0 a_0 s_1 \dots$ by $p_{|t} = s_0 a_0 \dots s_t$. Let $p = s_0 a_0 s_1 \dots s_i$ and $p' = s'_0 a'_0 s'_1 \dots s'_j$ be two paths such that $s_i = s'_0$, let a be an action and s be state of M. Then, $p \cdot p'$ denotes $s_0 a_0 s_1 \dots s_i a'_0 s'_1 \dots s'_j$ and $p \cdot as$ denotes $s_0 a_0 s_1 \dots s_i a s$.

For an MDP M, the set of all finite paths of length i is denoted by Paths^i_M. Let $\mathsf{Paths}^i_M(s)$ denote the set of paths p in Paths^i_M such that $\mathsf{first}(p) = s$. Similarly, if $p \in \mathsf{Paths}^i_M$ and $i \leq j$, then let $\mathsf{Paths}^j_M(p)$ denote the set of paths p' in Paths^j_M such that there exists $p'' \in \mathsf{Paths}^{j-i}_M$ with $p' = p \cdot p''$. We denote the set of all finite paths in M by Paths_M and the set of finite paths of length at most H by $\mathsf{Paths}^{\leq H}_M$.

▶ **Definition 2.** *The total reward of a finite path $p = s_0 a_0 \dots s_n$ in M is defined as*

$$\mathsf{Reward}_M(p) = \sum_{t=0}^{n-1} R(s_t, a_t) + R_T(s_n)\,.$$

A (probabilistic) *strategy* is a function $\sigma : \mathsf{Paths}_M \to \mathcal{D}(A)$ that maps a path p to a probability distribution in $\mathcal{D}(A)$. A strategy σ is *deterministic* if the support of the probability distributions $\sigma(p)$ has size 1, it is *memoryless* if $\sigma(p)$ depends only on $\mathsf{last}(p)$, *i.e.* if σ satisfies that for all $p, p' \in \mathsf{Paths}_M$, $\mathsf{last}(p) = \mathsf{last}(p') \Rightarrow \sigma(p) = \sigma(p')$. For a probabilistic strategy σ and $i \in \mathbb{N}$, let $\mathsf{Paths}^i_M(\sigma)$ denote the paths $p = s_0 a_0 \dots s_i$ in Paths^i_M such that for all $t \in [0, i-1]$, $a_t \in \mathsf{Supp}(\sigma(p_{|t}))$. For a finite path p of length $i \in \mathbb{N}$ and some $j \geq i$, let $\mathsf{Paths}^j_M(p, \sigma)$ denote $\mathsf{Paths}^j_M(\sigma) \cap \mathsf{Paths}^j_M(p)$.

For a strategy σ and a path $p \in \mathsf{Paths}^i_M(\sigma)$, let the probability of $p = s_0 a_0 \dots s_i$ in M according to σ be defined as $\mathbb{P}^i_{M,\sigma}(p) = \prod_{t=0}^{i-1} \sigma(p_{|t})(a_t) P(s_t, a_t)(s_{t+1})$. The mapping $\mathbb{P}^i_{M,\sigma}$ defines a probability distribution over $\mathsf{Paths}^i_M(\sigma)$.

[1] We assume for convenience that every action in A can be taken from every state. One may need to limit this choice to a subset of *legal* actions that depends on the current state. This concept can be encoded in our formalism by adding a sink state reached with probability 1 when taking an illegal action.

▶ **Definition 3.** *The expected average reward of a probabilistic strategy σ in an MDP M, starting from state s, is defined as*

$$\mathsf{Val}_M(s,\sigma) = \liminf_{n\to\infty} \frac{1}{n}\mathbb{E}\left[\mathsf{Reward}_M(p)\right] ,$$

where p is a random variable over $\mathsf{Paths}^n_M(\sigma)$ following the distribution $\mathbb{P}^n_{M,\sigma}$.

▶ **Definition 4.** *The optimal expected average reward starting from a state s in an MDP M is defined over all strategies σ in M as $\mathsf{Val}_M(s) = \sup_\sigma \mathsf{Val}_M(s,\sigma)$.*

One can restrict the supremum to deterministic memoryless strategies [23, Proposition 6.2.1]. A strategy σ is called ϵ-optimal for the expected average reward if $\mathsf{Val}_M(s,\sigma) \geq \mathsf{Val}_M(s) - \epsilon$ for all s.

▶ **Definition 5.** *The expected total reward of a probabilistic strategy σ in an MDP M, starting from state s and for a finite horizon i, is defined as $\mathsf{Val}^i_M(s,\sigma) = \mathbb{E}\left[\mathsf{Reward}_M(p)\right]$, where p is a random variable over $\mathsf{Paths}^i_M(\sigma)$ following the distribution $\mathbb{P}^i_{M,\sigma}$.*

▶ **Definition 6.** *The optimal expected total reward starting from a state s in an MDP M, with horizon $i \in \mathbb{N}$, is defined over all strategies σ in M as $\mathsf{Val}^i_M(s) = \sup_\sigma \mathsf{Val}^i_M(s,\sigma)$.*

One can restrict the supremum to deterministic strategies [23, Theorem 4.4.1.b].

Let $\sigma^{i,*}_{M,s}$ denote a deterministic strategy that maximises $\mathsf{Val}^i_M(s,\sigma)$, and refer to it as an optimal strategy for the expected total reward of horizon i at state s. For $i \in \mathbb{N}$, let σ^i_M refer to a deterministic memoryless strategy that maps every state s in M to the first action of a corresponding optimal strategy for the expected total reward of horizon i, so that $\sigma^i_M(s) = \sigma^{i,*}_{M,s}(s)$. As there may exist several deterministic strategies σ that maximise $\mathsf{Val}^i_M(s,\sigma)$, we denote by $\mathsf{opt}^i_M(s)$ the set of actions a such that there exists an optimal strategy $\sigma^{i,*}_{M,s}$ that selects a from s. A strategy σ^i_M can be obtained by the value iteration algorithm:

▶ **Proposition 7** (Value iteration [23, Section 5.4])**.** *For a state s in MDP M, for all $i \in \mathbb{N}$,*

- $\mathsf{Val}^{i+1}_M(s) = \max_{a\in A}\left[R(s,a) + \sum_{s'} P(s,a)(s')\mathsf{Val}^i_M(s')\right]$
- $\mathsf{opt}^{i+1}_M(s) = \arg\max_{a\in A}\left[R(s,a) + \sum_{s'} P(s,a)(s')\mathsf{Val}^i_M(s')\right]$

Moreover, for a large class of MDPs and a large enough n, the strategy σ^n_M is ϵ-optimal for the expected average reward:

▶ **Proposition 8** ([23, Theorem 9.4.5])**.** *For a strongly aperiodic[2] Markov decision process M, it holds that $\mathsf{Val}_M(s) = \lim_{n\to\infty}[\mathsf{Val}^{n+1}_M(s) - \mathsf{Val}^n_M(s)]$. Moreover, for any $\epsilon > 0$ there exists $N \in \mathbb{N}$ such that for all $n \geq N$, $Val_M(s,\sigma^n_M) \geq Val_M(s) - \epsilon$ for all s.*

A simple transformation can be used to make an MDP strongly aperiodic without changing the optimal expected average reward and the associated optimal strategies. Therefore, one can use an algorithm computing the strategy σ^H_M in order to optimise for the expected average reward, and obtain theoretical guarantees for a horizon H big enough. This is known as the *receding horizon* approach.

Finally, we will use the notation $T(M,s_0,H)$ to refer to an MDP obtained as a tree-shaped *unfolding* of M from state s_0 and for a depth of H. In particular, the states of $T(M,s_0,H)$ correspond to paths in $\mathsf{Paths}^{\leq H}_M(s_0)$. Then, it holds that:

▶ **Lemma 9.** *$\mathsf{Val}^H_M(s_0)$ is equal to $\mathsf{Val}^H_{T(M,s_0,H)}(s_0)$, and $\mathsf{opt}^H_M(s_0)$ is equal to $\mathsf{opt}^H_{T(M,s_0,H)}(s_0)$.*

The aperiodicity and unfolding transformations are detailed in Appendix A.

[2] A Markov decision process is strongly aperiodic if $P(s,a)(s) > 0$ for all $s \in S$ and $a \in A$.

2.2 Bandit problems and UCB

In this section, we present bandit problems, whose study forms the basis of a theoretical analysis of Monte Carlo tree search algorithms.

Let A denote a finite set of actions. For each $a \in A$, let $(x_{a,t})_{t\geq 1}$ be a sequence of random payoffs associated to a. They correspond to successive plays of action a, and for every action a and every $t \geq 1$, let $x_{a,t}$ be drawn with respect to a probability distribution $\mathcal{D}_{a,t}$ over $[0,1]$. We denote by $X_{a,t}$ the random variable associated to this drawing. In a *fixed distributions* setting (the classical bandit problem), every action is associated to a fixed probability distribution $\mathcal{D}_a$, so that $\mathcal{D}_{a,t} = \mathcal{D}_a$ for all $t \geq 1$.

The *bandit problem* consists of a succession of steps where the player selects an action and observes the associated payoff, while trying to maximise the cumulative gains. For example, selecting action a, then b and then a again would yield the respective payoffs $x_{a,1}$, $x_{b,1}$ and $x_{a,2}$ for the first three steps, drawn from their respective distributions. Let the regret R_n denote the difference, after n steps, between the optimal expected payoff $\max_{a\in A} \mathbb{E}[\sum_{t=1}^{n} X_{a,t}]$ and the expected payoff associated to our action selection. The goal is to minimise the long-term regret when the number of steps n increases.

The algorithm UCB1 of [3] offers a practical solution to this problem, and offers theoretical guarantees. For an action a and $n \geq 1$, let $\overline{x}_{a,n} = \frac{1}{n}\sum_{t=1}^{n} x_{a,t}$ denote the average payoff obtained from the first n plays of a. Moreover, for a given step number t let t_a denote how many times action a was selected in the first t steps. The algorithm UCB1 chooses, at step $t+1$, the action a that maximises $\overline{x}_{a,t_a} + c_{t,t_a}$, where c_{t,t_a} is defined as $\sqrt{\frac{2\ln t}{t_a}}$. This procedure enjoys optimality guarantees detailed in [3], as it keeps the regret R_n below $O(\log n)$.

We will make use of an extension of these results to the general setting of *non-stationary* bandit problems, where the distributions $\mathcal{D}_{a,t}$ are no longer fixed with respect to t. This problem has been studied in [18], and results were obtained for a class of distributions $\mathcal{D}_{a,t}$ that respect assumptions referred to as *drift conditions*.

For a fixed $n \geq 1$, let $\overline{X}_{a,n}$ denote the random variable obtained as the average of the random variables associated with the first n plays of a. Let $\mu_{a,n} = \mathbb{E}[\overline{X}_{a,n}]$. We assume that these expected means eventually converge, and let $\mu_a = \lim_{n\to\infty} \mu_{a,n}$.

▶ **Definition 10** (Drift conditions). *For all $a \in A$, the sequence $(\mu_{a,n})_{n\geq 1}$ converges to some value μ_a. Moreover, there exists a constant $C_p > 0$ and an integer N_p such that for $n \geq N_p$ and any $\delta > 0$, if $\Delta_n(\delta) = C_p\sqrt{n\ln(1/\delta)}$ then the tail inequalities $\mathbb{P}[n\overline{X}_{a,n} \geq n\mu_{a,n} + \Delta_n(\delta)] \leq \delta$ and $\mathbb{P}[n\overline{X}_{a,n} \leq n\mu_{a,n} - \Delta_n(\delta)] \leq \delta$ hold.*

We recall in Appendix B the results of [18], and provide an informal description of those results here. Consider using the algorithm UCB1 on a non-stationary bandit problem satisfying the drift conditions, with $c_{t,t_a} = 2C_p\sqrt{\frac{\ln t}{t_a}}$. First, one can bound logarithmically the number of times a suboptimal action is played. This is used to bound the difference between μ_a and $\mathbb{E}[\overline{X}_n]$ by $O(\ln n/n)$, where a is an optimal action and where $\overline{X}_n$ denotes the global average of payoffs received over the first n steps. This is the main theoretical guarantee obtained for the optimality of UCB1. Also for any action a, the authors state a lower bound for the number of times the action is played. The authors also prove a tail inequality similar to the one described in the drift conditions, but on the random variable $\overline{X}_n$ instead of $\overline{X}_{a,n}$. This will be useful for inductive proofs later on, when the usage of UCB1 is nested so that the global sequence $\overline{X}_n$ corresponds to a sequence $\overline{X}_{b,n}$ of an action b played from the next state of the MDP. Finally, it is shown that the probability of making the wrong decision (choosing a suboptimal action) converges to 0 as the number of plays n grows large enough.

3 Monte Carlo tree search with simulation

In a receding horizon approach, the objective is to compute $\mathsf{Val}_M^H(s_0)$ and σ_M^H for some state s_0 and some horizon H. Exact procedures such as the recursive computation of Proposition 7 can not be used on large MDPs, resulting in heuristic approaches. We focus on the Monte Carlo Tree Search (MCTS) algorithm [7], that can be seen as computing approximations of Val_M^H and $\sigma_M^H(s_0)$ on the unfolding $T(M, s_0, H)$. Note that rewards in the MDP M are bounded.[3] For the sake of simplicity we assume without loss of generality that for all paths p of length at most H the total reward $\mathsf{Reward}_M(p)$ belongs to $[0, 1]$.

Given an initial state s_0, MCTS is an iterative process that incrementally constructs a search tree rooted at s_0 describing paths of M and their associated values. This process goes on until a specified budget (of number of iterations or time) is exhausted. An iteration constructs a path in M by following a decision strategy to *select* a sequence of nodes in the search tree. When a node that is not part of the current search tree is reached, the tree is expanded with this new node, whose expected reward is approximated by *simulation*. This value is then used to update the knowledge of all selected nodes in *backpropagation*.

In the search tree, each node represents a path. For a node p and an action $a \in A$, let $\mathsf{children}(p, a)$ be a list of nodes representing paths of the form $p \cdot as'$ where $s' \in \mathsf{Supp}(P(\mathsf{last}(p), a))$. For each node (resp. node-action pair) we store a value $\mathsf{value}(p)$ (resp. $\mathsf{value}(p, a)$) computed for node p (resp. for playing a from node p), meant to approximate $\mathsf{Val}_M^{H-|p|}(\mathsf{last}(p))$ (resp. $R(\mathsf{last}(p), a) + \sum_{s'} P(\mathsf{last}(p), a)(s')\mathsf{Val}_M^{H-|p|-1}(s')$), and a counter $\mathsf{count}(p)$ (resp. $\mathsf{count}(p, a)$), that keeps track of the number of iterations that selected node p (resp. that selected the action a from p). We add subscripts $i \geq 1$ to these notations to denote the number of previous iterations, so that $\mathsf{value}_i(p)$ is the value of p obtained after i iterations of MCTS, among which p was selected $\mathsf{count}_i(p)$ times. We also define $\mathsf{total}_i(p)$ and $\mathsf{total}_i(p, a)$ as shorthand for respectively $\mathsf{value}_i(p) \times \mathsf{count}_i(p)$ and $\mathsf{value}_i(p, a) \times \mathsf{count}_i(p, a)$. Each iteration consists of three phases. Let us describe these phases at iteration number i.

Selection phase. Starting from the root node, MCTS descends through the existing search tree by choosing actions based on the current values and counters and by selecting next states stochastically according to the MDP. This continues until reaching a node q, either outside of the search tree or at depth H. In the former case, the simulation phase is called to obtain a value $\mathsf{value}_i(q)$ that will be backpropagated along the path q. In the latter case, we use the exact value $\mathsf{value}_i(q) = R_T(\mathsf{last}(q))$ instead.

The action selection process needs to balance between the exploration of new paths and the exploitation of known, promising paths. A popular way to balance both is the *upper confidence bound for trees* (UCT) algorithm [18], that interprets the action selection problem of each node of the MCTS tree as a bandit problem, and selects an action a^* in the set $\arg\max_{a \in A} \left[\mathsf{value}_{i-1}(p, a) + C\sqrt{\frac{\ln(\mathsf{count}_{i-1}(p))}{\mathsf{count}_{i-1}(p,a)}}\right]$, for some constant C.

Simulation phase. In the simulation phase, the goal is to get an *initial approximation* for the value of a node p, that will be refined in future iterations of MCTS. Classically, a sampling-based approach can be used, where one computes a fixed number $c \in \mathbb{N}$ of paths $p \cdot p'$ in $\mathsf{Paths}_M^H(p)$. Then, one can compute $\mathsf{value}_i(p) = \frac{1}{c}\sum_{p'} \mathsf{Reward}_M(p')$, and fix $\mathsf{count}_i(p)$ to 1. Usually, the samples are derived by selecting actions uniformly at random in the MDP.

[3] There are finitely many paths of length at most H, with rewards in $\mathbb{R}$.

In our theoretical analysis of MCTS, we take a *more general approach* to the simulation phase, defined by a finite domain $I \subseteq [0,1]$ and a function $f : \mathsf{Paths}_M^{\leq H} \to \mathcal{D}(I)$ that maps every path p to a probability distribution on I. In this approach, the simulation phase simply draws a value $\mathsf{value}_i(p)$ at random according to the distribution $f(p)$, and sets $\mathsf{count}_i(p) = 1$.

Backpropagation phase. From the value $\mathsf{value}_i(p)$ obtained at a leaf node $p = s_0 a_0 s_1 \dots s_h$ at depth h in the search tree, let $\mathsf{reward}_i(p_{|k}) = \sum_{l=k}^{h-1} R(s_l, a_l) + \mathsf{value}_i(p)$ denote the reward associated with the path from node $p_{|k}$ to p in the search tree. For k from 0 to $h-1$ we update the values according to $\mathsf{value}_i(p_{|k}) = \frac{\mathsf{total}_{i-1}(p_{|k}) + \mathsf{reward}_i(p_{|k})}{\mathsf{count}_i(p_{|k})}$. The value $\mathsf{value}_i(p_{|k}, a_k)$ is updated based on $\mathsf{total}_{i-1}(p_{|k}, a_k)$, $\mathsf{reward}_i(p_{|k})$ and $\mathsf{count}_i(p_{|k}, a_k)$ with the same formula.

Theoretical analysis. In the remainder of this section, we prove Theorem 11, that provides theoretical properties of the MCTS algorithm with a general simulation phase (defined by some fixed I and f). This theorem was proven in [18, Theorem 6] for a version of the algorithm that called MCTS recursively until leaves were reached, as opposed to the sampling-based approach that has become standard in practice. Note that sampling-based approaches are captured by our general description of the simulation phase. Indeed, if the number of samples c is set to 1, let I be the set of rewards associated with paths of $\mathsf{Paths}_M^{\leq H}$, and let $f(p)$ be a probability distribution over I, such that for every reward $\mathsf{Reward}_M(p') \in I$, $f(p)(\mathsf{Reward}_M(p'))$ is the probability of path p' being selected with a uniform action selections in $T(M, s_0, H)$, starting from the node p. Then, the value $\mathsf{value}_i(p)$ drawn at random according to the distribution $f(p)$ corresponds to the reward of a random sample $p \cdot p'$ drawn in Paths_M^H. If the number of samples c is greater than 1, one simply needs to extend I to be the set of average rewards over c paths, while $f(p)$ becomes a distribution over average rewards.

▶ **Theorem 11.** *Consider an MDP M, a horizon H and a state s_0. Let $V_n(s_0)$ be a random variable that represents the value $\mathsf{value}_n(s_0)$ at the root of the search tree after n iterations of the MCTS algorithm on M. Then, $|\mathbb{E}[V_n(s_0)] - \mathsf{Val}_M^H(s_0)|$ is bounded by $\mathcal{O}((\ln n)/n)$. Moreover, the failure probability $\mathbb{P}[\arg\max_a \mathsf{value}_n(s_0, a) \not\subseteq \mathsf{opt}_M^H(s_0)]$ converges to zero.*

Following the proof scheme of [18, Theorem 6], this theorem is obtained from the results mentioned in Section 2.2. To this end, every node p of the search tree is considered to be an instance of a bandit problem with non-stationary distributions. Every time a node is selected, a step is processed in the corresponding bandit problem.

Let $(\mathcal{I}_i(p))_{i \geq 1}$ be a sequence of iteration numbers for the MCTS algorithm that describes when the node p is selected, so that the simulation phase was used on p at iteration number $\mathcal{I}_1(p)$, and so that the i-th selection of node p happened on the iteration number $\mathcal{I}_i(p)$. We define sequences $(\mathcal{I}_i(p, a))_{i \in \mathbb{N}}$ similarly for node-action pairs.

For all paths p and actions a, a payoff sequence $(x_{a,t})_{t \geq 1}$ of associated random variables $(X_{a,t})_{t \geq 1}$ is defined by $x_{a,t} = \mathsf{reward}_{\mathcal{I}_t(p,a)}(p)$. Note that in the selection phase at iteration number $\mathcal{I}_t(p, a)$, p must have been selected and must be a prefix of length k of the leaf node p' reached in this iteration, so that $\mathsf{reward}_{\mathcal{I}_t(p,a)}(p)$ is computed as $\mathsf{reward}_{\mathcal{I}_t(p,a)}(p'_{|k})$ in the backpropagation phase. According to the notations of Section 2.2, for all $t \geq 1$ we have $\mathsf{count}_{\mathcal{I}_t(p)}(p) = t$, $\mathsf{count}_{\mathcal{I}_t(p)}(p, a) = t_a$ and $\mathsf{value}_{\mathcal{I}_t(p)}(p, a) = \overline{x}_{a,t_a}$.

Then, one can obtain Theorem 11 by applying inductively the UCB1 results recalled in Appendix B on the search tree in a bottom-up fashion. Indeed, as the root s_0 is selected at every iteration, $\mathcal{I}_n(s_0) = n$ and $\mathsf{value}_n(s_0) = \overline{x}_n$, while $\mathsf{Val}_M^H(s_0)$ corresponds to recursively selecting optimal actions by Proposition 7.

The main difficulty, and the difference our simulation phase brings compared with the proof of [18, Theorem 6], lies in showing that our payoff sequences $(x_{a,t})_{t\geq 1}$, defined with an initial simulation step, still satisfy the drift conditions of Definition 10. We argue that this is true for all simulation phases defined by any I and f:

▶ **Lemma 12.** *For any MDP M, horizon H and state s_0, the sequences $(X_{a,t})_{t\geq 1}$ satisfy the drift conditions.*

Although the long-term guarantees of Theorem 11 hold for any simulation phase independently of the MDP, in practice one would expect better results from a good simulation, that gives a value close to the real value of the current node. Domain-specific knowledge can be used to obtain such simulations, and also to guide the selection phase based on heuristics. Our goal will be to preserve the theoretical guarantees of MCTS in the process.

4 Symbolic advice for MCTS

In this section, we introduce a notion of advice meant to guide the construction of the Monte Carlo search tree. We argue that a symbolic approach is needed in order to handle large MDPs in practice. Let a *symbolic advice* $\mathcal{A}$ be a logical formula over finite paths whose truth value can be tested with an operator $\models$.

▶ **Example 13.** A number of standard notions can fit this framework. For example, reachability and safety properties, LTL formulæ over finite traces or regular expressions could be used. We will use a safety property for Pac-Man as a example (see Figure 1), by assuming that the losing states of the MDP should be avoided. This advice is thus satisfied by every path such that Pac-Man does not make contact with a ghost.

We denote by $\mathsf{Paths}_M^H(\mathcal{A})$ the set of paths $p \in \mathsf{Paths}_M^H$ such that $p \models \mathcal{A}$. For a path $p \in \mathsf{Paths}_M^{\leq H}$, we denote by $\mathsf{Paths}_M^H(p, \mathcal{A})$ the set of paths $p' \in \mathsf{Paths}_M^H(p)$ such that $p' \models \mathcal{A}$.[4]

A *nondeterministic strategy* is a function $\sigma : \mathsf{Paths}_M \to 2^A$ that maps a finite path p to a subset of A. For a strategy σ' and a nondeterministic strategy σ, $\sigma' \subseteq \sigma$ if for all p, $\mathsf{Supp}(\sigma'(p)) \subseteq \sigma(p)$. Similarly, a nondeterministic strategy for the environment is a function $\tau : \mathsf{Paths}_M \times A \to 2^S$ that maps a finite path p and an action a to a subset of $\mathsf{Supp}(P(\mathsf{last}(p), a))$. We extend the notations used for probabilistic strategies to nondeterministic strategies in a natural way, so that $\mathsf{Paths}_M^H(\sigma)$ and $\mathsf{Paths}_M^H(\tau)$ denote the paths of length H compatible with the strategy σ or τ, respectively.

For a symbolic advice $\mathcal{A}$ and a horizon H, we define a nondeterministic strategy $\sigma_{\mathcal{A}}^H$ and a nondeterministic strategy $\tau_{\mathcal{A}}^H$ for the environment such that for all paths p with $|p| < H$,

$$\sigma_{\mathcal{A}}^H(p) = \{a \in A \mid \exists s \in S, \exists p' \in \mathsf{Paths}_M^{H-|p|-1}(s), p \cdot as \cdot p' \models \mathcal{A}\},$$

$$\tau_{\mathcal{A}}^H(p, a) = \{s \in S \mid \exists p' \in \mathsf{Paths}_M^{H-|p|-1}(s), p \cdot as \cdot p' \models \mathcal{A}\}.$$

The strategies $\sigma_{\mathcal{A}}^H$ and $\tau_{\mathcal{A}}^H$ can be defined arbitrarily on paths p of length at least H, for example with $\sigma_{\mathcal{A}}^H(p) = A$ and $\tau_{\mathcal{A}}^H(p, a) = \mathsf{Supp}(P(\mathsf{last}(p), a))$ for all actions a. Note that by definition, $\mathsf{Paths}_M^H(s, \mathcal{A}) = \mathsf{Paths}_M^H(s, \sigma_{\mathcal{A}}^H) \cap \mathsf{Paths}_M^H(s, \tau_{\mathcal{A}}^H)$ for all states s.

[4] In particular, for all $s \in S$, $\mathsf{Paths}_M^H(s, \mathcal{A})$ refers to the paths of length H that start from s and that satisfy $\mathcal{A}$.

Let $\top$ (resp. $\bot$) denote the universal advice (resp. the empty advice) satisfied by every finite path (resp. never satisfied), and let $\sigma_\top$ and $\tau_\top$ (resp. $\sigma_\bot$ and $\tau_\bot$) be the associated nondeterministic strategies. We define a class of advice that can be enforced against an adversarial environment by following a nondeterministic strategy, and that are minimal in the sense that paths that are not compatible with this strategy are not allowed.

▶ **Definition 14** (Strongly enforceable advice)**.** *A symbolic advice $\mathcal{A}$ is called a strongly enforceable advice from a state s_0 and for a horizon H if there exists a nondeterministic strategy σ such that $\mathsf{Paths}^H_M(s_0,\sigma) = \mathsf{Paths}^H_M(s_0,\mathcal{A})$, and such that $\sigma(p) \neq \emptyset$ for all paths $p \in \mathsf{Paths}^{\leq H-1}_M(s_0,\sigma)$.*

Note that Definition 14 ensures that paths that follow σ can always be extended into longer paths that follow σ. This is a reasonable assumption to make for a nondeterministic strategy meant to enforce a property. In particular, s_0 is a path of length 0 in $\mathsf{Paths}^0_M(s_0,\sigma)$, so that $\sigma(s_0) \neq \emptyset$ and so that by induction $\mathsf{Paths}^i_M(s_0,\sigma) \neq \emptyset$ for all $i \in [0,H]$.

▶ **Lemma 15.** *Let $\mathcal{A}$ be a strongly enforceable advice from s_0 with horizon H. It holds that $\mathsf{Paths}^H_M(s_0,\sigma^H_\mathcal{A}) = \mathsf{Paths}^H_M(s_0,\mathcal{A})$. Moreover, for all paths $p \in \mathsf{Paths}^{\leq H-1}_M(s_0)$ and all actions a, either $\tau^H_\mathcal{A}(p,a) = \tau_\top(p,a)$ or $\tau^H_\mathcal{A}(p,a) = \tau_\bot(p,a)$. Finally, for all paths p in $\mathsf{Paths}^{\leq H-1}_M(s_0,\sigma^H_\mathcal{A})$, $\sigma^H_\mathcal{A}(p) \neq \emptyset$ and $a \in \sigma^H_\mathcal{A}(p)$ if and only if $\tau^H_\mathcal{A}(p,a) = \tau_\top(p,a)$.*

Proof. We have $\mathsf{Paths}^H_M(s_0,\mathcal{A}) = \mathsf{Paths}^H_M(s_0,\sigma^H_\mathcal{A}) \cap \mathsf{Paths}^H_M(s_0,\tau^H_\mathcal{A})$ for any advice $\mathcal{A}$. Let us prove that $\mathsf{Paths}^H_M(s_0,\sigma^H_\mathcal{A}) \subseteq \mathsf{Paths}^H_M(s_0,\mathcal{A})$ for a strongly enforceable advice $\mathcal{A}$ of associated strategy σ. Let $p = p' \cdot as$ be a path in $\mathsf{Paths}^H_M(s_0,\sigma^H_\mathcal{A})$. By definition of $\sigma^H_\mathcal{A}$, there exists $s' \in S$ such that $p' \cdot as' \models \mathcal{A}$, so that $p' \cdot as' \in \mathsf{Paths}^H_M(s_0,\mathcal{A}) = \mathsf{Paths}^H_M(s_0,\sigma)$. Since $s \in \mathsf{Supp}(P(\mathsf{last}(p'),a))$, $p = p' \cdot as$ must also belong to $\mathsf{Paths}^H_M(s_0,\sigma) = \mathsf{Paths}^H_M(s_0,\mathcal{A})$.

Consider a path p and an action a such that $|p| < H$. We want to prove that either all stochastic transitions starting from (p,a) are allowed by $\mathcal{A}$, or none of them are. By contradiction, let us assume that there exists s_1 and s_2 in $\mathsf{Supp}(P(\mathsf{last}(p),a))$ such that for all $p'_1 \in \mathsf{Paths}^{H-|p|-1}_M(s_1)$, $p \cdot as_1 \cdot p'_1 \not\models \mathcal{A}$, and such that there exists $p'_2 \in \mathsf{Paths}^{H-|p|-1}_M(s_2)$ with $p \cdot as_2 \cdot p'_2 \models \mathcal{A}$. From $p \cdot as_2 \cdot p'_2 \models \mathcal{A}$, we obtain $p \cdot as_2 \cdot p'_2 \in \mathsf{Paths}^H_M(\sigma)$, so that $p \cdot as_2$ is a path that follows σ. Then, $p \cdot as_1$ is a path that follows σ as well. It follows that $\sigma(p \cdot as_1) \neq \emptyset$, and $p \cdot as_1$ can be extended in to a path $p \cdot as_1 p'_3 \in \mathsf{Paths}^H_M(\sigma)$. This implies the contradiction $p \cdot as_1 p'_3 \models \mathcal{A}$.

Finally, consider a path p in $\mathsf{Paths}^{\leq H-1}_M(s_0,\sigma^H_\mathcal{A})$. By the definitions of $\sigma^H_\mathcal{A}$ and $\tau^H_\mathcal{A}$, $a \in \sigma^H_\mathcal{A}(p)$ if and only if $\tau^H_\mathcal{A}(p,a) \neq \emptyset$, so that $\tau^H_\mathcal{A}(p,a) = \tau_\top(p,a)$. Then, let us write $p = p' \cdot as$. From $p \in \mathsf{Paths}^{\leq H-1}_M(s_0,\sigma^H_\mathcal{A})$ we get $a \in \sigma^H_\mathcal{A}(p')$, so that $s \in \tau^H_\mathcal{A}(p',a)$, and therefore $\sigma^H_\mathcal{A}(p) \neq \emptyset$. ◀

A strongly enforceable advice is encoding a notion of guarantee, as $\sigma^H_\mathcal{A}$ is a winning strategy for the reachability objective on $T(M,s_0,H)$ defined by the set $\mathsf{Paths}^H_M(\mathcal{A})$.

We say that the strongly enforceable advice $\mathcal{A}'$ is *extracted* from a symbolic advice $\mathcal{A}$ for a horizon H and a state s_0 if $\mathcal{A}'$ is the greatest part of $\mathcal{A}$ that can be guaranteed for the horizon H starting from s_0, *i.e.* if $\mathsf{Paths}^H_M(s_0,\mathcal{A}')$ is the greatest subset of $\mathsf{Paths}^H_M(s_0,\mathcal{A})$ such that $\sigma^H_{\mathcal{A}'}$ is a winning strategy for the reachability objective $\mathsf{Paths}^H_M(s_0,\mathcal{A})$ on $T(M,s_0,H)$. This greatest subset always exists because if $\mathcal{A}'_1$ and $\mathcal{A}'_2$ are strongly enforceable advice in $\mathcal{A}$, then $\mathcal{A}'_1 \cup \mathcal{A}'_2$ is strongly enforceable by union of the nondeterministic strategies associated with $\mathcal{A}'_1$ and $\mathcal{A}'_2$. However, this greatest subset may be empty, and as $\bot$ is not a strongly enforceable advice we say that in this case $\mathcal{A}$ cannot be enforced from s_0 with horizon H.

▶ **Example 16.** Consider a symbolic advice $\mathcal{A}$ described by the safety property for Pac-Man of Example 13. For a fixed horizon H, the associated nondeterministic strategies $\sigma^H_{\mathcal{A}}$ and $\tau^H_{\mathcal{A}}$ describe action choices and stochastic transitions compatible with this property. Notably, $\mathcal{A}$ may not be a strongly enforceable advice, as there may be situations (p, a) where some stochastic transitions lead to bad states and some do not. In the small grid of Figure 1, the path of length 1 that corresponds to Pac-Man going left and the red ghost going up is allowed by the advice $\mathcal{A}$, but not by any safe strategy for Pac-Man as there is a possibility of losing by playing left. If a strongly enforceable advice $\mathcal{A}'$ can be extracted from $\mathcal{A}$, it is a more restrictive safety property, where the set of bad states is obtained as the attractor [20, Section 2.3] for the environment towards the bad states defined in $\mathcal{A}$. In this setting, $\mathcal{A}'$ corresponds to playing according to a strategy for Pac-Man that ensures not being eaten by adversarial ghosts for the next H steps.

▶ **Definition 17** (Pruned MDP). *For an MDP $M = (S, A, P, R, R_T)$ a horizon $H \in \mathbb{N}$, a state s_0 and an advice $\mathcal{A}$, let the pruned unfolding $T(M, s_0, H, \mathcal{A})$ be defined as a sub-MDP of $T(M, s_0, H)$ that contains exactly all paths in $\mathsf{Paths}^H_M(s_0)$ satisfying $\mathcal{A}$. It can be obtained by removing all action transitions that are not compatible with $\sigma^H_{\mathcal{A}}$, and all stochastic transitions that are not compatible with $\tau^H_{\mathcal{A}}$. The distributions $P(p, a)$ are then normalised over the stochastic transitions that are left.*

Note that by Lemma 15, if $\mathcal{A}$ is a strongly enforceable advice then $\tau^H_{\mathcal{A}}(p, a) = \tau_\top(p, a)$ for all paths p in $\mathsf{Paths}^{\leq H-1}_M(s_0, \sigma^H_{\mathcal{A}})$, so that the normalisation step for the distributions $P(p, a)$ is not needed. It follows that for all nodes p in $T(M, s_0, H, \mathcal{A})$ and all actions a, the distributions $P(p, a)$ in $T(M, s_0, H, \mathcal{A})$ are the same as in $T(M, s_0, H)$. Thus, for all strategies σ in $T(M, s_0, H, \mathcal{A})$, $\mathsf{Val}^H_{T(M,s_0,H,\mathcal{A})}(s_0, \sigma) = \mathsf{Val}^H_{T(M,s_0,H)}(s_0, \sigma)$, so that $\mathsf{Val}^H_{T(M,s_0,H,\mathcal{A})}(s_0) \leq \mathsf{Val}^H_{T(M,s_0,H)}(s_0) = \mathsf{Val}^H_M(s_0)$ by Lemma 9.

▶ **Definition 18** (Optimality assumption). *An advice $\mathcal{A}$ satisfies the optimality assumption for horizon H if $\sigma^{H,*}_{M,s} \subseteq \sigma^H_{\mathcal{A}}$ for all $s \in S$, where $\sigma^{H,*}_{M,s}$ is an optimal strategy for the expected total reward of horizon H at state s.*

▶ **Lemma 19.** *Let $\mathcal{A}$ be a strongly enforceable advice that satisfies the optimality assumption. Then, $\mathsf{Val}^H_M(s_0)$ equals $\mathsf{Val}^H_{T(M,s_0,H,\mathcal{A})}(s_0)$. Moreover, $\mathsf{opt}^H_{T(M,s_0,H,\mathcal{A})}(s_0) \subseteq \mathsf{opt}^H_M(s_0)$.*

Proof. By the optimality assumption $\sigma^{H,*}_{M,s}$ is a strategy that can be followed in $T(M, s_0, H, \mathcal{A})$. Indeed, from a path p in $\mathsf{Paths}^{\leq H-1}_M(s_0, \sigma^H_{\mathcal{A}})$ any action a in the support of $\sigma^{H,*}_{M,s}(\mathsf{last}(p))$ satisfies $a \in \sigma^H_{\mathcal{A}}(p)$. Thus, by Lemma 9 $\mathsf{Val}^H_M(s_0) = \mathsf{Val}^H_{T(M,s_0,H)}(s_0, \sigma^{H,*}_{M,s}) = \mathsf{Val}^H_{T(M,s_0,H,\mathcal{A})}(s_0, \sigma^{H,*}_{M,s})$. By definition of the optimal expected total reward, $\mathsf{Val}^H_{T(M,s_0,H,\mathcal{A})}(s_0, \sigma^{H,*}_{M,s}) \leq \mathsf{Val}^H_{T(M,s_0,H,\mathcal{A})}(s_0)$, so that $\mathsf{Val}^H_M(s_0) = \mathsf{Val}^H_{T(M,s_0,H,\mathcal{A})}(s_0)$. Let a be an action in $\mathsf{opt}^H_{T(M,s_0,H,\mathcal{A})}(s_0)$. There exists an optimal strategy σ that maximises $\mathsf{Val}^H_{T(M,s_0,H,\mathcal{A})}(s_0, \sigma)$ so that $\sigma(s_0) = a$. It follows from $\mathsf{Val}^H_{T(M,s_0,H)}(s_0) = \mathsf{Val}^H_{T(M,s_0,H,\mathcal{A})}(s_0)$ that σ is also an optimal strategy in $T(M, s_0, H)$, so that $a \in \mathsf{opt}^H_{T(M,s_0,H)}(s_0) = \mathsf{opt}^H_M(s_0)$ by Lemma 9. ◀

▶ **Example 20.** Let $\mathcal{A}'$ be a strongly enforceable safety advice for Pac-Man as described in Example 16. Assume that visiting a bad state leads to an irrecoverably bad reward, so that taking an unsafe action (*i.e.* an action such that there is a non-zero probability of losing associated with all Pac-Man strategies) is always worse (on expectation) than taking a safe action. Then, the optimality assumption holds for the advice $\mathcal{A}'$. This can be achieved by giving a penalty score for losing that is low enough.

4.1 MCTS under symbolic advice

We will augment the MCTS algorithm using two advice: a selection advice φ to guide the MCTS tree construction, and a simulation advice ψ to prune the sampling domain. We assume that the selection advice is a strongly enforceable advice that satisfies the optimality assumption. Notably, we make no such assumption for the simulation advice, so that any symbolic advice can be used.

Selection phase under advice. We use the advice φ to prune the tree according to σ^H_φ. Therefore, from any node p our version of UCT selects an action a^* in the set

$$\arg\max_{a \in \sigma^H_\varphi(p)} \left[\mathsf{value}(p,a) + C\sqrt{\frac{\ln(\mathsf{count}(p))}{\mathsf{count}(p,a)}} \right].$$

Simulation phase under advice. For the simulation phase, we use a sampling-based approach biased by the simulation advice: paths are sampled by picking actions uniformly at random in the pruned MDP $T(M, s_0, H, \psi)$, with a fixed prefix p defined by the current node in the search tree. This can be interpreted as a probability distribution over $\mathsf{Paths}^H_M(p, \psi)$. If $p \notin T(M, s_0, H, \psi)$, the simulation phase outputs a value of 0 as it is not possible to satisfy ψ from p. Another approach that does not require computing the pruned MDP repeats the following steps for a bounded number of time before returning 0 if no valid sample is found:

1. Pick a path $p \cdot p' \in \mathsf{Paths}^H_M(p)$ using a uniform sampling method;
2. If $p \cdot p' \not\models \psi$, reject and try again, otherwise output p' as a sample.

We compute $\mathsf{value}_i(p)$ by averaging the rewards of these samples.

Theoretical analysis. We show that the theoretical guarantees of the MCTS algorithm developed in Section 3 are maintained by the MCTS algorithm under symbolic advice.

▶ **Theorem 21.** *Consider an MDP M, a horizon H and a state s_0. Let $V_n(s_0)$ be a random variable that represents the value $\mathsf{value}_n(s_0)$ at the root of the search tree after n iterations of the MCTS algorithm under a strongly enforceable advice φ satisfying the optimality assumption and a simulation advice ψ. Then, $|\mathbb{E}[V_n(s_0)] - \mathsf{Val}^H_M(s_0)| = \mathcal{O}((\ln n)/n)$. Moreover, the failure probability $\mathbb{P}[\arg\max_a \mathsf{value}_n(s_0, a) \not\subseteq \mathsf{opt}^H_M(s_0)]$ converges to zero.*

In order to prove Theorem 21, we argue that running MCTS under a selection advice φ and a simulation advice ψ is equivalent to running the MCTS algorithm of Section 3 on the pruned MDP $T(M, s_0, H, \varphi)$, with a simulation phase defined using the advice ψ.

The simulation phase biased by ψ can be described in the formalism of Section 3, with a domain $I = \{\frac{1}{c}\sum_{i=1}^{c} \mathsf{Reward}_M(p_i) \mid p_1, \ldots, p_c \in \mathsf{Paths}^{\leq H}_{T(M,s_0,H,\varphi)}\}$, and a mapping f_ψ from paths p in $\mathsf{Paths}^{\leq H}_{T(M,s_0,H,\varphi)}$ to a probability distribution on I describing the outcome of a sampling phase launched from the node p. Formally, the weight of $\frac{1}{c}\sum_{i=1}^{c} \mathsf{Reward}_M(p_i) \in I$ in $f(p)$ is the probability of sampling the sequence of paths $p_1, \ldots, p_c$ in the simulation phase under advice launched from p. Then, from Theorem 11 we obtain convergence properties of MCTS under symbolic advice towards the value and optimal strategy in the pruned MDP, and Lemma 19 lets us conclude the proof of Theorem 21 as those values and strategies are maintained in M by the optimality assumption. In particular, the failure probability $\mathbb{P}[\arg\max_a \mathsf{value}_n(s_0, a) \not\subseteq \mathsf{opt}^H_M(s_0)]$ is upper bounded by $\mathbb{P}[\arg\max_a \mathsf{value}_n(s_0, a) \not\subseteq \mathsf{opt}^H_{T(M,s_0,H,\varphi)}(s_0)]$ since $\mathsf{opt}^H_{T(M,s_0,H,\varphi)} \subseteq \mathsf{opt}^H_M(s_0)$.

4.2 Using satisfiability solvers

We will now discuss the use of general-purpose solvers to implement symbolic advice according to the needs of MCTS.

A symbolic advice $\mathcal{A}$ describes a finite set of paths in Paths_M^H, and as such can be encoded as a Boolean formula over a set of variables V, such that satisfying assignments $v : V \to \{\mathsf{true}, \mathsf{false}\}$ are in bijection with paths in $\mathsf{Paths}_M^H(\mathcal{A})$.

If a symbolic advice is described in Linear Temporal Logic, and a symbolic model of the MDP M is available, one can encode $\mathcal{A}$ as a Boolean formula of size linear in the size of the LTL formula and H [5].

▶ **Example 22.** In practice, one can use Boolean variables to encode the positions of Pac-Man and ghosts in the next H steps of the game, then construct a CNF formula with clauses that encode the game rules and clauses that enforce the advice. The former clauses are implications such as "if Pac-Man is in position (x, y) and plays the action a, then it must be in position (x', y') at the next game step", while the latter clauses state that the position of Pac-Man should never be equal to the position of one of the Ghosts.

On-the-fly computation of a strongly enforceable advice. A direct encoding of a strongly enforceable advice may prove impractically large. We argue for an on-the-fly computation of $\sigma_{\mathcal{A}}^H$ instead, in the particular case where the strongly enforceable advice is extracted from a symbolic advice $\mathcal{A}$ with respect to the initial state s_0 and with horizon H.

▶ **Lemma 23.** *Let $\mathcal{A}'$ be a strongly enforceable advice extracted from $\mathcal{A}$ for horizon H. Consider a node p at depth i in $T(M, s_0, H, \mathcal{A}')$, for all $a_0 \in A$, $a_0 \in \sigma_{\mathcal{A}'}^H(p)$ if and only if*

$$\forall s_1 \exists a_1 \forall s_2 \dots \forall s_{H-i+1},\ p \cdot a_0 s_1 a_1 s_2 \dots s_{H-i+1} \models \mathcal{A}\,,$$

where actions are quantified over A and every s_k is quantified over $\mathsf{Supp}(P(s_{k-1}, a_{k-1}))$.

Proof. The proof is a reverse induction on the depth i of p. For the initialisation step with $i = H$, let us prove that $\forall s_1,\ p \cdot a_0 s_1 \models \mathcal{A}$ if and only if $a_0 \in \sigma_{\mathcal{A}'}^H(p)$. On the one hand, if $\mathcal{A}$ is guaranteed by playing a_0 from p, then a_0 must be allowed by the greatest strongly enforceable subset of $\mathcal{A}$. On the other hand, $a_0 \in \sigma_{\mathcal{A}'}^H(p)$ implies $\forall s_1,\ p \cdot a_0 s_1 \models \mathcal{A}'$ as $\mathcal{A}'$ is strongly enforceable, and finally $\mathcal{A}' \Rightarrow \mathcal{A}$. We now assume the property holds for $1 \leq i \leq H$, and prove it for $i - 1$. If $a_0 \in \sigma_{\mathcal{A}'}^H(p)$, then for all s_1 we have $s_1 \in \tau_{\mathcal{A}'}^H(p, a_0)$, so that there exists a_1 with $a_1 \in \sigma_{\mathcal{A}'}^H(p \cdot a_0 s_1)$. As $p \cdot a_0 s_1$ is at depth i we can conclude that $\forall s_1 \exists a_1 \forall s_2 \dots \forall s_{H-i+1},\ p \cdot a_0 s_1 a_1 s_2 \dots s_{H-i+1} \models \mathcal{A}$ by assumption. For the converse direction, the alternation of quantifiers states that $\mathcal{A}$ can be guaranteed from p by some deterministic strategy that starts by playing a_0, and therefore a_0 must be allowed by the strongly enforceable advice extracted from $\mathcal{A}$. ◀

Therefore, given a Boolean formula encoding $\mathcal{A}$, one can use a Quantified Boolean Formula (QBF) solver to compute $\sigma_{\mathcal{A}'}^H$, the strongly enforceable advice extracted from $\mathcal{A}$: this computation can be used whenever MCTS performs an action selection step under the advice $\mathcal{A}'$, as described in Section 4.1.

The performance of this approach will crucially depend on the number of alternating quantifiers, and in practice one may limit themselves to a smaller depth $h < H - i$ in this step, so that safety is only guaranteed for the next h steps.

Some properties can be inductively guaranteed, so that satisfying the QBF formula of Lemma 23 with a depth $H - i = 1$ is enough to guarantee the property globally. For example, if there always exists an action leading to states that are not bad, it is enough to check for safety locally with a depth of 1. This is the case in Pac-Man for a deadlock-free layout when there is only one ghost.

Weighted sampling under a symbolic advice. Given a symbolic advice $\mathcal{A}$ as a Boolean formula, and a probability distribution $w \in \mathcal{D}(\mathsf{Paths}_M^H)$, our goal is to sample paths of M that satisfy $\mathcal{A}$ with respect to w.[5] Let ω denote a weight function over Boolean assignments that matches w. This reduces our problem to the weighted sampling of satisfying assignments in a Boolean formula. An exact solver for this problem may not be efficient, but one can use the techniques of [8] to perform approximate sampling in polynomial time:

▶ **Proposition 24** ([8]). *Given a CNF formula $\mathcal{A}$, a tolerance $\epsilon > 0$ and a weight function ω, we can construct a probabilistic algorithm which outputs a satisfying assignment z such that for all y that satisfies $\mathcal{A}$:*

$$\frac{\omega(y)}{(1+\epsilon)\sum_{x\models\psi}\omega(x)} \leq Pr[z = y] \leq \frac{(1+\epsilon)\omega(y)}{\sum_{x\models\psi}\omega(x)} \ .$$

The above algorithm occasionally "fails" (outputs no assignment even though there are satisfying assignments) but its failure probability can be bounded by any given δ. Given an oracle for SAT, the above algorithm runs in time polynomial in $\ln\left(\frac{1}{\delta}\right)$, $|\psi|$, $\frac{1}{\epsilon}$ *and r where r is the ratio between highest and lowest weight according to ω.*

In particular, this algorithm uses ω as a black-box, and thus does not require precomputing the probabilities of all paths satisfying $\mathcal{A}$. In our particular application of Proposition 24, the value r can be bounded by $\left(\frac{p_{\max}|A|}{p_{\min}}\right)^H$ where $p_{\min}$ and $p_{\max}$ are the smallest and greatest probabilities for stochastic transitions in M.

Note that if we wish to sample from a given node p of the search tree, we can force p as a mandatory prefix of satisfying assignments by fixing the truth value of relevant variables in the Boolean formula.

5 A Pac-Man case study

We performed our experiments on the multi-agent game Pac-Man, using the code of [13]. The ghosts can have different strategies where they take actions based on their own position as well as position of Pac-Man. In our experiments, we used two different types of ghosts, the *random ghosts* (in green) always choose an action uniformly at random from the legal actions available, while the *directional ghosts* (in red) take the legal action that minimises the Manhattan distance to Pac-Man with probability 0.9, and move randomly otherwise.

The game can be seen as a Markov decision process, where states encode a position for each agent[6] and for the food pills in the grid, where actions encode individual Pac-Man moves, and where stochastic transitions encode the moves of ghosts according to their probabilistic models. For each state and action pair, we define a reward based on the score gained or lost by this move, as explained in the caption of Figure 1. We also assign a terminal reward to each state, so as to allow MCTS to compare paths of length H which would otherwise obtain

[5] The probability of a path p being sampled should be equal to $w(p)/\sum_{p'|p'\models\mathcal{A}} w(p')$.

[6] The last action played by ghosts should be stored as well, as they are not able to reverse their direction.

the same score. Intuitively, better terminal rewards are given to states where Pac-Man is closer to the food pills and further away from the ghosts, so that terminal rewards play the role of a static evaluation of positions.

Experiments. We used a receding horizon $H = 10$. The baseline is given by a standard implementation of the algorithm described in Section 3. A search tree is constructed with a maximum depth H, for 100 iterations, so that the search tree constructed by the MCTS algorithm contains up to 100 nodes. At the first selection of every node, 100 samples are obtained by using a uniform policy. Overall, this represents a tiny portion of the tree unfolding of depth 10, which underlines the importance of properly guiding the search to the most interesting neighborhoods. As a point of reference, we also had human players take control of Pac-Man, and computed the same statistics. The players had the ability to slow down the game as they saw fit, as we aimed for a comparison between the quality of the strategical decisions made by these approaches, and not of their reaction speeds.

We compare these baselines with the algorithm of Section 4.1, using the following advice. The *simulation advice* ψ that we consider is defined as a safety property satisfied by every path such that Pac-Man does not make contact with a ghost, as in Example 13. We provide a Boolean formula encoding ψ, so that one can use a SAT solver to obtain samples, or sampling tools as described in Proposition 24, such as WeightGen [8]. We use UniGen [9] to sample almost uniformly over the satisfying assignments of ψ.[7]

From this simulation advice, we extract whenever possible a strongly enforceable *selection advice* φ that guarantees that Pac-Man will not make contact with a ghost, as described in Example 16. If safety cannot be enforced, $\top$ is used as a selection advice, so that no pruning is performed. This is implemented by using the Boolean formula ψ in a QBF solver according to Lemma 23. For performance reasons, we guarantee safety for a smaller horizon $h < 10$, that we fixed at 3 in our experiments.

Several techniques were used to reduce the state-space of the MDP in order to obtain smaller formulæ. For example, a ghost that is too far away with respect to H or h can be safely ignored, and the current positions of the food pills is not relevant for safety.

Results. For each experiment, we ran 100 games in a high-end cluster using AMD Opteron Processors 6272 at 2.1 GHz. A summary of our results is displayed in Table 1. We mainly use the number of games won out of 100 to evaluate the performance of our algorithms.[8] In the small grid with four random ghosts, the baseline MCTS algorithm wins 17% of games. Adding the selection advice results in a slight increase of the win rate to 25%. The average score is improved as expected, but even if one ignores the ± 500 score associated with a win or a loss, we observe that more food pills were eaten on average as well. The simulation advice provides in turn a sizeable increase in both win rate (achieving 71%) and average score. Using both advice at the same time gave the best results overall, with a win rate of 85%. The same observations can be made in other settings as well, either with a directional ghost model or on a large grid. Moreover, the simulation advice significantly reduces the number of game turns Pac-Man needs to win, resulting in fewer game draws, most notably on the large grid.

[7] The distribution over path is slightly different than when sampling uniformly over actions in the pruned MDP $T(M, s_0, H, \psi)$, but UniGen enjoys better performances than WeightGen.

[8] We do not evaluate the accuracy in terms of making optimal choices because those cannot be computed due to the size of the MDPs (about 10^{16} states).

Table 1 Summary of experiments with different ghost models, algorithms and grid size. The win, loss and draw columns denote win/loss/draw rates in percents (the game ends in a draw after 300 game steps). The food eaten column refers to the number of food pills eaten on average, out of 25 food pills in total. Score refers to the average score obtained over all runs.

Grid	Ghosts	Algorithm	win	loss	draw	food	score
9 x 21	4 x Random	MCTS	17	59	24	16.65	-215.32
		MCTS+Selection advice	25	54	21	17.84	-146.44
		MCTS+Simulation advice	71	29	0	22.11	291.80
		MCTS+both advice	85	15	0	23.42	468.74
		Human	44	56	0	18.87	57.76
	1 x Directional + 3 x Random	MCTS	11	85	4	14.86	-339.99
		MCTS+Selection advice	16	82	2	15.25	-290.6
		MCTS+Simulation advice	27	70	3	17.14	-146.79
		MCTS+both advice	33	66	1	17.84	-92.47
		Human	24	76	0	15.10	-166.28
27 x 28	4 x Random	MCTS	1	10	89	14.85	-182.77
		MCTS+both advice	95	5	0	24.10	517.04

In the baseline experiments without any advice, computing the next action played by Pac-Man from a given state takes 200 seconds of runtime on average.[9] Comparatively, the algorithm with both advice is about three times slower than the baseline. If we make use of the same time budget in the standard MCTS algorithm (roughly increasing the number of nodes in the MCTS tree threefold), the win rate climbs to 26%, which is still significantly below the 85% win rate achieved with advice. Moreover, while this experiment was not designed to optimise the performance of these approaches in terms of computing time, we were able to achieve a performance of 5s per move on a standard laptop by reducing the number of iterations and samples in MCTS. This came at the cost of a decreased win-rate of 76% with both advice. Further code improvements *e.g.* using parallelism as in [10] could reasonably lead to real-time performances.

Supplementary material is available at `http://di.ulb.ac.be/verif/debraj/pacman/`.

6 Conclusion and future works

In this paper, we have introduced the notion of symbolic advice to guide the selection and the simulation phases of the MCTS algorithm. We have identified sufficient conditions to preserve the convergence guarantees offered by the MCTS algorithm while using symbolic advice. We have also explained how to implement them using SAT and QBF solvers in order to apply symbolic advice to large MDP defined symbolically rather than explicitly. We believe that the generality, flexibility and precision offered by logical formalism to express symbolic advice in MCTS can be used as the basis of a methodology to systematically inject domain knowledge into MCTS. We have shown that domain knowledge expressed as simple symbolic advice (safety properties) improves greatly the efficiency of the MCTS algorithm in the Pac-Man application. This application is challenging as the underlying MDPs have huge state spaces, *i.e.* up to 10^{23} states. In this application, symbolic advice allow the MCTS algorithm to reach or even surpass human level in playing.

[9] This holds for both the small and large grids, as in both cases we consider the next 10 game steps only, resulting in MCTS trees of similar size.

As further work, we plan to offer a compiler from LTL to symbolic advice, in order to automate their integration in the MCTS algorithm for diverse application domains. We also plan to work on the efficiency of the implementation. So far, we have developed a prototype implementation written in Python (an interpreted language). This implementation cannot be used to evaluate performances in absolute terms but it was useful to show that if the same amount of resources is allocated to the two algorithms the one with advice performs much better. We believe that by using a well-optimised code base and by exploiting parallelism, we should be able to apply our algorithm in real-time and preserve the level of quality reported in the experimental section. Finally, we plan to study how learning can be incorporated in our framework. One natural option is to replace the static reward function used after H steps by a function learned from previous runs of the algorithm and implemented using a neural network (as it is done in AlphaGo [25] for example).

We thank Gilles Geeraerts for fruitful discussions in the early phases of this work.

References

1 Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence, (AAAI 2018)*, pages 2669–2678. AAAI Press, 2018.

2 Pranav Ashok, Tomás Brázdil, Jan Kretínský, and Ondrej Slámecka. Monte Carlo tree search for verifying reachability in Markov decision processes. In *Proceedings of the 8th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation (ISoLA 2018)*, volume 11245 of *Lecture Notes in Computer Science*, pages 322–335. Springer, 2018. doi:10.1007/978-3-030-03421-4_21.

3 Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2-3):235–256, 2002. doi:10.1023/A:1013689704352.

4 Hendrik Baier and Mark H. M. Winands. MCTS-minimax hybrids. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(2):167–179, 2015. doi:10.1109/TCIAIG.2014.2366555.

5 Armin Biere, Keijo Heljanko, Tommi Junttila, Timo Latvala, and Viktor Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, Volume 2, Issue 5, November 2006. doi:10.2168/LMCS-2(5:5)2006.

6 Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Kretínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of Markov decision processes using learning algorithms. In *Proceedings of the 12th International Symposium on Automated Technology for Verification and Analysis (ATVA 2014)*, volume 8837 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2014. doi:10.1007/978-3-319-11936-6_8.

7 Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012. doi:10.1109/TCIAIG.2012.2186810.

8 Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. Distribution-aware sampling and weighted model counting for SAT. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence, 2014 (AAAI 2014)*, pages 1722–1730. AAAI Press, 2014. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8364.

9 Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. On parallel scalable uniform SAT witness generation. In *Proceedings of the 21st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2015), Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS 2015)*, volume 9035 of *Lecture Notes in Computer Science*, pages 304–319. Springer, 2015. doi:10.1007/978-3-662-46681-0_25.

10 Guillaume M. J. B. Chaslot, Mark H. M. Winands, and H. Jaap van den Herik. Parallel Monte-Carlo tree search. In H. Jaap van den Herik, Xinhe Xu, Zongmin Ma, and Mark H. M. Winands, editors, *Proceedings of the 6th International Conference on Computers and Games (CG 2008)*, volume 5131 of *Lecture Notes in Computer Science*, pages 60–71. Springer, 2008. doi:10.1007/978-3-540-87608-3_6.

11 Krishnendu Chatterjee, Petr Novotný, Guillermo A. Pérez, Jean-François Raskin, and Dorde Zikelic. Optimizing expectation with guarantees in POMDPs. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI 2017)*, pages 3725–3732. AAAI Press, 2017.

12 Przemyslaw Daca, Thomas A. Henzinger, Jan Kretínský, and Tatjana Petrov. Faster statistical model checking for unbounded temporal properties. In *Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2016), Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS 2016)*, volume 9636 of *Lecture Notes in Computer Science*, pages 112–129. Springer, 2016. doi:10.1007/978-3-662-49674-9_7.

13 John DeNero and Dan Klein. CS 188 : Introduction to artificial intelligence. URL: https://inst.eecs.berkeley.edu/~cs188.

14 Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Cautious reinforcement learning with logical constraints. In Amal El Fallah Seghrouchni, Gita Sukthankar, Bo An, and Neil Yorke-Smith, editors, *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, (AAMAS 2020)*, pages 483–491. International Foundation for Autonomous Agents and Multiagent Systems, 2020. URL: https://dl.acm.org/doi/abs/10.5555/3398761.3398821.

15 Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963. doi:10.1080/01621459.1963.10500830.

16 Jing Huang, Zhiqing Liu, Benjie Lu, and Feng Xiao. Pruning in UCT algorithm. In *Proceedings of the International Conference on Technologies and Applications of Artificial Intelligence (TAAI 2010)*, pages 177–181, 2010. doi:10.1109/TAAI.2010.38.

17 Michael J. Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning*, 49(2-3):193–208, 2002. doi:10.1023/A:1017932429737.

18 Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML 2006)*, volume 4212 of *Lecture Notes in Computer Science*, pages 282–293. Springer, 2006. doi:10.1007/11871842_29.

19 Jan Kretínský, Guillermo A. Pérez, and Jean-François Raskin. Learning-based mean-payoff optimization in an unknown MDP under omega-regular constraints. In *Proceedings of the 29th International Conference on Concurrency Theory (CONCUR 2018)*, volume 118 of *Leibniz International Proceedings in Informatics*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

20 Christof Löding. Infinite games and automata theory. In Krzysztof R. Apt and Erich Grädel, editors, *Lectures in Game Theory for Computer Scientists*, pages 38–73. Cambridge University Press, 2011.

21 João Marques-Silva and Sharad Malik. Propositional SAT solving. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 247–275. Springer, 2018.

22 Nina Narodytska, Alexander Legg, Fahiem Bacchus, Leonid Ryzhyk, and Adam Walker. Solving games without controllable predecessor. In *Proceedings of the 26th International Conference on Computer Aided Verification (CAV 2014)*, volume 8559 of *Lecture Notes in Computer Science*, pages 533–540. Springer, 2014. doi:10.1007/978-3-319-08867-9_35.

23 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994. doi:10.1002/9780470316887.

24 Ankit Shukla, Armin Biere, Luca Pulina, and Martina Seidl. A survey on applications of quantified Boolean formulas. In *Proceedings of the 31st IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2019)*, pages 78–84. IEEE, 2019. doi:10.1109/ICTAI.2019.00020.

25 David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi:10.1038/nature16961.

26 David Silver and Gerald Tesauro. Monte-Carlo simulation balancing. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 945–952, 2009.

A Markov decision processes

One can make an MDP strongly aperiodic without changing the optimal expected average reward and its optimal strategies with the following transition:

▶ **Definition 25** (Aperiodic transformation [23, Section 8.5.4]). *For an MDP $M = (S, A, P, R)$, we define a new MDP $M_\alpha = (S, A, P_\alpha, R_\alpha)$ for $0 < \alpha < 1$, with $R_\alpha(s,a) = R(s,a)$, $P_\alpha(s,a)(s) = \alpha + (1-\alpha)P(s,a)(s)$ and $P_\alpha(s,a)(s') = (1-\alpha)P(s,a)(s')$. Notice that M_α is strongly aperiodic.*

Every finite path in M is also in M_α. Thus for a strategy $\widehat{\sigma}$ in M_α, there is a σ in M whose domain is restricted to the paths in M.

▶ **Proposition 26** ([23, Section 8.5.4]). *Let M be an MDP. M_α is a new MDP generated by applying the aperiodic transformation mentioned above. Then the set of memoryless strategies that optimises the expected average reward in M_α is the same as the set of memoryless strategies the optimises the expected average reward in M. Also from any s, $\mathsf{Val}_M(s) = \mathsf{Val}_{M_\alpha}(s)$.*

▶ **Definition 27** (Finite horizon unfolding of an MDP). *For an MDP $M = (S, A, P, R, R_T)$, a horizon depth $H \in \mathbb{N}$ and a state s_0, the unfolding of M from s_0 and with horizon H is a tree-shaped MDP defined as $T(M, s_0, H) = (S' = S_0 \cup \cdots \cup S_H, A, P', R', R'_T)$, where for all $i \in [0, H]$, $S_i = \mathsf{Paths}^i(s_0)$. The mappings P', R' and R'_T are inherited from P, R and R'_T in a natural way with additional self-loops at the leaves of the unfolding, so that for all $i \in [0, H]$, $p \in S_i$, $a \in A$ and $p' \in S'$,*

$$P'(p,a)(p') = \begin{cases} P(\mathsf{last}(p), a)(\mathsf{last}(p')) & \text{if } i < H \text{ and } \exists s' \in S, p' = p \cdot as' \\ 1 & \text{if } i = H \text{ and } p' = p \\ 0 & \text{otherwise,} \end{cases}$$

$$R'(p,a) = \begin{cases} R(\mathsf{last}(p), a) & \text{if } i < H \\ 0 & \text{otherwise.} \end{cases}$$

$$R'_T(p) = R_T(\mathsf{last}(p))$$

Proof of Lemma 9. Let us prove that for all $i \in [0, H]$ and all $p \in S_i$,

- $\mathsf{Val}_M^{H-i}(\mathsf{last}(p)) = \mathsf{Val}_{T(M,s_0,H)}^{H-i}(p)$, and
- $\mathsf{opt}_M^{H-i}(\mathsf{last}(p)) = \mathsf{opt}_{T(M,s_0,H)}^{H-i}(p)$.

We prove the first statement by induction on $H-i$. For $H-i=0$, for all $p \in S_i$, $\mathsf{Val}_M^{H-i}(\mathsf{last}(p)) = \mathsf{Val}_{T(M,s_0,H)}^{H-i}(p) = R_T(\mathsf{last}(p))$. Assume the statement is true for $H-i=k$, so that for all $p \in S_{H-k}$, $\mathsf{Val}_M^k(\mathsf{last}(p)) = \mathsf{Val}_{T(M,s_0,H)}^k(p)$. Then for all $p \in S_{H-k-1}$, we have for all $a \in A$ and $s \in Supp(P(\mathsf{last}(p),a))$, $\mathsf{Val}_M^k(s) = \mathsf{Val}_{T(M,s_0,H)}^k(p \cdot as)$. It follows that

$$\begin{aligned}\mathsf{Val}_M^{k+1}(\mathsf{last}(p)) &= \max_{a\in A}(R(\mathsf{last}(p),a) + \sum_s P(\mathsf{last}(p),a)\mathsf{Val}_M^k(s)) \\ &= \max_{a\in A}(R(\mathsf{last}(p),a) + \sum_s P(\mathsf{last}(p),a)\mathsf{Val}_{T(M,s_0,H)}^k(p\cdot as)) \\ &= \mathsf{Val}_{T(M,s_0,H)}^{k+1}(p)\,.\end{aligned}$$

From $\mathsf{Val}_M^{H-i}(\mathsf{last}(p)) = \mathsf{Val}_{T(M,s_0,H)}^{H-i}(p)$ and $\mathsf{opt}_M^H(\mathsf{last}(p)) = \arg\max_{a\in A}(R(\mathsf{last}(p),a) + \sum_s P(\mathsf{last}(p),a)\mathsf{Val}_M^{H-1}(s))$ we derive $\mathsf{opt}_M^{H-i}(\mathsf{last}(p)) = \mathsf{opt}_{T(M,s_0,H)}^{H-i}(p)$. ◀

B UCB

Let $\overline{X}_{a,n} = \frac{1}{n}\sum_{t=1}^n X_{a,t}$ denote the average of the first n plays of action a. Let $\mu_{a,n} = \mathbb{E}[\overline{X}_{a,n}]$. We assume that these expected means eventually converge, and let $\mu_a = \lim_{n\to\infty}\mu_{a,n}$.

▶ **Definition 28** (Drift conditions).

- *For all $a \in A$, the sequence $(\mu_{a,n})_{n\geq 1}$ converges to some value μ_a.*
- *There exists a constant $C_p > 0$ and an integer N_p such that for $n \geq N_p$ and any $\delta > 0$, $\Delta_n(\delta) = C_p\sqrt{n\ln(1/\delta)}$, the following bounds hold:*

$$\mathbb{P}\Big[n\overline{X}_{a,n} \geq n\mu_{a,n} + \Delta_n(\delta)\Big] \leq \delta\,,$$

$$\mathbb{P}\Big[n\overline{X}_{a,n} \leq n\mu_{a,n} - \Delta_n(\delta)\Big] \leq \delta\,.$$

We define $\delta_{a,n} = \mu_{a,n} - \mu_a$. Then, μ^*, μ_n^*, δ_n^* are defined as μ_j, $\mu_{j,n}$, $\delta_{j,n}$ where j is the optimal action.[10] Moreover, let $\Delta_a = \mu^* - \mu_a$.

As $\delta_{a,n}$ converges to 0 by assumption, for all $\epsilon > 0$ there exists $N_0(\epsilon) \in \mathbb{N}$, such that for $t > N_0(\epsilon)$, then $2|\delta_{a,t}| \leq \epsilon\Delta_a$ and $2|\delta_t^*| \leq \epsilon\Delta_a$ for all all suboptimal actions $a \in A$.

The authors start by bounding the number of time a suboptimal action is played:

▶ **Theorem 29** ([18, Theorem 1]). *Consider UCB1 applied to a non-stationary bandit problem with $c_{t,s} = 2C_p\sqrt{\frac{\ln t}{s}}$. Fix $\epsilon > 0$. Let $T_a(n)$ denote number of times action a has been played at time n. Then under the drift conditions, there exists N_p such that for all suboptimal actions $a \in A$,*

$$\mathbb{E}[T_a(n)] \leq \frac{16C_p^2\ln n}{(1-\epsilon)^2\Delta_a^2} + N_0(\epsilon) + N_p + 1 + \frac{\pi^2}{3}\,.$$

Let $\overline{X}_n = \sum_{a\in A}\frac{T_a(n)}{n}\bar{X}_{a,T_a(n)}$ denote the global average of payoffs received up to time n Then, one can bound the difference between μ^* and $\overline{X}_n$:

▶ **Theorem 30** ([18, Theorem 2]). *Under the drift conditions of Definition 28, it holds that*

$$|\mathbb{E}[\overline{X}_n] - \mu^*| \leq |\delta_n^*| + O\left(\frac{|A|(C_p^2\ln n + N_0(1/2))}{n}\right)\,.$$

[10] It is assumed for simplicity that a single action a is optimal, *i.e.* maximises $\mathbb{E}[X_{a,n}]$ for n large enough.

The following theorem shows that the number of times an action is played can be lower bounded:

▶ **Theorem 31** ([18, Theorem 3]). *Under the drift conditions of Definition 28, there exists some positive constant ρ such that after n iterations for all action a, $T_a(n) \geq \lceil \rho \ln(n) \rceil$.*

Then, the authors also prove a tail inequality similar to the one described in the drift conditions, but on the random variable $\overline{X}_n$ instead of $\overline{X}_{a,n}$.

▶ **Theorem 32** ([18, Theorem 4]). *Fix an arbitrary $\delta > 0$ and let $\Delta_n = 9\sqrt{2n\ln(2/\delta)}$. Let n_0 be such that $\sqrt{n_0} \geq O(|A|(C_p^2 \ln n_0 + N_0(1/2))$. Then under the drift conditions, for any $n \geq n_0$, the following holds true:*

$$\mathbb{P}[n\overline{X}_n \geq n\mathbb{E}[\overline{X}_n] + \Delta_n(\delta)] \leq \delta$$

$$\mathbb{P}[n\overline{X}_n \leq n\mathbb{E}[\overline{X}_n] - \Delta_n(\delta)] \leq \delta$$

Finally, the authors argue that the probability of making the wrong decision (choosing a suboptimal action) converges to 0 as the number of plays grows:

▶ **Theorem 33** ([18, Theorem 5]). *Let I_t be the action chosen at time t, and let a^* be the optimal action. Then $\lim_{t\to\infty} Pr(I_t \neq a^*) = 0$.*

C MCTS with Simulation

After n iterations of MCTS, we have $\mathsf{total}_n(p) = \sum_{i|I_i(p)\leq n} \mathsf{reward}_{I_i(p)}(p)$ and $\mathsf{total}(p,a) = \sum_{i|I_i(p,a)\leq n} \mathsf{reward}_{I_i(p,a)}(p,a)$.

We use the following observations, derived from the structure of the MCTS algorithm. For all nodes p in the search tree, after n iterations, we have:

$$\begin{aligned}
\mathsf{total}_n(p) &= \mathsf{reward}_{\mathcal{I}_1(p)}(p) + \sum_{a\in A} \mathsf{total}_n(p,a) \\
\mathsf{total}_n(p,a) &= \sum_{s\in \mathsf{Supp}(P(\mathsf{last}(p),a)} \mathsf{total}_n(p\cdot as) + R(\mathsf{last}(p),a)\cdot \mathsf{count}_n(p,a) \\
\mathsf{value}_n(p) &= \frac{\mathsf{total}_n(p)}{\mathsf{count}_n(p)} \\
\mathsf{count}_n(p) &= 1 + \sum_a \mathsf{count}_n(p,a) \\
\mathsf{count}_n(p,a) &= \sum_s \mathsf{count}_n(p\cdot as)
\end{aligned}$$

In the following proof we will abuse notations slightly and conflate the variables and counters used in MCTS with their associated random variables, *e.g.* we write $\mathbb{E}[\mathsf{value}_n(s_0)]$ instead of $\mathbb{E}[V_n(s_0)]$ with $V_n(s_0)$ a random variable that represents the value $\mathsf{value}_n(s_0)$.

Proof of Lemma 12. We use the following inequality (Chernoff-Hoeffding inequality)[15, Theorem 2] throughout the proof:

Let $X_1, X_2, \ldots X_n$ be independent random variables in $[0,1]$. Let $S_n = \sum_n X_i$. Then for all $a > 0$, $\mathbb{P}\Big[S_n \geq \mathbb{E}[S_n] + t\Big] \leq \exp\left(-\frac{2t^2}{n}\right)$ and $\mathbb{P}\Big[S_n \leq \mathbb{E}[S_n] - t\Big] \leq \exp\left(-\frac{2t^2}{n}\right)$.

We need to show that the following conditions hold:

1. $\lim_{\mathsf{count}_n(p)\to\infty} \mathbb{E}[\mathsf{value}_n(p,a)]$ exists for all a.
2. There exists a constant $C_p > 0$ such that for $\mathsf{count}_n(p,a)$ big enough and any $\delta > 0$, $\Delta_{\mathsf{count}_n(p,a)}(\delta) = C_p\sqrt{\mathsf{count}_n(p,a)\ln(1/\delta)}$, the following bounds hold:

$$\mathbb{P}\Big[\mathsf{total}_n(p,a) \geq \mathbb{E}[\mathsf{total}_n(p,a)] + \Delta_{\mathsf{count}_n(p,a)}(\delta)\Big] \leq \delta$$

$$\mathbb{P}\Big[\mathsf{total}_n(p,a) \leq \mathbb{E}[\mathsf{total}_n(p,a)] - \Delta_{\mathsf{count}_n(p,a)}(\delta)\Big] \leq \delta$$

We show it by induction on $H - |p|$.

For $|P| = H - 1$: $\mathsf{reward}_i(p,a)$ follows a stationary distribution: $\mathsf{reward}_i(p,a) = R(\mathsf{last}(p),a) + R_T(s)$ with probability $P(\mathsf{last}(p),a)(s)$. Thus

$$\begin{aligned}\mathbb{E}[\mathsf{total}_n(p,a)] &= \mathbb{E}\left[\sum_{i|I_i(p,a)\leq n} \mathsf{reward}_{I_i(p,a)}(p,a)\right] \\ &= \mathsf{count}_n(p,a)\left(\sum_s R_T(s)P(\mathsf{last}(p),a)(s) + R(\mathsf{last}(p),a)\right).\end{aligned}$$

Thus $\mathbb{E}[\mathsf{value}_n(p,a)] = \sum_s R_T(s)P(\mathsf{last}(p),a)(s) + R(\mathsf{last}(p),a)$.

From the Chernoff-Hoeffding inequality,

$$\mathbb{P}\left[\sum_{i|I_i(p,a)\leq n} \mathsf{reward}_{I_i(p,a)}(p,a) \geq \mathbb{E}\left[\sum_{i|I_i(p,a)\leq n} \mathsf{reward}_{I_i(p,a)}(p,a)\right] + \sqrt{\frac{\mathsf{count}_n(p,a)}{2}\ln\frac{1}{\delta}}\right] \leq \delta\,,$$

$$\mathbb{P}\left[\sum_{i|I_i(p,a)\leq n} \mathsf{reward}_{I_i(p,a)}(p,a) \leq \mathbb{E}\left[\sum_{i|I_i(p,a)\leq n} \mathsf{reward}_{I_i(p,a)}(p,a)\right] - \sqrt{\frac{\mathsf{count}_n(p,a)}{2}\ln\frac{1}{\delta}}\right] \leq \delta\,.$$

Therefore, condition 2 also holds with $C_p = \frac{1}{\sqrt{2}}$.

Assume that the conditions are true for all $p\cdot as$. Then, from Theorem 30 we get:

$$\begin{aligned}&\left|\mathbb{E}\left[\frac{\sum_{a'}\mathsf{total}_n(pas,a')}{\sum_{a'}\mathsf{count}_n(pas,a')}\right] - \lim_{\mathsf{count}_n(p\cdot as)\to\infty}\mathbb{E}\left[\mathsf{value}_n(p\cdot as,a^*)\right]\right| \\ &\leq \left|\mathbb{E}\left[\mathsf{value}_n(p\cdot as,a^*)\right] - \lim_{\mathsf{count}_n(p\cdot as)\to\infty}\mathbb{E}\left[\mathsf{value}_n(p\cdot as,a^*)\right]\right| + \mathcal{O}\left(\frac{\ln(\mathsf{count}_n(p\cdot as)-1)}{\mathsf{count}_n(p\cdot as)-1}\right),\end{aligned}$$

where a^* is the optimal action from $p\cdot as$. Now,

$$\begin{aligned}\lim_{\mathsf{count}_n(p)\to\infty}\mathbb{E}[\mathsf{value}_n(p\cdot as)] &= \lim_{\mathsf{count}_n(p)\to\infty}\mathbb{E}\left[\frac{\mathsf{total}_n(p\cdot as)}{\mathsf{count}_n(p\cdot as)}\right] \\ &= \lim_{\mathsf{count}_n(p)\to\infty}\mathbb{E}\left[\frac{\mathsf{total}_n(p\cdot as) - \mathsf{reward}_{\mathcal{I}_1(p)}(p\cdot as)}{\mathsf{count}_n(p\cdot as)-1}\right] \\ &= \lim_{\mathsf{count}_n(p)\to\infty}\mathbb{E}\left[\frac{\sum_{a'}\mathsf{total}_n(p\cdot as,a')}{\sum_{a'}\mathsf{count}_n(p\cdot as,a')}\right].\end{aligned}$$

Let $\lim_{\mathsf{count}_n(p\cdot as)\to\infty}\mathbb{E}[\mathsf{value}_n(p\cdot as,a^*)]$ be denoted by $\mu_{p\cdot as}$ (we know that this limit exists by the induction hypothesis). From Theorem 31, we know that $\mathsf{count}_n(p,a)\to\infty$ for all a when $\mathsf{count}_n(p)\to\infty$. And as for all states s, state s is chosen according to distribution $P(p,a)(s)$, $\mathsf{count}_n(p\cdot as)\to\infty$ with probability 1. Then,

$$\lim_{\mathsf{count}_n(p)\to\infty} \mathbb{E}[\mathsf{value}_n(p\cdot a)] = \lim_{\mathsf{count}_n(p)\to\infty} \mathbb{E}\left[\sum_s \mathsf{value}_n(p\cdot as)\frac{\mathsf{count}_n(p\cdot as)}{\mathsf{count}_n(p,a)} + R(\mathsf{last}(p),a)\right]$$
$$= R(\mathsf{last}(p),a) + \sum_s \mu_{p\cdot as}\cdot P(\mathsf{last}(p),a)(s)\,.$$

So $\lim_{\mathsf{count}_n(p)\to\infty} \mathbb{E}[\mathsf{value}_n(p\cdot a)]$ exists.

From Theorem 32, when $\mathsf{count}_n(p\cdot as)$ is big enough, for all $\delta>0$, $\mathbb{P}\Big[\sum_{a'}\mathsf{total}_n(pas,a') \geq \mathbb{E}[\sum_{a'}\mathsf{total}_n(pas,a')] + \Delta_1^s(\delta)\Big] \leq \frac{\delta}{2|S|}$ where $\Delta_1^s(\delta) = 9\left(\sqrt{\mathsf{count}_n(p\cdot as)\ln\left(\frac{4|S|}{\delta}\right)}\right)$.

Therefore $\mathbb{P}\Big[\mathsf{total}_n(p\cdot as) - \mathsf{reward}_{\mathcal{I}_1(p\cdot as)}(p\cdot as) \geq \mathbb{E}[\mathsf{total}_n(p\cdot as) - \mathsf{reward}_{\mathcal{I}_1(p\cdot as)}(p\cdot as)] + \Delta_1^s(\delta)\Big] \leq \frac{\delta}{2|S|}$. Also the random variable associated to $\mathsf{reward}_{\mathcal{I}_1(p\cdot as)}(p\cdot as)$ following a fixed stationary distribution $f(p)$ in $[0,1]$. So from the Chernoff-Hoeffding inequality, $\mathbb{P}\Big[\mathsf{reward}_{\mathcal{I}_1(p\cdot as)}(p\cdot as) \geq \mathbb{E}[\mathsf{reward}_{\mathcal{I}_1(p\cdot as)}(p\cdot as)] + \Delta_2(\delta)\Big] \leq \frac{\delta}{2|S|}$ where $\Delta_2(\delta) = \frac{1}{\sqrt{2}}\left(\sqrt{\ln\left(\frac{2|S|}{\delta}\right)}\right)$.

Now using the fact that for n random variables $\{A_i\}_{i\leq n}$ and n random variables $\{B_i\}_{i\leq n}$, $\mathbb{P}[\sum_i A_i \geq \sum_i B_i] \leq \sum_i \mathbb{P}[A_i \geq B_i]$, we get:

$$\mathbb{P}\Big[\mathsf{total}_n(p\cdot as) \geq \mathbb{E}[\mathsf{total}_n(p\cdot as)] + \Delta_1^s(\delta) + \Delta_2(\delta)\Big]$$
$$\leq \mathbb{P}\Big[\mathsf{total}_n(p\cdot as) - \mathsf{reward}_{\mathcal{I}_1(p\cdot as)}(p\cdot as) \geq \mathbb{E}[\mathsf{total}_n(p\cdot as) - \mathsf{reward}_{\mathcal{I}_1(p\cdot as)}(p\cdot as)] + \Delta_1^s(\delta)\Big]$$
$$+ \mathbb{P}\Big[\mathsf{reward}_{\mathcal{I}_1(p\cdot as)}(p\cdot as) \geq \mathbb{E}[\mathsf{reward}_{\mathcal{I}_1(p\cdot as)}(p\cdot as)] + \Delta_2(\delta)\Big] \leq \frac{\delta}{|S|}\,.$$

As $\mathsf{count}_n(p,a)\mathbb{E}[R(\mathsf{last}(p),a)] = \mathbb{E}[\mathsf{count}_n(p,a)\cdot R(\mathsf{last}(p),a)]$,

$$\mathbb{P}\left[\mathsf{total}_n(p,a) \geq \mathbb{E}[\mathsf{total}_n(p,a)] + \sum_s(\Delta_1^s(\delta) + \Delta_2(\delta))\right]$$
$$\leq \sum_s \mathbb{P}\Big[\mathsf{total}_n(p\cdot as) \geq \mathbb{E}[\mathsf{total}_n(p\cdot as)] + (\Delta_1^s(\delta) + \Delta_2(\delta))\Big] \leq \delta\,.$$

Similarly, when $\mathsf{count}_n(p\cdot as)$ is big enough, for all $\delta>0$ it holds that $\mathbb{P}\Big[\mathsf{total}_n(p\cdot as) - \mathsf{reward}_{\mathcal{I}_1(p\cdot as)}(p\cdot as) \leq \mathbb{E}[\mathsf{total}_n(p\cdot as) - \mathsf{reward}_{\mathcal{I}_1(p\cdot as)}(p\cdot as)] - \Delta_1^s(\delta)\Big] \leq \frac{\delta}{2|S|}$ and $\mathbb{P}\Big[\mathsf{reward}_{\mathcal{I}_1(p\cdot as)}(p\cdot as) \leq \mathbb{E}[\mathsf{reward}_{\mathcal{I}_1(p\cdot as)}(p\cdot as)] - \Delta_2(\delta)\Big] \leq \frac{\delta}{2|S|}$.

Thus $\mathbb{P}\Big[\mathsf{total}_n(p,a) \leq \mathbb{E}[\mathsf{total}_n(p,a)] - \sum_s(\Delta_1^s(\delta) + \Delta_2(\delta))\Big] \leq \delta$.

As $\mathsf{count}_n(p\cdot as) \leq \mathsf{count}_n(p,a)$, there exists $C\in\mathbb{N}$ such that for $\mathsf{count}_n(p\cdot as)$ big enough and for all $\delta>0$:

$$\sum_s(\Delta_1^s(\delta) + \Delta_2(\delta)) \leq C\sum_s\sqrt{\mathsf{count}_n(p\cdot as)\ln\left(\frac{1}{\delta}\right)}$$
$$\leq C\sum_s\sqrt{\mathsf{count}_n(p,a)\ln\left(\frac{1}{\delta}\right)}$$
$$\leq C|S|\sqrt{\mathsf{count}_n(p,a)\ln\left(\frac{1}{\delta}\right)}\,.$$

So, there is a constant C_p such that for $\mathsf{count}_n(p,a)$ big enough and any $\delta > 0$, it holds that $\Delta_{\mathsf{count}_n(p,a)}(\delta) = C_p\sqrt{\mathsf{count}_n(p,a)\ln(1/\delta)} \geq \sum_s(\Delta_1^s(\delta) + \Delta_2(\delta))$. Therefore, the following bound hold: $\mathbb{P}\Big[\mathsf{total}_n(p,a) \geq \mathbb{E}[\mathsf{total}_n(p,a)] + \Delta_{\mathsf{count}_n(p,a)}(\delta)\Big]$ is upper bounded by $\mathbb{P}\Big[\mathsf{total}_n(p,a) \geq \mathbb{E}[\mathsf{total}_n(p,a)] + \sum_s(\Delta_1^s(\delta) + \Delta_2(\delta))\Big]$. It follows that $\mathbb{P}\Big[\mathsf{total}_n(p,a) \geq \mathbb{E}[\mathsf{total}_n(p,a)] + \Delta_{\mathsf{count}_n(p,a)}(\delta)\Big] \leq \delta$. Similarly, $\mathbb{P}\Big[\mathsf{total}_n(p,a) \leq \mathbb{E}[\mathsf{total}_n(p,a)] - \Delta_{\mathsf{count}_n(p,a)}(\delta)\Big]$ is upper bounded by $\mathbb{P}\Big[\mathsf{total}_n(p,a) \leq \mathbb{E}[\mathsf{total}_n(p,a)] - \sum_s(\Delta_1^s(\delta) + \Delta_2(\delta))\Big]$. It follows that $\mathbb{P}\Big[\mathsf{total}_n(p,a) \leq \mathbb{E}[\mathsf{total}_n(p,a)] - \Delta_{\mathsf{count}_n(p,a)}(\delta)\Big] \leq \delta$.

This proves that for any p, the sequences $(x_{a,t})_{t\geq 1}$ associated with $\mathsf{reward}_{\mathcal{I}_t(p,a)}(p)$ satisfy the drift conditions. ◀

The Big-O Problem for Labelled Markov Chains and Weighted Automata

Dmitry Chistikov
Centre for Discrete Mathematics and its Applications (DIMAP) and
Department of Computer Science, University of Warwick, Coventry, UK

Stefan Kiefer
Department of Computer Science, University of Oxford, UK

Andrzej S. Murawski
Department of Computer Science, University of Oxford, UK

David Purser
Centre for Discrete Mathematics and its Applications (DIMAP) and
Department of Computer Science, University of Warwick, Coventry, UK
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

Given two weighted automata, we consider the problem of whether one is big-O of the other, i.e., if the weight of every finite word in the first is not greater than some constant multiple of the weight in the second.

We show that the problem is undecidable, even for the instantiation of weighted automata as labelled Markov chains. Moreover, even when it is known that one weighted automaton is big-O of another, the problem of finding or approximating the associated constant is also undecidable.

Our positive results show that the big-O problem is polynomial-time solvable for unambiguous automata, **coNP**-complete for unlabelled weighted automata (i.e., when the alphabet is a single character) and decidable, subject to Schanuel's conjecture, when the language is bounded (i.e., a subset of $w_1^* \dots w_m^*$ for some finite words $w_1, \dots, w_m$).

On labelled Markov chains, the problem can be restated as a ratio total variation distance, which, instead of finding the maximum difference between the probabilities of any two events, finds the maximum ratio between the probabilities of any two events. The problem is related to ϵ-differential privacy, for which the optimal constant of the big-O notation is exactly $\exp(\epsilon)$.

2012 ACM Subject Classification Theory of computation → Probabilistic computation

Keywords and phrases weighted automata, labelled Markov chains, probabilistic systems

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.41

Related Version A full version of the paper is available at `https://arxiv.org/abs/2007.07694`.

Funding *Dmitry Chistikov*: Supported in part by the Royal Society International Exchanges scheme (IEC\R2\170123).
Stefan Kiefer: Supported by a Royal Society Research Fellowship.
Andrzej S. Murawski: Supported by a Royal Society Leverhulme Trust Senior Research Fellowship and the International Exchanges Scheme (IE161701).
David Purser: Supported by the UK EPSRC Centre for Doctoral Training in Urban Science (EP/L016400/1) and in part by the Royal Society International Exchanges scheme (IEC\R2\170123).

Acknowledgements The authors would like to thank to Engel Lefaucheux, Joël Ouaknine, and James Worrell for discussions during the development of this work.

1 Introduction

Weighted automata over finite words are a well-known and powerful model of computation, a quantitative analogue of finite-state automata. Special cases of weighted automata include nondeterministic finite automata and labelled Markov chains, two standard formalisms for modelling systems and processes. Algorithms for analysis of weighted automata have been studied both in the early theory of computing and more recently by the infinite-state systems and algorithmic verification communities.

Given two weighted automata $\mathcal{A}$, $\mathcal{B}$ over an algebraic structure $(\mathcal{S}, +, \times)$, the equivalence problem asks whether the two associated functions $f_{\mathcal{A}}, f_{\mathcal{B}} \colon \Sigma^* \to \mathcal{S}$ are equal: $f_{\mathcal{A}}(w) = f_{\mathcal{B}}(w)$ for all finite words w over the alphabet Σ. Over the ring $(\mathbb{Q}, +, \times)$, equivalence is decidable in polynomial time by the results of Schützenberger [34] and Tzeng [38]; subsequently, fast parallel (**NC** and **RNC**) algorithms have been found for this problem [39, 20]. In contrast, for semirings the equivalence problem is hard: undecidable [21, 1] for the semiring $(\mathbb{Q}, \max, +)$ and **PSPACE**-hard [28] for the Boolean semiring (for which weighted automata are usual nondeterministic finite automata and equivalence is equality of recognized languages). Replacing $=$ with $\leq$ makes the problem harder: even for the ring $(\mathbb{Q}, +, \times)$ the question of whether $f_{\mathcal{A}}(w) \leq f_{\mathcal{B}}(w)$ for all $w \in \Sigma^*$ is undecidable – even if $f_{\mathcal{A}}$ is constant [31]. This problem subsumes the universality problem for (Rabin) probabilistic automata, yet another subclass of weighted automata (see, e.g., [12]).

In this paper, we introduce and study another natural problem, in which the ordering is relaxed from exact (in)equality to (in)equality to within a constant factor. Given $\mathcal{A}$ and $\mathcal{B}$ as above, is it true that there exists a constant $c > 0$ such that

$$f_{\mathcal{A}}(w) \leq c \cdot f_{\mathcal{B}}(w) \qquad \text{for all } w \in \Sigma^* \,?$$

Using standard mathematical notation, this condition asserts that $f_{\mathcal{A}}(w) = O(f_{\mathcal{B}}(w))$ as $|w| \to \infty$, and we refer to this problem as the *big-O* problem accordingly.[1] The *big-*Θ problem (which turns out to be computationally equivalent to the big-O problem), in line with the $\Theta(\cdot)$ notation in analysis of algorithms, asks whether $f_{\mathcal{A}} = O(f_{\mathcal{B}})$ and $f_{\mathcal{B}} = O(f_{\mathcal{A}})$.

We restrict our attention to the ring $(\mathbb{Q}, +, \times)$ and only consider *non-negative weighted automata*, i.e., those in which all transitions have non-negative weights. We remark that, even under this restriction, weighted automata still form a superclass of (Rabin) probabilistic automata, a non-trivial and rich model of computation. Our initial motivation to study the big-O problem came from yet another formalism, labelled Markov chains (LMCs). One can think of the semantics of LMCs as giving a probability distribution or subdistribution on the set of all finite words. LMCs, often under the name Hidden Markov Models, are widely employed in a diverse range of applications; in computer-aided verification, they are perhaps the most fundamental model for probabilistic systems, with model-checking tools such as Prism [22] or Storm [10] based on analyzing LMCs efficiently. All the results in our paper (including hardness results) hold for LMCs too. Our main findings are as follows.

- The big-O problem for non-negative WA and LMCs turns out to be **undecidable in general**, by a reduction from nonemptiness for probabilistic automata.
- For **unambiguous automata**, i.e., where every word has at most one accepting path, the big-O problem becomes decidable and can be solved in polynomial time.

[1] There also exists a related but slightly different definition of big-O; see Remark 12 for details on the corresponding version of our big-O problem.

- In the **unary case**, i.e., if the input alphabet Σ is a singleton, the big-O problem is also decidable and, in fact, complete for the complexity class **coNP**. Unary LMCs are a simple and pure probabilistic model of computation: they run in discrete time and can terminate at any step; the big-O problem refers to this termination probability in two LMCs (or two WA). Our upper bound argument refines an analysis of growth of entries in powers of non-negative matrices by Friedland and Schneider [33], and the lower bound is obtained by a reduction from unary NFA universality [37].
- In a more general **bounded case**, i.e., if the languages of all words w associated with non-zero weight are included in $w_1^* w_2^* \dots w_m^*$ for some finite words $w_1, \dots, w_m \in \Sigma^*$ (that is, are *bounded in the sense of Ginsburg and Spanier*; see [16, Chapter 5] and [17]), the big-O problem is decidable subject to Schanuel's conjecture. This is a well-known conjecture in transcendental number theory [23], which implies that the first-order theory of the real numbers with the exponential function is decidable [24]. Intuitively, our reliance on this conjecture is linked to the expressions for the growth rate in powers of non-negative matrices. These expressions are sums of terms of the form $\rho^n \cdot n^k$, where n is the length of a word, $k \in \mathbb{N}$, and ρ is an algebraic number. Our algorithms (however implicitly) need to compare for equality pairs of real numbers of the form $\log \rho_1 / \log \rho_2$, where ρ_i are algebraic, and it is an open problem in number theory whether there is an effective procedure for this task (the four exponentials conjecture asks whether two such ratios can ever be equal; see, e.g., Waldschmidt [40, Sections 1.3 and 1.4]).
  Bounded languages form a well-known subclass of regular languages. In fact, a regular (or even context-free) language L is bounded if and only if the number of words of length n in L is at most polynomial in n. All other regular languages have, in contrast, exponential growth rate (a fact rediscovered multiple times; see, e.g., references in Gawrychowski et al. [14]). Bounded languages have been studied from combinatorial and algorithmic points of view since the 1960s [17, 14], and have recently been used, e.g., in the analysis of quantitative information flow problems in computer security [27, 26]. In the context of labelled Markov chains, languages that are subsets of $a_1^* a_2^* \dots a_m^*$ (for individual letters $a_1, \dots, a_m \in \Sigma$) model consecutive arrival of m events in a discrete-time system. It is curious that natural decision problems for such simple systems can lead to intricate algorithmic questions in number theory at the border of decidability.

Further motivation and related work

In the labelled Markov chain setting, the big-O problem can be reformulated as a boundedness problem for the following function. For two LMCs $\mathcal{A}$ and $\mathcal{B}$, define the (asymmetric) *ratio variation function* by

$$r(\mathcal{A}, \mathcal{B}) = \sup_{E \subseteq \Sigma^*} (f_{\mathcal{A}}(E) / f_{\mathcal{B}}(E)),$$

where $f_{\mathcal{A}}(E)$ and $f_{\mathcal{B}}(E)$ denote the total probability mass associated with an arbitrary set of finite words $E \subseteq \Sigma^*$ in $\mathcal{A}$ and $\mathcal{B}$, respectively. Here we assume $\frac{0}{0} = 0$ and $\frac{x}{0} = \infty$ for $x > 0$. Observe that, because $\max(\frac{a}{b}, \frac{c}{d}) \geq \frac{a+c}{b+d}$ for $a, b, c, d \geq 0$, the supremum over $E \subseteq \Sigma^*$ can be replaced with supremum over $w \in \Sigma^*$. Consequently, the big-O problem for LMCs is equivalent to deciding whether $r(\mathcal{A}, \mathcal{B}) < \infty$.

Finding the value of r amounts to asking for the optimal (minimal) constant in the big-O notation. Further, one can consider a symmetric variant, the *ratio distance*: $rd(\mathcal{A}, \mathcal{B}) = \max\{r(\mathcal{A}, \mathcal{B}), r(\mathcal{B}, \mathcal{A})\}$, in an analogy with big-Θ. Now, rd is a ratio-oriented variant of the classic *total variation distance* tv, defined by $tv(\mathcal{A}, \mathcal{B}) = \sup_{E \subseteq \Sigma^*}(f_{\mathcal{A}}(E) - f_{\mathcal{B}}(E))$, which is

a well-established way of comparing two labelled Markov chains [5, 19]. We also consider the problem of approximating r (as well as rd) to a given precision and the problem of comparing it with a given constant (threshold problem), showing that both are undecidable.

The ratio distance rd is also equivalent to the exponential of the *multiplicative total variation distance* defined in [4, 36] in the context of differential privacy. Consider a system $\mathcal{M}$, modelled by a single labelled Markov chain, where output words are observable to the environment but we want to protect the privacy of the starting configuration. Let $R \subseteq Q \times Q$ be a symmetric relation, which relates the starting configurations intended to remain indistinguishable. Given $\epsilon \geq 0$, we say that $\mathcal{M}$ is ϵ-differentially private (with respect to R) if, for all $(s, s') \in R$, we have $f_s(E) \leq e^\epsilon \cdot f_{s'}(E)$ for every observable set of traces $E \subseteq \Sigma^*$ [11, 6]. **Here in the subscript of f and elsewhere, references to states s and s' replace references to LMCs/automata: $\mathcal{M}$ stays implicit, and we specify which state it is executed from.** Note that there exists such an ϵ if and only if $r(s, s') < \infty$ for all $(s, s') \in R$ or, equivalently, (the LMC $\mathcal{M}$ executed from) s is big-O of (the LMC $\mathcal{M}$ executed from) s' for all $(s, s') \in R$. In fact, the minimal such ϵ satisfies $e^\epsilon = \max_{(s,s')\in R} r(s, s')$, thus r captures the level of differential privacy between s and s'.

Our results show that even deciding whether the multiplicative total variation distance is finite or $+\infty$ is, in general, impossible. Likewise, it is undecidable whether a system modelled by a labelled Markov chain provides any degree of differential privacy, however low.

2 Preliminaries

▶ **Definition 1.** *A* weighted automaton $\mathcal{W}$ *over the* $(\mathbb{Q}, +, \times)$ *semi-ring is a 4-tuple* $\langle Q, \Sigma, M, F\rangle$*, where* Q *is a finite set of* states*,* Σ *is a finite* alphabet*,* $M : \Sigma \to \mathbb{Q}^{Q\times Q}$ *is a* transition weighting function*, and* $F \subseteq Q$ *is a set of* final states*. We consider only non-negative weighted automata, i.e.* $M(a)(q, q') \geq 0$ *for all* $a \in \Sigma$ *and* $q, q' \in Q$*.*

In complexity-theoretic arguments, we assume that each weight is given as a pair of integers (numerator and denominator) in binary. The description size is then the number of bits required to represent $\langle Q, \Sigma, M, F\rangle$, including the bit size of the weights.

Each weighted automaton defines functions $f_s : \Sigma^* \to \mathbb{R}$, where for all $s \in Q$

$$f_s(w) = \sum_{t\in F} (M(a_1) \times M(a_2) \times \cdots \times M(a_n))_{s,t} \qquad \text{for } w = a_1 a_2 \dots a_n \in \Sigma^*$$

and $A \times B$ is standard matrix multiplication. We refer to $f_s(w)$ as *the weight of* w *from state* s. Without loss of generality, a weighted automaton can have a single final state. If not, introduce a new unique final state t s.t. $M(a)(q, t) = \sum_{q'\in F} M(a)(q, q')$ for all $q \in Q$,$a \in \Sigma$.

▶ **Definition 2.** *We denote by* $\mathcal{L}_s(\mathcal{W})$ *the set of* $w \in \Sigma^*$ *with* $f_s(w) > 0$*, that is, with positive weight from* s*. Equivalently, this is the language of* $\mathcal{N}_s(\mathcal{W})$*, the non-deterministic finite automaton (NFA) formed from the same set of states (and final states) as* $\mathcal{W}$*, start state* s*, and transitions* $q \xrightarrow{a} q'$ *whenever* $M(a)(q, q') > 0$*.*

Given $s, s' \in Q$, we say that ***s is big-O of s'*** if there exists $C > 0$ such that $f_s(w) \leq C \cdot f_{s'}(w)$ for all $w \in \Sigma^*$. The paper studies the following problem.

▶ **Definition 3** (Big-O Problem)**.**

INPUT	*Weighted automaton* $\langle Q, \Sigma, M, F\rangle$ *and* $s, s' \in Q$
OUTPUT	*Is* s *big-O of* s'*?*

▶ Remark 4. One could consider whether s is big-Θ of s', defined as s is big-O of s' and s' is big-O of s; equivalently, whether $rd(s,s') < \infty$ for LMCs. We note that these two notions reduce to each other, justifying our consideration of only the big-O problem. There is an obvious reduction from big-Θ to big-O making two oracle calls (a Cook reduction), but this can be strengthened to a single call preserving the answer (a Karp reduction). This, however, requires at least two characters. In the other direction, one can ask if s big-O of s' using big-Θ by asking if a linear combination of s and s' is big-Θ of s'.

In the paper we also work with labelled Markov chains. In particular, they will appear in examples and hardness (including undecidability) arguments. As they are a special class of weighted automata, this will imply hardness (resp. undecidability) for weighted automata in general. On the other hand, our decidability results will be phrased using weighted automata, which makes them applicable to labelled Markov chains.

▶ **Definition 5.** *A* labelled Markov chain *(LMC) is a (non-negative) weighted automaton* $\langle Q, \Sigma, M, F\rangle$ *such that, for all* $q \in Q \setminus F$*, we have* $\sum_{q' \in Q} \sum_{a \in \Sigma} M(a)(q,q') = 1$ *and* $M(a)(q,q') = 0$ *for all for all* $a \in \Sigma, q \in F$ *and* $q' \in Q$*.*

Since final states have no outgoing transitions, w.l.o.g., one can assume a unique final state. For LMCs, the function f_s can be extended to a measure on the powerset of Σ^* by $f_s(E) = \sum_{w \in E} f_s(w)$, where $E \subseteq \Sigma^*$. The measure is a subdistribution: $\sum_{w \in \Sigma^*} f_s(w) \leq 1$.

We will also consider *unary* weighted automata, and similarly LMCs, where $|\Sigma| = 1$. Then we will often omit Σ on the understanding that $\Sigma = \{a\}$, and describe transitions with a single matrix $A = M(a)$ so that $f_s(a^n) = A^n_{s,t}$, where t is the unique final state. Note that $A^n_{s,t}$ stands for $(A^n)(s,t)$, and not $(A(s,t))^n$. Using the notation of regular expressions, we can write $\mathcal{L}_s(\mathcal{W}) \subseteq a^*$. It will turn out fruitful to consider several larger classes of languages:

▶ **Definition 6.** *Let* $L \subseteq \Sigma^*$*. L is* bounded *[17] if* $L \subseteq w_1^* w_2^* \cdots w_m^*$ *for some* $w_1, \ldots, w_m \in \Sigma^*$*. L is* letter-bounded *if* $L \subseteq a_1^* a_2^* \ldots a_m^*$ *for some* $a_1, \ldots, a_m \in \Sigma$*. L is* plus-letter-bounded *if* $L \subseteq a_1^+ a_2^+ \ldots a_m^+$ *for some* $a_1, \ldots, a_m \in \Sigma$*.*

In each case, if the language of an NFA is suitably bounded, one can extract a corresponding bounding regular expression [14].

3 Big-O, Threshold and Approximation problems are undecidable

We show that the big-O problem is undecidable. We also establish undecidability for several other problems related to computing and approximating the ratio variation distance. Recall that this corresponds to identifying the optimal constant for positive instances of the big-O problem or the level of differential privacy between two states in a labelled Markov chain.

▶ **Definition 7.** *The* asymmetric threshold problem *takes an LMC along with two states* s, s' *and a constant* θ*, and asks if* $r(s,s') \leq \theta$*. The variant under the promise of boundedness promises that* $r(s,s') < \infty$*. The strict variant of each problem replaces* $\leq$ *with* $<$*.*

The asymmetric additive approximation task *takes an LMC, two states* s, s' *and a constant* γ*, and asks for* x *such that* $|r(s,s') - x| \leq \gamma$*. The* asymmetric multiplicative approximation task *takes an LMC, two states* s, s' *and a constant* γ*, and asks for* x *such that* $1 - \gamma \leq \frac{x}{r(s,s')} \leq 1 + \gamma$*.*

In each case, the symmetric variant is obtained by replacing r *with* rd*.*

▶ **Theorem 8.**

- *The big-O problem is undecidable, even for LMCs.*
- *Each variant of the threshold problem (asymmetric/symmetric, non-strict/strict) is undecidable, even under the promise of boundedness.*
- *All variants of the approximation tasks (asymmetric/symmetric, additive/multiplicative) are unsolvable, even under the promise of boundedness.*

Probabilistic automata are similar to LMCs, except that $M(a)$ is stochastic for every a, rather than $\sum_{a\in\Sigma} M(a)$ being stochastic. Formally, a *probabilistic automaton* is a *non-negative weighted automaton* with a distinguished start state q_s such that $\sum_{q'\in Q} M(a)(q,q') = 1$ for all $q \in Q$ and $a \in \Sigma$. The problem Empty asks if $f_{q_s}(w) \leq \frac{1}{2}$ for all words w. It is known to be undecidable [31, 12].

Proof sketch of Theorem 8. We reduce from Empty. The construction creates two branches of a labelled Markov chain. The first simulates the probabilistic automaton using the original weights multiplied by a scalar ($\frac{1}{4}$ in the case $|\Sigma| = 2$). The other branch will process each letter from Σ with equal weight (also $\frac{1}{4}$ in an infinite loop). Consequently, if there is a word accepted with probability greater than $\frac{1}{2}$, the ratio between the two branches will be greater than 1. The construction will enable words to be processed repeatedly, so that the ratio can then be pumped unboundedly. Certain linear combinations of the branches enable a gap promise, entailing undecidability of the threshold and approximation tasks. ◀

▶ Remark. The classic *non-strict* threshold problem for the total variation distance (i.e. whether $tv(s,s') \leq \theta$) is known to be undecidable [19], like our distances. However, it is not known if its strict variant (i.e. whether $tv(s,s') < \theta$) is also undecidable. In contrast, in our case, both variants are undecidable. Further note that (additive) approximation of tv is possible [19, 5], but this is not the case for our distances r and rd.

▶ Remark. We have shown the undecidability of the big-O problem using the undecidability of the emptiness problem for probabilistic automata. Another proof of undecidability can be obtained using the Value-1 problem (shown to be undecidable in [15]): indeed the big-O problem and the Value-1 problem are interreducible. However, the reduction from big-O to Value-1 does not entail decidability for subclasses of weighted automata (such as those with bounded languages), as the image of these subclasses does not fall into the known decidable fragments of the Value-1 problem. Further details are available in the full version.

4 The LC condition

Towards decidability results, we identify a simple necessary (but insufficient) condition for s being big-O of s'.

▶ **Definition 9** (LC condition). *A weighted automaton $\mathcal{W} = \langle Q, \Sigma, M, F\rangle$ and $s, s' \in Q$ satisfy the* language containment condition *(LC) if for all words w with $f_s(w) > 0$ we also have $f_{s'}(w) > 0$. Equivalently, $\mathcal{L}_s(\mathcal{W}) \subseteq \mathcal{L}_{s'}(\mathcal{W})$.*

The condition can be verified by constructing NFA $\mathcal{N}_s(\mathcal{W}), \mathcal{N}_{s'}(\mathcal{W})$ that accept $\mathcal{L}_s(\mathcal{W})$ and $\mathcal{L}_{s'}(\mathcal{W})$ respectively and verifying $\mathcal{L}(\mathcal{N}_s(\mathcal{W})) \subseteq \mathcal{L}(\mathcal{N}_{s'}(\mathcal{W}))$.

▶ Remark 10. Recall that NFA language containment is **NL**-complete if the automata are in fact deterministic, in **P** if they are unambiguous [8, Theorem 3], **coNP**-complete if they are unary [37] and **PSPACE**-complete in general [28]. In all cases this complexity level will match, or be lower than that for our respective algorithm for the big-O problem.

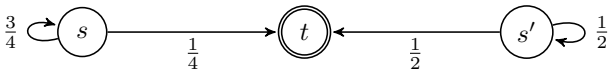


Figure 1 Unbounded ratio but language equivalent.

We observe that, if s is big-O of s', the LC condition must hold and so the LC condition is the first step in each of our verification routines. Example 11 shows that the condition alone is not sufficient to solve the big-O problem, because two states can admit the same set of words with non-zero weight, yet the weight ratios become unbounded.

▶ **Example 11.** Consider the unary automaton $\mathcal{W}$ in Figure 1. We have $\mathcal{L}_s(\mathcal{W}) = \mathcal{L}_{s'}(\mathcal{W}) = \{a^n \mid n \geq 1\}$, but $\frac{f_s(a^n)}{f_{s'}(a^n)} = \frac{(0.75)^{n-1} \cdot 0.25}{(0.5)^{n-1} \cdot 0.5} = 0.5 \cdot 1.5^{n-1} \xrightarrow[n\to\infty]{} \infty$.

▶ Remark 12. The original big-O notation on $f, g : \mathbb{N} \to \mathbb{N}$, states that f is $O(g)$ if $\exists C, k > 0$ $\forall n > k$ $f(n) \leq C\, g(n)$. Despite excluding finitely many points, when $g(n) \geq 1$, it is equivalent to $\exists C > 0$ $\forall n > 0$ $f(n) \leq C\, g(n)$ by taking C large enough to deal with the finite prefix.

In the paper, though, we formally consider s to not be big-O of s' if there exists even a single word w such that $f_s(w) > 0$ and $f_{s'}(w) = 0$. However, for weighted automata, we could amend our definition to "eventually big-O" as follows: $\exists C > 0, k > 0 : \forall w \in \Sigma^{\geq k}$ $f_s(w) \leq C \cdot f_{s'}(w)$.

The big-O problem reduces to its eventual variant by checking both the LC condition and the eventually big-O condition. Thus our undecidability (and hardness) results transfer to the eventually big-O problem. The eventually big-O problem can be solved via the big-O problem by "fixing" the LC condition through the addition of a branch from s' that accepts all appropriate words with very low probability. Further details are available in the full version.

4.1 Application: unambiguous weighted automata

In this section, we prove the first decidability result, that is, polynomial-time solvability in the unambiguous case. We say a weighted automaton $\mathcal{W}$ is *unambiguous from a state s* if every word has at most one accepting path in $\mathcal{N}_s(\mathcal{W})$.

▶ **Lemma 13.** *If a weighted automaton $\mathcal{W}$ is unambiguous from states s and s', the big-O problem is decidable in polynomial time.*

Proof sketch. We construct a product weighted automaton, with edge weights of the form $M'(a)((q_1, q_1'), (q_2, q_2')) = \frac{M(a)(q_1, q_2)}{M(a)(q_1', q_2')}$ and ask if there is a cycle on a path from (s, s') to (t, t) with weight > 1, which can be detected in polynomial time using a variation on the Bellman-Ford algorithm. ◀

Note the relevant behaviours are those on cycles – transitions which are taken at most once are of little significance to the big-O problem. Such transitions have at most a constant multiplicative effect on the ratio. This is the case whether or not the system is unambiguous.

5 The big-O problem for unary weighted automata is coNP-complete

In this section we show **coNP**-completeness in the unary case.

▶ **Theorem 14.** *The big-O problem for unary weighted automata is* **coNP***-complete. It is* **coNP***-hard even for unary labelled Markov chains.*

For the upper bound, our analysis will refine the analysis of the growth of powers of non-negative matrices of Friedland and Schneider [13, 33] which gives the asymptotic order of growth of $A^n_{s,t} + A^{n+1}_{s,t} + \cdots + A^{n+q}_{s,t} \approx \rho^n n^k$ for some ρ, k and q, which smooths over the periodic behaviour (see Theorem 18). Our results require a non-smoothed analysis, valid for each n. This isn't provided in [13, 33], where the smoothing forces the existence of a single limit – which we don't require. Our big-Θ lemma (Lemma 21) will accurately characterise the asymptotic behaviour of $A^n_{s,t}$ by exhibiting the correct value of ρ and k for every word.

5.1 Preliminaries

Let $\mathcal{W}$ be a unary non-negative weighted automaton with states Q, transition matrix A and a unique final state t. When we refer to a *path* in $\mathcal{W}$, we mean a path in the NFA of $\mathcal{W}$, i.e. paths only use transitions with non-zero weights and states on a path may repeat.

▶ **Definition 15.**

- *A state q can reach q' if there is a path from q to q'. In particular, any state q can always reach itself.*
- *A strongly connected component (SCC) $\varphi \subseteq Q$ is a maximal set of states such that for each $q, q' \in \varphi$, q can reach q'. We denote by $SCC(q)$ the SCC of state q and by A^φ, the $|\varphi| \times |\varphi|$ transition matrix of φ. Note every state is in a SCC, even if it is a singleton.*
- *The DAG of $\mathcal{W}$ is the directed acyclic graph of strongly connected components. Components φ, φ' are connected by an edge if there exist $q \in \varphi$ and $q' \in \varphi'$ with $A(q, q') > 0$.*
- *The spectral radius of an $m \times m$ matrix A is the largest absolute value of its eigenvalues. Recall the eigenvalues of A are $\{\lambda \in \mathbb{C} \mid$ exists vector $\vec{x} \in \mathbb{C}^m, \vec{x} \neq 0$ with $A\vec{x} = \lambda\vec{x}\}$. The spectral radius of φ, denoted by ρ_φ, is the spectral radius of A^φ. By $\rho(q)$ we denote the spectral radius of the SCC in which q is a member.*
- *We denote by T^φ the period of the SCC φ: the greatest common divisor of return times for some state $s \in \varphi$, i.e. $\gcd\{t \in \mathbb{N} \mid A^t(s, s) > 0\}$. It is known that any choice of state in the SCC gives the same value (see e.g. [35, Theorem 1.20]). If $A^\varphi = [0]$ then $T^\varphi = 0$.*
- *Let $\mathscr{P}(s, s')$ be the set of paths from the SCC of s to the SCC of s' in the DAG of $\mathcal{W}$. Thus a path $\pi \in \mathscr{P}(s, s')$ is a sequence of SCCs $\varphi_1, \ldots, \varphi_m$.*
- *$T(s, s')$, called the local period between s and s', is defined by $T(s, s') = \operatorname{lcm}_{\pi \in \mathscr{P}(s,s')} \gcd_{\varphi \in \pi} T^\varphi$.*
- *The spectral radius between states s and s', written $\rho(s, s')$, is the largest spectral radius of any SCC seen on a path from s to s': $\rho(s, s') = \max_{\pi \in \mathscr{P}(s,s')} \rho(\pi)$, where $\rho(\pi) = \max_{\varphi \in \pi} \rho_\varphi$ for $\pi \in \mathscr{P}(s, s')$.*
- *The following function captures the number of SCCs which attain the largest spectral radius on the path that has the most SCCs of maximal spectral radius. Let $k(s, s') = \max_{\pi \in \mathscr{P}(s,s')} k(\pi) - 1$, where, for $\pi \in \mathscr{P}(s, s')$, $k(\pi) = |\{\varphi \in \pi \mid \rho_\varphi = \rho(s, s')\}|$.*

▶ Remark 16. Since our weighted automata have rational weights, the spectral radius of an SCC is an algebraic number, as the absolute value of a root of a polynomial with rational coefficients. In general, an algebraic number $z \in \mathbb{A}$ can be represented by a tuple $(p_z, a, b, r) \in \mathbb{Q}[x] \times \mathbb{Q}^3$, where p_z is a polynomial over x and a, b, r specify an approximation to distinguish z from all other roots: z is the only root of $p_z(x)$ with $|z - (a + bi)| \leq r$. This representation, which admits standard operations (addition, multiplication, absolute value, (in)equality testing, etc.), can be found in polynomial time (see, e.g. [29]). Henceforth, when we refer to the spectral radius we will implicitly mean representation in this form.

The asymptotic behaviours of weighted automata will be characterised using (ρ, k)-pairs:

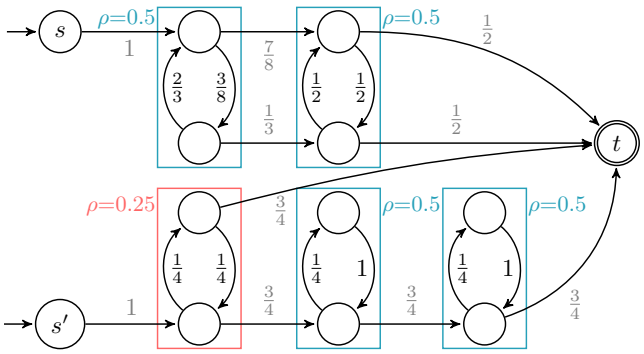


Figure 2 Different rates for different phases.

▶ **Definition 17.** *A (ρ, k)-pair is an element of $\mathbb{R}\times\mathbb{N}$. The ordering on $\mathbb{R}\times\mathbb{N}$ is lexicographic, i.e.* $(\rho_1, k_1) \leq (\rho_2, k_2) \iff \rho_1 < \rho_2 \vee (\rho_1 = \rho_2 \wedge k_1 \leq k_2)$.

Friedland and Schneider [13, 33] essentially use (ρ, k)-pairs to show the asymptotic behaviour of the powers of non-negative matrices. In particular they find the asymptotic behaviour of the sum of several $A^n_{s,s'}$, smoothing the periodic behaviour of the matrix.

▶ **Theorem 18** (Friedland and Schneider [13, 33])**.** *Let A be an $m \times m$ non-negative matrix, inducing a unary weighted automaton $\mathcal{W}$ with states $Q = \{1, \ldots, m\}$. Given $s, t \in Q$, let $B^n_{s,t} = A^n_{s,t} + A^{n+1}_{s,t} + \cdots + A^{n+T(s,t)-1}_{s,t}$. Then $\lim_{n\to\infty} \frac{B^n_{s,t}}{\rho(s,t)^n n^{k(s,t)}} = c, \quad 0 < c < \infty$.*

In the case where the local period is 1 ($T(s,t) = T(s',t) = 1$), Theorem 18 can already be used to solve the big-O problem (in particular if the matrix A is aperiodic). In this case $A^n_{s,t} = B^n_{s,t} = \Theta(\rho(s,t)^n n^{k(s,t)})$. Then to establish that s is big-O of s' we check that the language containment condition holds and that $(\rho(s,t), k(s,t)) \leq (\rho(s',t), k(s',t))$. However, this is not sufficient if the local period is not 1.

▶ **Example 19.** Consider the chains shown in Figure 2 with local period 2. The behaviour for $n \geq 3$ is $A^n_{s,t} = \Theta(0.5^n n)$ and $A^n_{s',t} = \Theta(0.25^n)$ when n is odd and $A^n_{s',t} = \Theta(0.5^n n)$ when n is even. However, Theorem 18 tells us $B^n_{s,t} = \Theta(0.5^n n)$ and $B^n_{s',t} = \Theta(0.5^n n)$ suggesting the ratio is bounded, but in fact s is not big-O s' (although s' is big-O of s) because $\frac{A^{2n+1}_{s,t}}{A^{2n+1}_{s',t}} \xrightarrow[n\to\infty]{} \infty$.

5.2 Upper bound: The unary big-O problem is in coNP

Let $\mathcal{W}$ be a unary weighted automaton and suppose we are asked whether s is big-O of s'. We assume w.l.o.g. (a) that there is a unique final state t with no outgoing transitions, and (b) that s, s' do not appear on any cycle (if this is not the case, copies of s, s' and their transitions can be taken).

Next we define a "degree function", which captures the asymptotic behaviour of each word a^n by a (ρ, k)-pair, capturing the exponential and polynomial behaviours respectively.

▶ **Definition 20.** *Given a unary weighted automaton $\mathcal{W}$, let $d_{s,t} : \mathbb{N} \to \mathbb{R}\times\mathbb{N}$ be defined by $d_{s,t}(n) = (\rho, k)$, where:*

- *ρ is the largest spectral radius of any vertex visited on any path of length n from s to t;*
- *the path from s to t that visits the most SCCs of spectral radius ρ visits $k+1$ such SCCs;*
- *if there is no length-n path from s to t, then (ρ, k)=$(0, 0)$.*

Let $s, t \in Q$ be fixed. We are now ready to state the key technical lemma of this subsection (cf. Theorem 18, Friedland and Schneider [13, 33]), where we assume the functions $\rho(n), k(n)$, defined by $d_{s,t}(n) = (\rho(n), k(n))$.

▶ **Lemma 21** (The big-Θ lemma). *There exist $c, C > 0$ such that, for every $n > |Q|$,*

$$c \cdot \rho(n)^n n^{k(n)} \leq A^n_{s,t} \leq C \cdot \rho(n)^n n^{k(n)}.$$

The set of *admissible* (ρ, k)-pairs is the image of $d_{s,t}$. Observe that this set is finite and of size at most $|Q|^2$: there can be no more than $|Q|$ values of ρ (if at worst each state were its own SCC) and the value of k is also bounded by the number of SCCs and thus $|Q|$.

We next define the (ρ, k)-annotated version of $\mathcal{W}$, i.e. in each state we record the relevant value of (ρ, k) corresponding to the current run to the state.

▶ **Definition 22** (The weighted automaton $\mathcal{W}^\dagger$). *Given $\mathcal{W} = \langle Q, \Sigma, A, \{t\}\rangle$ and $s \in Q$, the weighted automaton $\mathcal{W}^\dagger$ has states of the form (q, ρ, k) for all $q \in Q$ and all admissible (ρ, k)-pairs, the same Σ and no final states. For every transition $q \xrightarrow{p} q'$ from $\mathcal{W}$ denoting $A(q, q') = p$, include the following transition in $\mathcal{W}^\dagger$ for each admissible (ρ, k):*

- $(q, \rho, k) \xrightarrow{p} (q', \rho, k)$ *if* $SCC(q)$=$SCC(q')$,
- $(q, \rho, k) \xrightarrow{p} (q', \rho, k+1)$ *if* $SCC(q)\neq SCC(q')$ *and* $\rho = \rho(q')$,
- $(q, \rho, k) \xrightarrow{p} (q', \rho, k)$ *if* $SCC(q)\neq SCC(q')$ *and* $\rho > \rho(q')$,
- $(q, \rho, k) \xrightarrow{p} (q', \rho(q'), 0)$ *if* $SCC(q)\neq SCC(q')$ *and* $\rho(q') > \rho$.

$\mathcal{W}^\dagger$ is constructable in polynomial time given $\mathcal{W}$. Indeed, the spectral radii of all SCCs can be computed and compared to each other in time polynomial in the size of $\mathcal{W}$ (see Remark 16).

For the following lemma, recall the language containment (LC) condition from Definition 9 and the ordering on (ρ, k)-pairs from Definition 17.

▶ **Lemma 23.** *A state s is big-O of s' if and only if the LC condition holds and, for all but finitely many $n \in \mathbb{N}$, we have $d_{s,t}(n) \leq d_{s',t}(n)$.*

Proof sketch. Whenever $d_{s,t}(n) \leq d_{s',t}(n)$, by Lemma 21, we have $f_s(a^n) \leq (\frac{C}{c}(\frac{\rho}{\rho'})^n n^{k-k'}) \cdot f_{s'}(a^n)$, in which case either $d_{s,t}(n) = d_{s',t}(n)$ and $(\frac{\rho}{\rho'})^n n^{k-k'} = 1$ or $\lim_{n\to\infty}(\frac{\rho}{\rho'})^n n^{k-k'} = 0$ and so $(\frac{\rho}{\rho'})^n n^{k-k'} \leq 1$ for all but finitely many n.

However, whenever $d_{s,t}(n) > d_{s',t}(n)$, Lemma 21 yields $f_s(a^n) \geq (\frac{c}{C}(\frac{\rho}{\rho'})^n n^{k-k'}) \cdot f_{s'}(a^n)$ but then $\lim_{n\to\infty}(\frac{\rho}{\rho'})^n n^{k-k'} = \infty$. ◀

We are going to use the characterisation from Lemma 23 to prove Theorem 14. As already discussed, the LC condition can be checked via NFA inclusion testing. To tackle the "for all but finitely many ..." condition, we introduce the concept of eventual inclusion.

▶ **Definition 24.** *Given sets A, B, we say A is eventually included in B, written $A \subsetneqq B$, if and only if $A \setminus B$ is finite.*

The next three lemmas relate deciding the big-O problem using the characterisation of Lemma 23 to eventual inclusion.

▶ **Lemma 25.** *Given unary NFAs $\mathcal{N}_1, \mathcal{N}_2$, the problem $\mathcal{L}(\mathcal{N}_1) \subsetneqq \mathcal{L}(\mathcal{N}_2)$ is in* **coNP**.

▶ **Lemma 26.** *Suppose $d_1, d_2 : \mathbb{N} \to X$, with $(X, \leq)$ a finite total order. Then $d_1(n) \leq d_2(n)$ for all but finitely many n if and only if $\{n \mid d_1(n) \geq x\} \subsetneqq \{n \mid d_2(n) \geq x\}$ for all $x \in X$.*

▶ **Lemma 27.** *Given a unary weighted automaton $\mathcal{W}$, the associated problem whether $d_{s,t}(n) \leq d_{s',t}(n)$ for all but finitely many $n \in \mathbb{N}$ is in* **coNP**.

Proof. Given an admissible pair $x = (\rho, k)$, we construct an NFA $\mathcal{N}_{s,x}$ accepting $\{a^n \mid d_{s,t}(n) \geq x\}$ (similarly $\mathcal{N}_{s',x}$ for s'), by taking the NFA $\mathcal{N}_s(\mathcal{W}^\dagger)$ (Definitions 2, 22) with a suitable choice of accepting states. Recall that states in $\mathcal{W}^\dagger$ are of the form (q, ρ', k'), where q is a state from $\mathcal{W}$ and (ρ', k') is admissible. If we designate states (t, ρ', k') with $(\rho', k') \geq x$ as accepting, it will accept $\{a^n \mid d_{s,t}(n) \geq x\}$. This is a polynomial-time construction.

Then, by Lemma 26, the problem whether $d_{s,t}(n) \leq d_{s',t}(n)$ for all but finitely many $n \in \mathbb{N}$ is equivalent to $\mathcal{L}(\mathcal{N}_{s,x}) \subsetneq \mathcal{L}(\mathcal{N}_{s',x})$ for all admissible x. As there are at most $|Q|^2$ values of x and each can be verified non-deterministically in **coNP**, it suffices to show that $\mathcal{L}(\mathcal{N}_{s,x}) \subsetneq \mathcal{L}(\mathcal{N}_{s',x})$ is in **coNP** for each x. This is the case by Lemma 25. ◀

Remark 10 and Lemma 27 together complete the upper bound result for Theorem 14.

▶ Remark. Lemma 26 may appear simpler using $\{n \mid f_1(n) = x\} \subsetneq \{n \mid f_2(n) \geq x\}$. However, it does not seem possible to construct an NFA for $\{a^n \mid d_{s,t}(n) = x\}$ in polynomial time. Taking just (t, ρ, k) as accepting would not be correct, as there could be paths of the same length ending in (t, ρ', k') with $(\rho', k') > (\rho, k)$. Using $\geq$ instead of $=$ avoids this problem.

▶ Remark. An alternative approach for obtaining an upper bound could be to compute the Jordan normal form of the transition matrix and consider its powers. Instead of the interplay of strongly connected components in the transition graph, we would need to consider linear combinations of the nth powers of complex numbers (such as roots of unity). It is not clear this algebraic approach leads to a representation more convenient for our purposes.

5.3 coNP-hardness for unary LMC

Given a unary NFA $\mathcal{N}$, the *NFA universality problem* asks if $\mathcal{L}(\mathcal{N}) = \{a^n \mid n \in \mathbb{N}\}$. This problem is **coNP**-complete [37]. We exhibit a polynomial-time reduction from (a variant of) the unary universality problem to the big-O problem on unary Markov chains. Further details are available in the full version.

6 Decidability for weighted automata with bounded languages

In this section we consider the big-O problem for a weighted automaton $\mathcal{W}$ and states s, s' such that $\mathcal{L}_s(\mathcal{W})$, $\mathcal{L}_{s'}(\mathcal{W})$ are bounded. Throughout the section, we assume that the LC condition has already been checked, i.e. $\mathcal{L}_s(\mathcal{W}) \subseteq \mathcal{L}_{s'}(\mathcal{W})$. We will show that the problem is conditionally decidable, subject to Schanuel's conjecture.

Logical theories of arithmetic and Schanuel's conjecture. In *first-order logical theories of arithmetic,* variables denote numbers (from $\mathbb{Z}$ or $\mathbb{R}$, as appropriate), and atomic predicates are equalities and inequalities between terms built from variables and function symbols. Nullary function symbols are constants, always from $\mathbb{Z}$. If binary addition and multiplication are available, then:

- for $\mathbb{R}$ we obtain the first-order theory of the reals, where the truth value of sentences is decidable due to the celebrated Tarski–Seidenberg theorem [3, Chapter 11 and Theorem 2.77];
- for $\mathbb{Z}$, the first-order theory of the integers is, in contrast, undecidable (see, e.g, [32]).

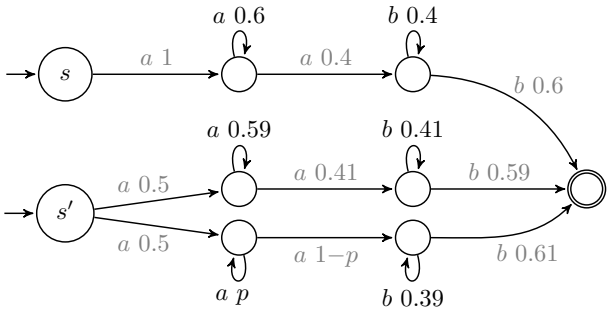


Figure 3 Relative orderings are the same, but the boundedness question is different.

In the case of $\mathbb{R}$, adding the unary symbol for the exponential function $x \mapsto e^x$, leads to *the first-order theory of the real numbers with exponential function* ($\mathrm{Th}(\mathbb{R}_{\mathsf{exp}})$). Logarithms base 2, for example, are easily expressible in $\mathrm{Th}(\mathbb{R}_{\mathsf{exp}})$. The decidability of $\mathrm{Th}(\mathbb{R}_{\mathsf{exp}})$ is an open problem and hinges upon Schanuel's conjecture [24].

Schanuel's conjecture [23] is a unifying conjecture of transcendental number theory, saying that for all $z_1, \ldots, z_n \in \mathbb{C}$ linearly independent over $\mathbb{Q}$ the field extension $\mathbb{Q}(z_1, \ldots, z_n, e^{z_1}, \ldots, e^{z_n})$ has transcendence degree at least n over $\mathbb{Q}$, meaning that for some $S \subseteq \{z_1, \ldots, z_n, e^{z_1}, \ldots, e^{z_n}\}$ of cardinality n, say $S = \{s_1, \ldots, s_n\}$, the only polynomial p over $\mathbb{Q}$ satisfying $p(s_1, \ldots, s_n) = 0$ is $p \equiv 0$. See, e.g., Waldschmidt's book [40, Section 1.4] for further context. If indeed true, this conjecture would generalise several known results, including the Lindemann–Weierstrass theorem and Baker's theorem, and would entail the decidability of $\mathrm{Th}(\mathbb{R}_{\mathsf{exp}})$. Our work follows an exciting line of research that reduces problems from verification [9, 25], linear dynamical systems [2, 7], and symbolic computation [18] to the decision problem for $\mathrm{Th}(\mathbb{R}_{\mathsf{exp}})$.

▶ **Theorem 28.** *Given a weighted automaton $\mathcal{W} = \langle Q, \Sigma, M, F \rangle$, $s, s' \in Q$, with $\mathcal{L}_s(\mathcal{W})$ and $\mathcal{L}_{s'}(\mathcal{W})$ bounded, it is decidable whether s is big-O of s', subject to Schanuel's conjecture.*

In the unary case, it was sufficient to consider the *relative order* between spectral radii, with careful handling of the periodic behaviour. This approach is insufficient in the bounded case. Example 29 highlights that the actual values of the spectral radii have to be examined.

▶ **Example 29** (Relative orderings are insufficient)**.** Consider the LMC in Figure 3, with $0.61 \leq p \leq 0.62$. We have $f_s(a^m b^n) = \Theta(0.6^m 0.4^n)$ and $f_{s'}(a^m b^n) = \Theta(p^m 0.39^n + 0.59^m 0.41^n)$. Note that neither $0.59^m 0.41^n$ nor $p^m 0.39^n$ dominate, nor are dominated by, $0.6^m 0.4^n$ for any value of $0.61 \leq p \leq 0.62$. That is, there are values of m, n where $0.59^m 0.41^n \gg 0.6^m 0.4^n$ (in particular large n) and values of m, n where $0.59^m 0.41^n \ll 0.6^m 0.4^n$ (in particular large m); similarly for $p^m 0.39^n$ vs $0.6^m 0.4^n$ (but the cases in which n or m needs to be large are swapped). However, the big-O status can be different for different values of $p \in [0.61, 0.62]$, despite the same relative ordering between spectral radii. When $p = 0.62$, the ratio turns out to be bounded: $\frac{f_s(a^m b^n)}{f_{s'}(a^m b^n)} \leq \frac{1600}{1579}$ for all m, n (in particular, maximal at $m = n = 0$). In contrast, when $p = 0.61$, we have $\frac{f_s(a^m b^{0.66m})}{f_{s'}(a^m b^{0.66m})} \xrightarrow[m \to \infty]{} \infty$.

We first prove Theorem 28 for the plus-letter-bounded case, which is the most technically involved; the other bounded cases will be reduced to it. In the plus-letter-bounded case, we will characterise the behaviour of such automata, generalising (ρ, k)-pairs of the unary case. We will need to rely upon the first-order theory of the reals with exponentials to compare these behaviours.

6.1 The plus-letter-bounded case

We assume $\mathcal{L}_{s'}(\mathcal{W}) \subseteq a_1^+ \cdots a_m^+$, where $a_1, \cdots, a_n \in \Sigma$ and because the LC condition holds, we also have $\mathcal{L}_s(\mathcal{W}) \subseteq a_1^+ \cdots a_m^+$. In the plus-letter-bounded cases, without loss of generality, we assume $a_i \neq a_j$ for $i \neq j$. Then any word $w = a_1^{n_1} \dots a_m^{n_m}$ is uniquely specified by a vector $(n_1, \dots, n_m) \in \mathbb{N}_{>0}^m$, where n_i is the number of a_i's in w.

Like in Definition 20, we define a degree function d, which will be used to study the asymptotic behaviour of words. This time we will associate a separate (ρ, k) pair to each of the m characters and, consequently, words will induce sequences of the form $(\rho_1, k_1) \cdots (\rho_m, k_m)$.

Further, as there may be multiple, incomparable behaviours, words will induce sets of such sequences, i.e. $d\colon \mathbb{N}^m \to \mathcal{P}((\mathbb{R} \times \mathbb{N})^m)$. For the sake of comparisons, it will be convenient to focus on maximal elements with respect to the pointwise order on $(\mathbb{R} \times \mathbb{N})^m$, written $\leq$, where the lexicographic order (recall Definition 17) is used to compare elements of $\mathbb{R} \times \mathbb{N}$.

Recall Lemma 21 does not capture the asymptotics when $n \leq |Q|$. In the unary case this is inconsequential as small words are covered by the *finitely many* exceptions and the LC condition. However, here, a small number of one character may be used to enable access to a particular part of the automaton in another character. For this case, we introduce a new number $\delta = \frac{1}{2} \min_{\varphi : \rho_\varphi > 0} \rho_\varphi$ which is strictly smaller than the spectral radius of every non-zero SCC (so will not dominate with the partial order), but non-zero.

▶ **Definition 30.** *Let $\hat{\rho} = (\rho_1, k_1), \cdots, (\rho_m, k_m) \in (\mathbb{R} \times \mathbb{N})^m$. An $a_1^{n_1} a_2^{n_2} \dots a_m^{n_m}$-labelled path from s (to the final state) is* compatible with $\hat{\rho}$ *if, for each $i = 1, \dots, m$, it visits $k_i + 1$ SCCs with spectral radius ρ_i while reading a_i, unless the path visits only singletons with no loops, in which case $(\rho_i, k_i) = (\delta, 0)$. The notation $(\rho, k) \in \hat{\rho}$ is used for "(ρ, k) is an element of $\hat{\rho}$".*

▶ **Definition 31.** *Let $d_s : \mathbb{N}^m \to \mathcal{P}((\mathbb{R} \times \mathbb{N})^m)$ be s.t.: $\hat{\rho} \in d_s(n_1, \dots, n_m)$ if and only if*

(1) *there exists an $a_1^{n_1} a_2^{n_2} \dots a_m^{n_m}$-labelled path from s to the final state compatible with $\hat{\rho}$, and*

(2) *for every $a_1^{n_1} a_2^{n_2} \dots a_m^{n_m}$-labelled path from s compatible with $\hat{\sigma}$ s.t. $\hat{\rho} \leq \hat{\sigma}$, we have $\hat{\rho} = \hat{\sigma}$.*

Observe that $\hat{\rho}$ may range over at most $|Q|^{2m}$ possible values. We write $\mathcal{D}$ for the set containing them, so that $d_s : \mathbb{N}^m \to \mathcal{P}(\mathcal{D})$. In this extended setting, the big-Θ lemma (Lemma 21) may be generalised as follows.

▶ **Lemma 32.** *Denote $z(n_1, \cdots, n_m) = \sum_{\hat{\rho} \in d_s(n_1, \dots, n_m)} \prod_{(\rho_i, k_i) \in \hat{\rho}} \rho_i^{n_i} \cdot n_i^{k_i}$. There exist $c, C > 0$ such that for all $n_1 \dots, n_m \in \mathbb{N}$:*

$$c \cdot z(n_1, \cdots, n_m) \leq f_s(a_1^{n_1} a_2^{n_2} \dots a_m^{n_m}) \leq C \cdot z(n_1, \cdots, n_m).$$

The following lemma provides the key characterisation of negative instances of the big-O problem, in the plus-letter-bounded case and assuming the LC condition. Here and below, we write $n(t)$ to refer to the the tth vector in a sequence $n\colon \mathbb{N} \to \mathbb{N}^m$.

▶ **Lemma 33** (Main lemma). *Assume $\mathcal{L}_s(\mathcal{W}) \subseteq \mathcal{L}_{s'}(\mathcal{W})$. Then s is not big-O of s' if and only if there exists a sequence $n\colon \mathbb{N} \to \mathbb{N}^m$ and $X \in \mathcal{D}$, $\mathcal{Y} \subseteq \mathcal{D}$ such that*

(a) *$X \in d_s(n(t))$ and $\mathcal{Y} = d_{s'}(n(t))$ for all t, and*

(b) *for all $j \in h_{\mathcal{Y}}$, the sequence n satisfies*

$$\sum_{i=1}^{m} \alpha_{j,i} n(t)_i + p_{j,i} \log n(t)_i \xrightarrow[t \to \infty]{} -\infty,$$

where $h_{\mathcal{Y}} \subseteq \{1, \dots, |\mathcal{Y}|\}$, $\alpha_{j,i} \in \mathbb{R}$, $p_{j,i} \in \mathbb{Z}$ $(1 \leq i \leq m)$ are uniquely determined by X and $\mathcal{Y}$ (in a way detailed below), $h_{\mathcal{Y}}$ and $p_{j,i}$'s are effectively computable and $\alpha_{j,i}$'s are first-order expressible (with exponential function).

Proof. Observe that then s is *not* big-O of s' iff there exists an infinite sequence of words such that, for all $C > 0$, the sequence contains a word w such that $\frac{f_s(w)}{f_{s'}(w)} > C$. Thanks to Lemma 32, this is equivalent to the existence of a sequence $n : \mathbb{N} \to \mathbb{N}^m$ such that

$$\frac{\displaystyle\sum_{X \in d_s(n(t)_1,\ldots,n(t)_m)} \prod_{(\rho_i,k_i)\in X} \rho_i^{n(t)_i} \cdot n(t)_i^{k_i}}{\displaystyle\sum_{Y \in d_{s'}(n(t)_1,\ldots,n(t)_m)} \prod_{(\sigma_i,\ell_i)\in Y} \sigma_i^{n(t)_i} \cdot n(t)_i^{\ell_i}} \xrightarrow[t\to\infty]{} \infty,$$

where $n(t)_i$ denotes the ith component of $n(t)$. Since there are finitely many possible values of d_s and $d_{s'}$, it suffices to look for sequences n such that $d_s(n(t))$ and $d_{s'}(n(t))$ are fixed. Further, because of the sum in the numerator, only one $X \in \mathcal{X}$ is required such that $X \in d_s(n_1, \ldots, n_m)$. Thus, we need to determine whether there exist $X \in \mathcal{D}$, $\mathcal{Y} \subseteq \mathcal{D}$ and $n : \mathbb{N} \to \mathbb{N}^m$ such that $X \in d_s(n(t))$, $d_{s'}(n(t)) = \mathcal{Y}$ (for all t) and

$$\frac{\prod_{i=1}^m \rho_i^{n(t)_i} \cdot n(t)_i^{k_i}}{\sum_{j=1}^{h_\mathcal{Y}} \prod_{i=1}^m \sigma_{ji}^{n(t)_i} \cdot n(t)_i^{\ell_{ji}}} \xrightarrow[t\to\infty]{} \infty.$$

where $X = (\rho_1, k_1) \cdots (\rho_m, k_m)$, $\mathcal{Y} = \{Y_1, \cdots, Y_{|\mathcal{Y}|}\}$, and $Y_j = (\sigma_{j1}, \ell_{j1}) \cdots (\sigma_{jm}, \ell_{jm})$ ($1 \leq j \leq |\mathcal{Y}|$). Taking the reciprocal and requiring each of the summands to go to zero, we obtain

$$\frac{\prod_{i=1}^m \sigma_{ji}^{n(t)_i} \cdot n(t)_i^{\ell_{ji}}}{\prod_{i=1}^m \rho_i^{n(t)_i} \cdot n(t)_i^{k_i}} = \prod_{i=1}^m \left(\frac{\sigma_{ji}}{\rho_i}\right)^{n(t)_i} n(t)_i^{\ell_{ji}-k_i} \xrightarrow[t\to\infty]{} 0 \qquad \text{for all } 1 \leq j \leq |\mathcal{Y}|.$$

If we take logarithms, letting $\alpha_{j,i} = \log(\frac{\sigma_{ji}}{\rho_i})$ and $p_{j,i} = \ell_{ji} - k_i$, we get

$$\sum_{i=1}^m \alpha_{j,i} n(t)_i + p_{j,i} \log n(t)_i \xrightarrow[t\to\infty]{} -\infty$$

for all j in $h_\mathcal{Y} = \{1 \leq j \leq |\mathcal{Y}| \mid \sigma_{ji} > 0 \text{ for all } 1 \leq i \leq m\}$.

The number $\alpha_{j,i}$ is the logarithm of the ratio of two algebraic numbers, which are not given explicitly. However, they admit an unambiguous, first-order expressible characterisation (see Remark 16). The logarithm is encoded using the exponential function: $\log(z)$ is $\exists x \in \mathbb{R} : \exp(x) = z$. ◀

Lemma 33 identifies violation of the big-O property using two conditions. In the remainder of this subsection we will handle Condition (a) using automata-theoretic tools (the Parikh theorem and semi-linear sets) and Condition (b) using logics. In summary, the characterisation of Lemma 33 will be expressed in the first-order theory of the reals with exponentiation, which is decidable subject to Schanuel's conjecture.

Condition (a) via automata

It turns out that sequences n satisfying Condition (a) in Lemma 33 can be captured by a finite automaton. In more detail, for any $X \in \mathcal{D}$, there exists an automaton $\mathcal{N}_X^s$ such that $\mathcal{L}(\mathcal{N}_X^s) = \{a_1^{n_1} \cdots a_m^{n_m} \mid X \in d_s(n_1, \cdots, n_m)\}$. For any $\mathcal{Y} \subseteq \mathcal{D}$, there exists an automaton $\mathcal{N}_\mathcal{Y}^s$ such that $\mathcal{L}(\mathcal{N}_\mathcal{Y}^s) = \{a_1^{n_1} \cdots a_m^{n_m} \mid d_s(n_1, \cdots, n_m) = \mathcal{Y}\}$. The relevant automaton capturing X and $\mathcal{Y}$ is then found by taking the intersection of $\mathcal{L}(\mathcal{N}_X^s)$ and $\mathcal{L}(\mathcal{N}_\mathcal{Y}^{s'})$.

▶ **Lemma 34.** *For any $X \in \mathcal{D}$ and $\mathcal{Y} \subseteq \mathcal{D}$, there exists an automaton $\mathcal{N}_{X,\mathcal{Y}}$ such that $\mathcal{L}(\mathcal{N}_{X,\mathcal{Y}}) = \{a_1^{n_1} \cdots a_m^{n_m} \mid X \in d_s(n_1, \cdots, n_m),\ \mathcal{Y} = d_{s'}(n_1, \cdots, n_m)\}$.*

Because of our $a_i \neq a_j$ assumption, the vector $(n_1, \cdots, n_m)$ indicates the number of occurrences of each character. The set of such vectors derived from the language of an automaton is known as the Parikh image of this language [30]. It is well known that the Parikh image of an NFA is a semi-linear set, i.e. a finite union of linear sets (a linear set has the form $\{\vec{b} + \lambda_1 \vec{r^1} + \cdots + \lambda_s \vec{r^s} \mid \lambda_1, \ldots, \lambda_s \in \mathbb{N}\}$, where $\vec{b} \in \mathbb{N}^m$ is the base vector and $\vec{r^1}, \cdots \vec{r^s} \in \mathbb{N}^m$ are called period vectors). However, since $\mathcal{L}(\mathcal{N}_{X,\mathcal{Y}}) \subseteq a_1^+ a_2^+ \ldots a_m^+$, the linear sets are of a very particular form, where each $\vec{r^i}$ is a constant multiple of the ith unit vector.

▶ **Lemma 35.** *The language of $\mathcal{N}_{X,\mathcal{Y}}$ can be effectively decomposed as $\mathcal{L}(\mathcal{N}_{X,\mathcal{Y}}) = \bigcup_{k=1}^{S_{X,\mathcal{Y}}} \mathcal{L}_k$, where $\mathcal{L}_k = \left\{ a_1^{b_{k1}+r_{k1}\lambda_1} \cdots a_m^{b_{km}+r_{km}\lambda_m} \mid \lambda_1, \cdots, \lambda_m \in \mathbb{N} \right\}$, $S_{X,\mathcal{Y}} \in \mathbb{N}$ and $b_{ki}, r_{ki} \in \mathbb{N}$ $(1 \leq k \leq S_{X,\mathcal{Y}},\ 1 \leq i \leq m)$.*

Lemma 35 captures Condition (a) of Lemma 33 precisely.

Condition (b) via logic

With Lemma 35 in place, we now move on to add Condition (b) to the existing machinery. In fact, the logical formulae in the following lemmas will express the conjunction of both conditions of Lemma 33.

▶ **Lemma 36.** *Assume $\mathcal{L}_s(\mathcal{W}) \subseteq \mathcal{L}_{s'}(\mathcal{W})$. Then s is not big-O of s' if and only if there exists $X \in \mathcal{D}$, $\mathcal{Y} \subseteq \mathcal{D}$, $1 \leq k \leq S_{X,\mathcal{Y}}$ such that*

$$\forall C < 0\ \exists \vec{\lambda} \in \mathbb{N}^m \qquad \bigwedge_{j \in h_{\mathcal{Y}}} \sum_{i=1}^{m} \alpha_{j,i}(b_{ki} + r_{ki}\lambda_i) + p_{j,i} \log(b_{ki} + r_{ki}\lambda_i) < C,$$

where $h_{\mathcal{Y}}, \alpha_{j,i}, p_{j,i}$ (resp. b_{ki}, r_{ki}) satisfy the same conditions as in Lemma 33 (resp. 35).

Note that the formula of Lemma 36 uses quantification over natural numbers. Our next step will be to replace integer variables with real variables. In other words, we will obtain an equivalent condition in the first-order theory of the reals with exponentiation, as follows.

▶ **Lemma 37.** *Assume $\mathcal{L}_s(\mathcal{W}) \subseteq \mathcal{L}_{s'}(\mathcal{W})$. Then s is not big-O of s' if and only if there exist $X \in \mathcal{D}$, $\mathcal{Y} \subseteq \mathcal{D}$, $1 \leq k \leq S_{X,\mathcal{Y}}$ and $U \subseteq \{i \in \{1, \cdots, m\} \mid r_{ki} > 0\}$ such that*

$$\forall C < 0\ \exists \vec{x} \in \mathbb{R}^{|U|}_{\geq B_k} \qquad \bigwedge_{j \in h_{\mathcal{Y}}} \sum_{i \in U} \alpha_{j,i} r_{ki} x_i + p_{j,i} \log(x_i) < C,$$

where $B_k = \max_i b_{ki}$ and $h_{\mathcal{Y}}, \alpha_{j,i}, p_{j,i}, b_{ki}, r_{ki}$ are as in Lemma 36.

Proof Sketch. Compare the logical characterisation in Lemmas 36 and 37. The first difference to note is that the effect of b_{ki}'s is simply a constant offset, and so the sequence would tend to $-\infty$ with or without its presence. Similar simplifications can be made inside the logarithm: the multiplicative effect of r_{ki} inside the logarithm can be extracted as an additive offset and thus similarly be discarded.

The second crucial difference is to relax the variable domains from integers to reals. If each of the λ_i in the satisfying assignment is sufficiently large, we show we can relax the condition to real numbers rather than integers without affecting whether the sequence goes to $-\infty$. To do this, we test sets of indices U, where if $i \in U$ then λ_i needs to be arbitrarily large over all C (i.e. unbounded). The positions where λ_i is always bounded are again a constant offset and are omitted. ◀

By testing the LC condition and the condition from Lemma 37 for each possible $X, \mathcal{Y}, k, U$, in turn using the relevant (conditionally decidable) first-order theory of the reals, we have:

▶ **Lemma 38.** *Given a weighted automaton $\mathcal{W}$ and states s, s' such that $\mathcal{L}_s(\mathcal{W})$ and $\mathcal{L}_{s'}(\mathcal{W})$ are plus-letter-bounded, it is decidable whether s is big-O s', subject to Schanuel's conjecture.*

6.2 The letter-bounded case

Here we consider the case where $\mathcal{L}_s(\mathcal{W})$ and $\mathcal{L}_{s'}(\mathcal{W})$ are letter-bounded, $\mathcal{L}_s(\mathcal{W})$ and $\mathcal{L}_{s'}(\mathcal{W})$ are subsets of $a_1^* \dots a_m^*$ for some $a_1, \dots, a_m \in \Sigma$, which is a relaxation of the preceding case. For the plus-letter-bounded case, we relied on a 1-1 correspondence between numeric vectors and words. This correspondence no longer holds in the letter-bounded case: for example, a^n matches $a^*b^*a^*$, but it could correspond to $(n, 0, 0)$, $(0, 0, n)$, as well as any $(n_1, 0, n_2)$ with $n_1 + n_2 = n$. Still, there is a reduction to the plus-letter-bounded case.

▶ **Lemma 39.** *The big-O problem for $\mathcal{W}, s, s'$ with $\mathcal{L}_s(\mathcal{W})$ and $\mathcal{L}_{s'}(\mathcal{W})$ letter-bounded reduces to the plus-letter-bounded case.*

Proof. Suppose the LC condition holds and $\mathcal{L}_s(\mathcal{W}) \subseteq \mathcal{L}_{s'}(\mathcal{W}) \subseteq a_1^* \cdots a_m^*$. Let I be the set of strictly increasing sequences $\vec{\imath} = i_1 \cdots i_k$ of integers between 1 and m. Given $\vec{\imath} \in I$, let $\mathcal{W}_{\vec{\imath}}$ be the weighted automaton obtained by intersecting $\mathcal{W}$ with a DFA for $a_{i_1}^+ \cdots a_{i_k}^+$ whose initial state is q. Note that s is big-O of s' (in $\mathcal{W}$) iff (s, q) is big-O of (s', q) in $\mathcal{W}_{\vec{\imath}}$ for all $\vec{\imath} \in I$, because $a_1^* \cdots a_m^* = \bigcup_{\vec{\imath} \in I} a_{i_1}^+ \cdots a_{i_k}^+$. Because the big-O problem for each $\mathcal{W}_{\vec{\imath}}$, (s, q), (s', q) falls into the plus-letter-bounded case, the results follows from Lemma 38. ◀

6.3 The bounded case

Here we consider the case where $\mathcal{L}_s(\mathcal{W})$ and $\mathcal{L}_{s'}(\mathcal{W})$ are bounded, which is a relaxation of letter-boundedness (see Definition 6): $\mathcal{L}_s(\mathcal{W})$ and $\mathcal{L}_{s'}(\mathcal{W})$ are subsets of $w_1^* \dots w_m^*$ for some $w_1, \dots, w_m \in \Sigma^*$. We show a reduction to the letter-bounded case from Section 6.2.

To showcase the difference to the letter-bounded case, consider the language $(abab)^*a^*b^*(ab)^*$. Observe that, for example the word $(ab)^4$ can be decomposed in a number of ways: $(abab)^2a^0b^0(ab)^0$, $(abab)^1a^1b^1(ab)^1$, $(abab)^1a^0b^0(ab)^2$, $(abab)^0a^1b^1(ab)^3$ or $(abab)^0a^0b^0(ab)^4$. One must be careful to consider all such decompositions.

▶ **Lemma 40.** *The big-O problem for $\mathcal{W}, s, s'$ with $\mathcal{L}_s(\mathcal{W})$ and $\mathcal{L}_{s'}(\mathcal{W})$ bounded reduces to the letter-bounded case.*

Proof sketch. Suppose $\mathcal{W}$ is bounded over $w_1^* \dots w_m^*$, we will construct a new weighted automaton $\mathcal{W}'$ letter-bounded over a new alphabet $a_1^* \dots a_m^*$ with the following property. For *every* decomposition of a word w, as $w_1^{n_1} \dots w_m^{n_m}$, the weight of $a_1^{n_1} \dots a_m^{n_m}$ in $\mathcal{W}'$ is equal to the weight of w in $\mathcal{W}$. ◀

7 Conclusion

Despite undecidability results, we have identified several decidable cases of the big-O problem. However, for bounded languages, the result depends on a conjecture from number theory, leaving open the exact borderline between decidability and undecidability.

Natural directions for future work include the analogous problem for infinite words, further analysis on ambiguity (e.g., is the big-O problem decidable for k-ambiguous weighted automata?), and the extension to negative edge weights.

References

1 Shaull Almagor, Udi Boker, and Orna Kupferman. What's decidable about weighted automata? In *ATVA*, volume 6996 of *Lecture Notes in Computer Science*, pages 482–491. Springer, 2011.

2 Shaull Almagor, Dmitry Chistikov, Joël Ouaknine, and James Worrell. O-minimal invariants for linear loops. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 114:1–114:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.114.

3 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and computation in mathematics*. Springer, 2nd edition, 2006.

4 Konstantinos Chatzikokolakis, Daniel Gebler, Catuscia Palamidessi, and Lili Xu. Generalized Bisimulation Metrics. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014*, volume 8704 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2014. doi:10.1007/978-3-662-44584-6_4.

5 Taolue Chen and Stefan Kiefer. On the total variation distance of labelled Markov chains. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS 2014*, pages 33:1–33:10. ACM, 2014. doi:10.1145/2603088.2603099.

6 Dmitry Chistikov, Andrzej S. Murawski, and David Purser. Asymmetric distances for approximate differential privacy. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019*, volume 140 of *LIPIcs*, pages 10:1–10:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.10.

7 Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the Skolem problem for continuous linear dynamical systems. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 100:1–100:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.100.

8 Thomas Colcombet. Unambiguity in automata theory. In Jeffrey O. Shallit and Alexander Okhotin, editors, *Descriptional Complexity of Formal Systems - 17th International Workshop, DCFS 2015*, volume 9118 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2015. doi:10.1007/978-3-319-19225-3_1.

9 Laure Daviaud, Marcin Jurdzinski, Ranko Lazic, Filip Mazowiecki, Guillermo A. Pérez, and James Worrell. When is containment decidable for probabilistic automata? In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 121:1–121:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.121.

10 C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk. A Storm is coming: A modern probabilistic model checker. In *Proceedings of Computer Aided Verification (CAV)*, pages 592–600. Springer, 2017.

11 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2006. doi:10.1007/11681878_14.

12 Nathanaël Fijalkow. Undecidability results for probabilistic automata. *SIGLOG News*, 4(4):10–17, 2017. URL: https://dl.acm.org/citation.cfm?id=3157833.

13 Shmuel Friedland and Hans Schneider. The growth of powers of a nonnegative matrix. *SIAM J. Matrix Analysis Applications*, 1(2):185–200, 1980. doi:10.1137/0601022.

14 Pawel Gawrychowski, Dalia Krieger, Narad Rampersad, and Jeffrey Shallit. Finding the growth rate of a regular or context-free language in polynomial time. *Int. J. Found. Comput. Sci.*, 21(4):597–618, 2010. doi:10.1142/S0129054110007441.

15 Hugo Gimbert and Youssouf Oualhadj. Probabilistic automata on finite words: Decidable and undecidable problems. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part II*, volume 6199 of *Lecture Notes in Computer Science*, pages 527–538. Springer, 2010. doi:10.1007/978-3-642-14162-1_44.

16 Seymour Ginsburg. *The Mathematical Theory of Context-Free Languages*. McGraw-Hill, 1966.

17 Seymour Ginsburg and Edwin H Spanier. Bounded algol-like languages. *Transactions of the American Mathematical Society*, 113(2):333–368, 1964.

18 Cheng-Chao Huang, Jing-Cao Li, Ming Xu, and Zhi-Bin Li. Positive root isolation for poly-powers by exclusion and differentiation. *Journal of Symbolic Computation*, 85:148–169, 2018. 41th International Symposium on Symbolic and Alge-braic Computation (ISSAC'16). doi:10.1016/j.jsc.2017.07.007.

19 Stefan Kiefer. On computing the total variation distance of hidden Markov models. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018*, volume 107 of *LIPIcs*, pages 130:1–130:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.130.

20 Stefan Kiefer, Andrzej S. Murawski, Joël Ouaknine, Björn Wachter, and James Worrell. On the Complexity of Equivalence and Minimisation for Q-weighted Automata. *Logical Methods in Computer Science*, 9(1), 2013. doi:10.2168/LMCS-9(1:8)2013.

21 Daniel Krob. The equality problem for rational series with multiplicities in the tropical semiring is undecidable. *International Journal of Algebra and Computation*, 4:405–425, 1994.

22 M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Proceedings of Computer Aided Verification (CAV)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

23 Serge Lang. *Introduction to transcendental numbers*. Addison-Wesley Pub. Co., 1966.

24 Angus Macintyre and Alex J Wilkie. On the decidability of the real exponential field, 1996.

25 Rupak Majumdar, Mahmoud Salamati, and Sadegh Soudjani. On decidability of time-bounded reachability in CTMDPs. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 133:1–133:19, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.133.

26 David Mestel. Quantifying information flow in interactive systems. In *32nd IEEE Computer Security Foundations Symposium, CSF 2019, Hoboken, NJ, USA, June 25-28, 2019*, pages 414–427. IEEE, 2019. doi:10.1109/CSF.2019.00035.

27 David Mestel. Widths of Regular and Context-Free Languages. In *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019)*, volume 150 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 49:1–49:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2019.49.

28 Albert R. Meyer and Larry J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th Annual Symposium on Switching and Automata Theory, College Park, Maryland, USA, October 25-27, 1972*, pages 125–129. IEEE Computer Society, 1972.

29 Joël Ouaknine and James Worrell. Positivity problems for low-order linear recurrence sequences. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 366–379. SIAM, 2014. doi:10.1137/1.9781611973402.27.

30 Rohit J Parikh. On context-free languages. *Journal of the ACM (JACM)*, 13(4):570–581, 1966.

31 Azaria Paz. *Introduction to probabilistic automata*. Academic Press, 2014.

32 Bjorn Poonen. Hilbert's tenth problem over rings of number-theoretic interest. *Note from the lecture at the Arizona Winter School on "Number Theory and Logic"*, 2003. URL: https://math.mit.edu/~poonen/papers/aws2003.pdf.

33 Hans Schneider. The influence of the marked reduced graph of a nonnegative matrix on the Jordan form and on related properties: A survey. *Linear Algebra and its Applications*, 84:161–189, 1986.

34 Marcel Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961. doi:10.1016/S0019-9958(61)80020-X.

35 Bruno Sericola. *Markov chains: theory and applications*. John Wiley & Sons, 2013.

36 Adam D. Smith. Efficient, Differentially Private Point Estimators. *CoRR*, abs/0809.4794, 2008. arXiv:0809.4794.

37 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, 1973*, pages 1–9. ACM, 1973. doi:10.1145/800125.804029.

38 Wen-Guey Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM J. Comput.*, 21(2):216–227, 1992. doi:10.1137/0221017.

39 Wen-Guey Tzeng. On path equivalence of nondeterministic finite automata. *Inf. Process. Lett.*, 58(1):43–46, 1996. doi:10.1016/0020-0190(96)00039-7.

40 Michel Waldschmidt. *Diophantine Approximation on Linear Algebraic Groups*, volume 326 of *Grundlehren der mathematischen Wissenschaften (A Series of Comprehensive Studies in Mathematics)*. Springer, Berlin, Heidelberg, 2000.

Determinisability of One-Clock Timed Automata

Lorenzo Clemente
University of Warsaw, Poland
clementelorenzo@gmail.com

Sławomir Lasota
University of Warsaw, Poland
sl@mimuw.edu.pl

Radosław Piórkowski
University of Warsaw, Poland
r.piorkowski@mimuw.edu.pl

Abstract

The deterministic membership problem for timed automata asks whether the timed language recognised by a nondeterministic timed automaton can be recognised by a deterministic timed automaton. We show that the problem is decidable when the input automaton is a one-clock nondeterministic timed automaton without epsilon transitions and the number of clocks of the deterministic timed automaton is fixed. We show that the problem in all the other cases is undecidable, i.e., when either 1) the input nondeterministic timed automaton has two clocks or more, or 2) it uses epsilon transitions, or 3) the number of clocks of the output deterministic automaton is not fixed.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Timed and hybrid models

Keywords and phrases Timed automata, determinisation, deterministic membership problem

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.42

Related Version A full version of the paper is available as an arXiv technical report [17] at `https://arxiv.org/abs/2007.09340`.

Funding *Lorenzo Clemente*: Partially supported by the Polish NCN grant 2017/26/D/ST6/00201.
Sławomir Lasota: Partially supported by the Polish NCN grant 2019/35/B/ST6/02322 and by the ERC grant LIPA, agreement no. 683080.
Radosław Piórkowski: Partially supported by the Polish NCN grant 2017/27/B/ST6/02093.

Acknowledgements We thank S. Krishna for fruitful discussions and the anonymous reviewers for their constructive comments.

1 Introduction

Nondeterministic timed automata (NTA) are one of the most widespread model of real-time reactive systems. They are an extension of finite automata with real-valued clocks which can be reset and compared by inequality constraints. The nonemptiness problem for NTA is decidable and in fact PSPACE-complete, as shown by Alur and Dill in their landmark paper [2]. As a testimony to the importance of the model, the authors received the 2016 Church Award for the invention of timed automata. This paved the way to the automatic verification of timed systems, leading to mature tools such as UPPAAL [8], UPPAAL Tiga (timed games) [14], and PRISM (probabilistic timed automata) [32]. The reachability problem is still a very active research area to these days [21, 29, 1, 25, 26, 28], as well as expressive generalisations thereof, such as the binary reachability problem [19, 20, 31, 23].

Deterministic timed automata (DTA) form a strict subclass of NTA where the next configuration is uniquely determined from the current one and the timed input symbol. The class of DTA enjoys stronger properties than NTA, such as decidable universality and inclusion problems and closure under complementation [2]. Moreover, the more restrictive nature of DTA is necessary in several applications of timed automata, such as test generation [36], fault diagnosis [11], and learning [45, 41], winning conditions in timed games [4, 30, 12], and in a notion of recognisability of timed languages [34]. For these reasons, and for the more general quest of understanding the nature of the expressive power of nondeterminism in timed automata, many researchers have focused on defining determinisable classes of timed automata, such as strongly non-zeno NTA [5], event-clock NTA [3], and NTA with integer-resets [40]. The classes above are not exhaustive, in the sense that there are NTA recognising deterministic timed languages not falling into any of the classes above.

Another remarkable subclass of NTA is obtained by requiring the presence of just one clock (without epsilon transitions). The resulting class of NTA_1 is incomparable with DTA: For instance, NTA_1 are not closed under complement (unlike DTA) and there are very simple DTA languages which are not recognisable by any NTA_1. Nonetheless, NTA_1, like DTA, have decidable inclusion, equivalence, and universality problems [37, 33], albeit the complexity is non-primitive recursive [33, Corollary 4.2] (see also [38, Theorem 7.2] for an analogous lower bound for the satisfiability problem of metric temporal logic). Moreover, the non-emptiness problem for NTA_1 is NLogSpace-complete (vs. PSpace-complete for unrestricted NTA and DTA, already with two clocks [21]), and computing the binary reachability relation is simpler when there is only one clock than in the general case [16].

The deterministic membership problem. The DTA *membership problem* asks, given an NTA, whether there exists a DTA recognising the same language. There are two natural variants of this problem, which are obtained by restricting the resources available to the sought DTA. Let $k \in \mathbb{N}$ be a bound on the number of clocks, and let $m \in \mathbb{N}$ be a bound on the maximal absolute value of numerical constants. The DTA_k and $\mathsf{DTA}_{k,m}$ *membership problems* are the restriction of the problem above where the DTA is required to have at most k clocks, resp., at most k clocks and absolute value of maximal constant bounded by m. Notice that we do not bound the number of control locations of the DTA, which makes the problem non-trivial.

Since regular languages are deterministic, the DTA_k membership problem can be seen as a quantitative generalisation of the regularity problem. For instance, the DTA_0 membership problem is exactly the regularity problem since a timed automaton with no clocks is the same as a finite automaton. We remark that the regularity problem is usually undecidable for nondeterministic models of computation generalising finite automata, e.g., context-free grammars/pushdown automata [39, Theorem 6.6.6], labelled Petri nets under reachability semantics [44], Parikh automata [13], etc. One way to obtain decidability is to either restrict the input model to be deterministic (e.g., [43, 44, 7]), or to consider finer notions of equivalence, such as bisimulation (e.g., [27]).

This negative situation is generally confirmed for timed automata. For every number of clocks $k \in \mathbb{N}$ and maximal constant m, the DTA, DTA_k, and $\mathsf{DTA}_{k,m}$ membership problems are known to be undecidable when the input NTA has ≥ 2 clocks, and for 1-clock NTA with epsilon transitions [22, 42]. To the best of our knowledge, the deterministic membership problem was not studied before when the input automaton is NTA_1 without epsilon transitions.

Contributions. We complete the study of the decidability border for the deterministic membership problem initiated in [22, 42]. Our main result is the following.

▶ **Theorem 1.1.** *The* DTA_k *membership and the* $\mathsf{DTA}_{k,m}$ *membership problems are decidable for* NTA_1 *languages.*

Our decidability result contrasts starkly with the abdundance of undecidability results for the regularity problem. We establish decidability by showing that if a $\mathsf{NTA}_{k,m}$ recognises a DTA_k language, then in fact it recognises a $\mathsf{DTA}_{k,m}$ language and moreover there is a computable bound on the number of control locations of the deterministic acceptor (cf. Lemma 4.1). This provides a decision procedure since there are finitely many DTA once the number of clocks, the maximal constant, and the number of control locations are fixed.

In our technical analysis we find it convenient to introduce the so called *always resetting* subclass of NTA_k. These automata are required to reset at least one clock at every transition and are thus of expressive power intermediate between NTA_{k-1} and NTA_k. Always resetting NTA_2 are strictly more expressive than NTA_1: For instance, the language of timed words of the form $(a,t_0)(a,t_1)(a,t_2)$ s.t. $t_2 - t_0 > 2$ and $t_2 - t_1 < 1$ can be recognised by an always resetting NTA_2 but by no NTA_1. Despite their increased expressive power, always resetting NTA_2 still have a decidable universality problem (the well-quasi order approach of [37] goes through), which is not the case for NTA_2. Thanks to this restricted form, we are able to provide in Lemma 4.1 an elegant characterisation of those NTA_1 languages which are recognised by an always resetting DTA_k.

We complement the decidability result above by showing that the problem becomes undecidable if we do not restrict the number of clocks of the DTA.

▶ **Theorem 1.2.** *The* DTA *and* $\mathsf{DTA}_{\_,m}$ *(*$m > 0$*) membership problems are undecidable for* NTA_1*.*

Finally, by refining the analysis of [22], we show that the DTA_k and $\mathsf{DTA}_{k,m}$ membership problems for NTA_1 are non-primitive recursive.

▶ **Theorem 1.3.** *The* DTA_k *and* $\mathsf{DTA}_{k,m}$ *membership problems are* HyperAckermann*-hard for* NTA_1 *and undecidable for* NTA_1 *with epsilon transitions.*

Related research. Many works addressed the construction of a DTA equivalent to a given NTA (see [9] and references therein), however since the general problem is undecidable, one has to either sacrifice termination, or consider deterministic under/over-approximations. In a related line of work, we have shown that the *deterministic separability problem* is decidable for the full class of NTA, when the number of clocks of the separator is given in the input [18]. This contrasts with undecidability of the corresponding membership problem. Decidability of the deterministic separability problem when the number of clocks of the separator is not provided remains a challenging open problem.

2 Preliminaries

Timed words and languages. Fix a finite alphabet Σ. Let $\mathbb{R}$ and $\mathbb{R}_{\geq 0}$ denote reals and nonnegative reals[1], respectively. A *timed word* over Σ is any sequence of the form

$$w \;=\; (a_1,t_1)\ldots(a_n,t_n) \;\in\; (\Sigma\times\mathbb{R}_{\geq 0})^* \tag{1}$$

[1] Equivalently, nonnegative rationals may be considered in place of reals.

which is *monotonic*, in the sense that the timestamps t_i's satisfy $0 \leq t_1 \leq t_2 \leq \cdots \leq t_n$. Let $\mathbb{T}(\Sigma)$ be the set of all timed words over Σ, and let $\mathbb{T}_{\geq t}(\Sigma)$ be, for $t \in \mathbb{R}_{\geq 0}$, the set of timed words with $t_1 \geq t$. A *timed language* is a subset of $\mathbb{T}(\Sigma)$.

The concatenation $w \cdot v$ of two timed words w and v is defined only when the first time-stamp of v is greater or equal than the last timestamp of w. Using this partial operation, we define, for a timed word $w \in \mathbb{T}(\Sigma)$ and a timed language $L \subseteq \mathbb{T}(\Sigma)$, the left quotient $w^{-1}L := \{v \in \mathbb{T}(\Sigma) \mid w \cdot v \in L\}$. Clearly $w^{-1}L \subseteq \mathbb{T}_{\geq t_n}(\Sigma)$.

Clock constraints and regions. Let $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$ be a finite set of clocks. A *clock valuation* is a function $\mu \in \mathbb{R}_{\geq 0}^{\mathtt{X}}$ assigning a non-negative real number $\mu(\mathtt{x})$ to every clock $\mathtt{x} \in \mathtt{X}$. A *clock constraint* is a quantifier-free formula of the form

$$\varphi, \psi \ ::\equiv\ \mathbf{true} \mid \mathbf{false} \mid \mathtt{x}_i - \mathtt{x}_j \sim z \mid \mathtt{x}_i \sim z \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi,$$

where "$\sim$" is a comparison operator in $\{=, <, \leq, >, \geq\}$ and $z \in \mathbb{Z}$. A clock valuation μ satisfies a constraint φ, written $\mu \models \varphi$, if interpreting each clock $\mathtt{x}_i$ by $\mu(\mathtt{x}_i)$ makes φ a tautology. An k, m-*region* is a non-empty set of valuations $[\![\varphi]\!]$ satisfied by a constraint φ with k clocks and absolute value of maximal constant bounded by m, which is minimal w.r.t. set inclusion. For instance, the clock constraint $1 < \mathtt{x}_1 < 2 \wedge 4 < \mathtt{x}_2 < 5 \wedge \mathtt{x}_2 - \mathtt{x}_1 < 3$ defines a $2, 5$-region consisting of an open triangle with nodes $(1, 4)$, $(2, 4)$ and $(2, 5)$.

Timed automata. A (nondeterministic) *timed automaton* is a tuple $A = (\Sigma, \mathtt{L}, \mathtt{X}, \mathtt{I}, \mathtt{F}, \Delta)$, where Σ is a finite input alphabet, $\mathtt{L}$ is a finite set of control locations, $\mathtt{X}$ is a finite set of clocks, $\mathtt{I}, \mathtt{F} \subseteq \mathtt{L}$ are the subsets of initial, resp., final, control locations, and Δ is a finite set of transition rules of the form

$$(p, a, \varphi, \mathtt{Y}, q) \tag{2}$$

with $p, q \in \mathtt{L}$ control locations, $a \in \Sigma$, φ a clock constraint to be tested, and $\mathtt{Y} \subseteq \mathtt{X}$ the set of clocks to be reset. We write NTA for the class of all nondeterministic timed automata, NTA_k when the number k of clocks is fixed, $\mathsf{NTA}_{\_,m}$ when the bound m on constants is fixed, and $\mathsf{NTA}_{k,m}$ when both k and m are fixed.

An $\mathsf{NTA}_{\_,m}$ A is *always resetting* if every transition rule as in (2) resets some clock $\mathtt{Y} \neq \emptyset$, and *greedily resetting* if, for every clock $\mathtt{x}$, whenever φ implies that $\mathtt{x}$ belongs to $\{0, \ldots, m\} \cup (m, \infty)$, then $\mathtt{x} \in \mathtt{Y}$.

Reset-point semantics. A *configuration* of an NTA A is a tuple (p, μ, t_0) consisting of a control location $p \in \mathtt{L}$, a reset-point assignment $\mu \in \mathbb{R}_{\geq 0}^{\mathtt{X}}$, and a "now" timestamp $t_0 \in \mathbb{R}_{\geq 0}$ satisfying $\mu(\mathtt{x}) \leq t_0$ for all clocks $\mathtt{x} \in \mathtt{X}$. Intuitively, t_0 is the last timestamp seen in the input and, for every clock $\mathtt{x}$, $\mu(\mathtt{x})$ stores the timestamp of the last reset of $\mathtt{x}$. A configuration is *initial* if p is so, $t_0 = 0$, and $\mu(\mathtt{x}) = 0$ for all clocks $\mathtt{x}$, and it is *final* if p is so (without any further restriction on μ or t_0). For a set of clocks $\mathtt{Y} \subseteq \mathtt{X}$ and a timestamp $u \in \mathbb{R}_{\geq 0}$, let $\mu[\mathtt{Y} \mapsto u]$ be the assignment which is u on $\mathtt{Y}$ and agrees with μ on $\mathtt{X} \setminus \mathtt{Y}$. An assignment μ together with t_0 induces a clock valuation $t_0 - \mu$ defined as $(t_0 - \mu)(\mathtt{x}) = t_0 - \mu(\mathtt{x})$ for all clocks $\mathtt{x} \in \mathtt{X}$. Clock assignments and valuations have the same type $\mathbb{R}_{\geq 0}^{\mathtt{X}}$, however we find it technically convenient to store assignments in configurations and use the derived valuations to interpret the clock constraints. Such reset-point semantics based on reset-point assignments has already appeared in the literature on timed automata [24] and it is the foundation of the related model of timed-register automata [10].

Every transition rule (2) induces a *transition* between configurations $(p, \mu, t_0) \xrightarrow{a,t} (q, \nu, t)$ labelled by $(a, t) \in \Sigma \times \mathbb{R}_{\geq 0}$ whenever $t \geq t_0$, $t - \mu \models \varphi$, and $\nu = \mu[\mathtt{Y} \mapsto t]$. The *timed transition system* induced by A is $(\llbracket A \rrbracket, \rightarrow, F)$, where $\llbracket A \rrbracket$ is the set of configurations, $\rightarrow \subseteq \llbracket A \rrbracket \times \Sigma \times \mathbb{R}_{\geq 0} \times \llbracket A \rrbracket$ is as defined above, and $F \subseteq \llbracket A \rrbracket$ is the set of final configurations. Since there is no danger of confusion, we use $\llbracket A \rrbracket$ to denote either the timed transition system above, or its domain. A *run* of A *over* a timed word w as in (1) *starting* in configuration (p, μ, t_0) and *ending* in configuration (q, ν, t_n) is a path ρ in $\llbracket A \rrbracket$ of the form $\rho = (p, \mu, t_0) \xrightarrow{a_1,t_1} \ldots \xrightarrow{a_n,t_n} (q, \nu, t_n)$. The run ρ is accepting if its last configuration satisfies $(q, \nu, t_n) \in F$. The language *recognised* by configuration (p, μ, t_0) is defined as:

$$L_{\llbracket A \rrbracket}(p, \mu, t_0) = \{w \in \mathbb{T}(\Sigma) \mid \llbracket A \rrbracket \text{ has an accepting run over } w \text{ starting in } (p, \mu, t_0)\}.$$

Clearly $L_{\llbracket A \rrbracket}(p, \mu, t_0) \subseteq \mathbb{T}_{\geq t_0}(\Sigma)$. We write $L_A(c)$ instead of $L_{\llbracket A \rrbracket}(c)$. The language recognised by the automaton A is $L(A) = \bigcup_{c \text{ initial}} L_A(c)$. A configuration is *reachable* if it is the ending configuration in a run starting in an initial configuration. In an always resetting $\mathsf{NTA}_{\_,m}$, every reachable configuration (p, μ, t_0) satisfies $t_0 \in \mu(\mathtt{X})$, and in a greedily resetting one, 1) (p, μ, t_0) has *m-bounded span*, in the sense that $\mu(\mathtt{X}) \subseteq (t_0 - m, t_0]$, and moreover 2) any two clocks $\mathtt{x}, \mathtt{y}$ with integer difference $\mu(\mathtt{x}) - \mu(\mathtt{y}) \in \mathbb{Z}$ are actually equal $\mu(\mathtt{x}) = \mu(\mathtt{y})$. Condition 2) follows from the fact that if $\mathtt{x}, \mathtt{y}$ have integer difference and $\mathtt{y}$ was reset last, then $\mathtt{x}$ was itself an integer when this happened, and in fact they were both reset together in a greedily resetting automaton.

Deterministic timed automata. A timed automaton A is *deterministic* if it has exactly one initial location and, for every two rules $(p, a, \varphi, \mathtt{Y}, q), (p, a', \varphi', \mathtt{Y}', q') \in \Delta$, if $a = a'$ and $\llbracket \varphi \wedge \varphi' \rrbracket \neq \emptyset$ then $\mathtt{Y} = \mathtt{Y}'$ and $q = q'$. Hence A has at most one run over every timed word w. A DTA can be easily transformed to a *total* one, where for every location $p \in \mathtt{L}$ and $a \in \Sigma$, the sets defined by clock constraints $\{\llbracket \varphi \rrbracket \mid \exists \mathtt{Y}, q \cdot (p, a, \varphi, \mathtt{Y}, q) \in \Delta\}$ are a partition of $\mathbb{R}^{\mathtt{X}}_{\geq 0}$. Thus, a total DTA has exactly one run over every timed word w. We write DTA for the class of deterministic timed automata, and DTA_k, $\mathsf{DTA}_{\_,m}$, and $\mathsf{DTA}_{k,m}$ for the respective subclasses thereof. A timed language is called NTA language, DTA language, etc., if it is recognised by a timed automaton of the respective type.

▶ **Example 2.1.** Let $\Sigma = \{a\}$ be a unary alphabet. As an example of a timed language L recognised by a NTA_1, but not by any DTA, consider the set of non-negative timed words of the form $(a, t_1) \cdots (a, t_n)$ where $t_n - t_i = 1$ for some $1 \leq i < n$. The language L is recognised by the NTA_1 $A = (\Sigma, \mathtt{L}, \mathtt{X}, \mathtt{I}, \mathtt{F}, \Delta)$ with a single clock $\mathtt{X} = \{\mathtt{x}\}$ and three locations $\mathtt{L} = \{p, q, r\}$, of which $\mathtt{I} = \{p\}$ is initial and $\mathtt{F} = \{r\}$ is final, and transition rules

$$(p, a, \mathbf{true}, \emptyset, p) \qquad (p, a, \mathbf{true}, \{\mathtt{x}\}, q) \qquad (q, a, \mathtt{x} < 1, \emptyset, q) \qquad (q, a, \mathtt{x} = 1, \emptyset, r).$$

Intuitively, in p the automaton waits until it guesses that the next input will be (a, t_i), at which point it moves to q by resetting the clock (and subsequently reading a). From q, the automaton can accept by going to r only if exactly one time unit elapsed since (a, t_i) was read. The language L is not recognised by any DTA since, intuitively, any deterministic acceptor needs to store unboundedly many timestamps t_i's.

Deterministic membership problems. Let $\mathcal{X}$ be a subclass of NTA. We are interested in the following decision problem.

$\mathcal{X}$ MEMBERSHIP PROBLEM.
Input: A timed automaton $A \in \mathsf{NTA}$.
Output: Does there exist a $B \in \mathcal{X}$ s.t. $L(A) = L(B)$?

In the rest of the paper, we study the decidability status of the $\mathcal{X}$ membership problem where $\mathcal{X}$ ranges over DTA, DTA_k (for every fixed number of clocks k), $\mathsf{DTA}_{\_,m}$ (for every maximal constant m), and $\mathsf{DTA}_{k,m}$ (when both clocks k and maximal constant m are fixed). Example 2.1 shows that there are NTA languages which cannot be accepted by any DTA. Moreover, there is no computable bound for the number of clocks k which suffice to recognise a NTA_1 language by a DTA_k (when such a number exists), which follows from the following three observations: 1) the DTA membership problem is undecidable for NTA_1 (Theorem 1.2), 2) the problem of deciding equivalence of a given NTA_1 to a given DTA is decidable [37], and 3) if a $\mathsf{NTA}_{1,m}$ is equivalent to some DTA_k then it is in fact equivalent to some $\mathsf{DTA}_{k,m}$ with computably many control locations (by Lemma 4.1).

3 Timed automorphisms and invariance

A fundamental tool in this paper is invariance properties of timed languages recognised by NTA with respect to permutations of $\mathbb{R}$ preserving integer differences. In this section we establish these properties. A *timed automorphism* is a monotone bijection $\pi : \mathbb{R} \to \mathbb{R}$ s.t. for every $x \in \mathbb{R}$, $\pi(x+1) = \pi(x)+1$. For instance, if $\pi(3.4) = 4.5$, then necessarily $\pi(5.4) = 6.5$ and $\pi(-3.6) = -2.5$. Timed automorphisms π are extended point-wise to timed words $\pi((a_1,t_1)\dots(a_n,t_n)) = (a_0,\pi(t_1))\dots(a_n,\pi(t_n))$, configurations $\pi(p,\mu,t_0) = (p,\pi\circ\mu,\pi(t_0))$, transitions $\pi(c \xrightarrow{a,t} c') = \pi(c) \xrightarrow{a,\pi(t)} \pi(c')$, and sets X thereof $\pi(X) = \{\pi(x) \mid x \in X\}$.

▶ Remark 3.1. A timed automorphism π can in general take a nonnegative real $t \geq 0$ to a negative one. Whenever we write $\pi(x)$, we always implicitly assume that π is defined on x.

Let $S \subseteq \mathbb{R}_{\geq 0}$. An *$S$-timed automorphism* is a timed automorphism s.t. $\pi(t) = t$ for all $t \in S$. Let Π_S denote the set of all S-timed automorphisms, and let $\Pi = \Pi_\emptyset$. A set X is *S-invariant* if $\pi(X) = X$ for every $\pi \in \Pi_S$; equivalently, for every $\pi \in \Pi_S$, $x \in X$ if, and only if $\pi(x) \in X$. A set X is *invariant* if it is S-invariant with $S = \emptyset$. The following three facts express some basic invariance properties.

▶ **Fact 3.2.** *The timed transition system $[\![A]\!]$ is invariant.*

By unrolling the definition of invariance in the previous fact, we obtain that the set of configurations is invariant, the set of transitions $\to$ is invariant, and that the set of final configurations F is invariant.

▶ **Fact 3.3** (Invariance of the language semantics)**.** *The function $c \mapsto L_A(c)$ from $[\![A]\!]$ to languages is invariant, i.e., for all timed permutations π, $L_A(\pi(c)) = \pi(L_A(c))$.*

▶ **Fact 3.4** (Invariance of the language of a configuration)**.** *The language $L_A(p,\mu,t_0)$ is $(\mu(\mathtt{X}) \cup \{t_0\})$-invariant. Moreover, if A is always resetting, then $L_A(p,\mu,t_0)$ is $\mu(\mathtt{X})$-invariant.*

Since timed automorphisms preserve integer differences, only the fractional parts of elements of $S \subseteq \mathbb{R}_{\geq 0}$ matter for S-invariance, and hence it makes sense to restrict to subsets of the half-open interval $[0,1)$. Let $\mathsf{fract}(S) = \{\mathsf{fract}(x) \mid x \in S\} \subseteq [0,1)$ stand for the set of fractional parts of elements of S. The following lemma shows that, modulo the irrelevant integer parts, there is always the least set S witnessing S-invariance.

▶ **Lemma 3.5.** *For finite subsets $S, S' \subseteq \mathbb{R}_{\geq 0}$, if a timed language L is both S-invariant and S'-invariant, then it is also S''-invariant where $S'' = \mathsf{fract}(S) \cap \mathsf{fract}(S')$.*

The S-*orbit* of an element $x \in X$ (which can be an arbitrary object on which the action of timed automorphisms is defined) is the set $\text{ORBIT}_S(x) = \{\pi(x) \in X \mid \pi \in \Pi_S\}$ of all elements $\pi(x)$ which can be obtained by applying some S-automorphism to x. The *orbit* of x is just its S-orbit with $S = \emptyset$, written $\text{ORBIT}(x)$. Clearly x and x' have the same S-orbit if, and only if, $\pi(x) = x'$ for some $\pi \in \Pi_S$. For greedily resetting NTA, orbits of single configurations are in bijective correspondence with bounded regions.

▶ **Fact 3.6.** *Assume A is a greedily resetting $\mathsf{NTA}_{k,m}$. Two reachable configurations (p, μ, t_0) and (p, μ', t_0') of A with the same control location p have the same orbit if, and only if, the corresponding clock valuations $t_0 - \mu$ and $t_0' - \mu'$ belong to the same k, m-region.*

The S-*closure* of a set Y, written $\Pi_S(Y) = \bigcup_{x \in Y} \text{ORBIT}_S(x)$, is the union of the S-orbits of all its elements. The following fact characterises invariance in term of closures.

▶ **Fact 3.7.** *A set Y is S-invariant if, and only if, $\Pi_S(Y) = Y$.*

Proof. Only if direction follows by the definition of S-invariance. For the converse direction observe that $\Pi_S(X) = X$ implies $\pi(X) \subseteq X$ for every $\pi \in \Pi_S$. The opposite inclusion follows by closure of S-timed automorphisms under inverse: $\pi^{-1}(X) \subseteq X$, hence $X \subseteq \pi(X)$. ◀

4 Decidability of DTA_k and $\mathsf{DTA}_{k,m}$ membership for NTA_1

In this section we prove Theorem 1.1 thus establishing decidability of the DTA_k and $\mathsf{DTA}_{k,m}$ membership problems for NTA_1. Both results are shown using the following key characterisation of DTA_k languages as a subclass of NTA_1 languages. In particular, this characterisation provides a small bound on the number of control locations of a DTA_k equivalent to a given NTA_1 (if any exists).

▶ **Lemma 4.1.** *Let A be a $\mathsf{NTA}_{1,m}$ with n control locations, and let $k \in \mathbb{N}$. The following conditions are equivalent:*

1. *$L(A) = L(B)$ for some always resetting DTA_k B.*
2. *For every timed word w, there is $S \subseteq \mathbb{R}_{\geq 0}$ of size at most k s.t. the last timestamp of w is in S and $w^{-1}L(A)$ is S-invariant.*
3. *$L(A) = L(B)$ for some always resetting $\mathsf{DTA}_{k,m}$ B with at most $f(k, m, n) = \mathsf{Reg}(k, m) \cdot 2^{n(2km+1)}$ control locations ($\mathsf{Reg}(k, m)$ stands for the number of k, m-regions).*

The proof of Theorem 1.1 builds on Lemma 4.1 and on the following fact:

▶ **Lemma 4.2.** *The DTA_k and $\mathsf{DTA}_{k,m}$ membership problems are both decidable for DTA languages.*

Proof. We reduce to a deterministic separability problem. Recall that a language S *separates* two languages L, M if $L \subseteq S$ and $S \cap M = \emptyset$. It has recently been shown that the DTA_k and $\mathsf{DTA}_{k,m}$ separability problems are decidable for NTA [18, Theorem 1.1], and thus, in particular, for DTA. To solve the membership problem, given a DTA A, the procedure computes a DTA A' recognising the complement of $L(A)$ and checks whether A and A' are DTA_k separable (resp., $\mathsf{DTA}_{k,m}$ separable) by using the result above. It is a simple set-theoretic observation that $L(A)$ is a DTA_k language if, and only if, the languages $L(A)$ and $L(A')$ are separated by some DTA_k language, and likewise for $\mathsf{DTA}_{k,m}$ languages. ◀

Proof of Theorem 1.1. We solve both problems in essentially the same way. Given a $\mathsf{NTA}_{1,m}$ A, the decision procedure enumerates all always resetting $\mathsf{DTA}_{k+1,m}$ B with at most $f(k, m, n)$ locations and checks whether $L(A) = L(B)$ (which is decidable by [37]). If no such

DTA_{k+1} B is found, the $L(A)$ is not an always resetting DTA_{k+1} language, due to Lemma 4.1, and hence forcedly is not a DTA_k language either; the procedure therefore answers negatively. Otherwise, in case when such a DTA_{k+1} B is found, then DTA_k membership (resp. $\mathsf{DTA}_{k,m}$ membership) test is performed on B, decidable due to Lemma 4.2. ◀

▶ Remark 4.3 (Complexity). The decision procedure for NTA_1 invokes the HyperAckermann subroutine of [37] to check equivalence between a NTA_1 and a candidate DTA. This is in a sense unavoidable, since we show in Lemma 5.5 that the DTA_k and $\mathsf{DTA}_{k,m}$ membership problems are HyperAckermann-hard for NTA_1.

In the rest of this section we present the proof of Lemma 4.1. Let us fix a $\mathsf{NTA}_{1,m}$ $A = (\Sigma, \mathtt{L}, \{\mathtt{x}\}, \mathtt{I}, \mathtt{F}, \Delta)$, where m is the greatest constant used in clock constraints in A, and $k \in \mathbb{N}$. We assume w.l.o.g. that A is greedily resetting: This is achieved by resetting the clock as soon as upon reading an input symbol its value becomes greater than m or is an integer $\leq m$; we can record in the control location the actual integral value if it is $\leq m$, or a special flag otherwise. Consequently, after every discrete transition the value of the clock is at most m, and if it is an integer then it equals 0.

The implication $3 \implies 1$ follows by definition. For the implication $1 \implies 2$ suppose, by assumption, $L(A) = L(B)$ for a total always resetting DTA_k B. Every left quotient $w^{-1}L(A)$ equals $L_B(c)$ for some configuration c, hence Point 2 follows by Fact 3.4. Here we use the fact that B is always resetting in order to apply the second part of Fact 3.4; without the assumption, we would only have S-invariance for sets S of size at most $k+1$.

It thus remains to prove the implication $2 \implies 3$, which will be the content of the rest of the section. Assuming Point 2, we are going to define an always resetting $\mathsf{DTA}_{k,m}$ B' with clocks $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$ and with at most $f(k,m,n)$ locations such that $L(B') = L(A)$. We start from the timed transition system $\mathcal{X}$ obtained by the finite powerset construction underlying the determinisation of A, and then transform this transition system gradually, while preserving its language, until it finally becomes isomorphic to the reachable part of $[\![B']\!]$ for some $\mathsf{DTA}_{k,m}$ B'. As the last step we extract from this deterministic timed transition system a syntactic definition of B' and prove equality of their languages. This is achievable due to the invariance properties witnessed by the transition systems in the course of the transformation.

Macro-configurations. Configurations of the NTA_1 A are of the form $c = (p, u, t_0)$ where $u, t_0 \in \mathbb{R}_{\geq 0}$ and $u \leq t_0$. A *macro-configuration* is a (not necessarily finite) set X of configurations (p, u, t_0) of A which share the same value of the current timestamp t_0, which we denote as $\textsc{now}(X) = t_0$. We use the notation $L_A(X) := \bigcup_{c \in X} L_A(c)$. Let $\textsc{succ}_{a,t}(X) := \left\{ c' \in [\![A]\!] \;\middle|\; c \xrightarrow{a,t} c' \text{ for some } c \in X \right\}$ be the set of successors of configurations in X. We define a deterministic timed transition system $\mathcal{X}$ consisting of the macro-configurations reachable in the course of determinisation of A. Let $\mathcal{X}$ be the smallest set of macro-configurations and transitions such that

- $\mathcal{X}$ contains the initial macro-configuration: $X_0 = \{(p, 0, 0) \mid p \in \mathtt{I}\} \in \mathcal{X}$;
- $\mathcal{X}$ is closed under successor: for every $X \in \mathcal{X}$ and $(a, t) \in \Sigma \times \mathbb{R}_{\geq 0}$, there is a transition $X \xrightarrow{a,t} \textsc{succ}_{a,t}(X)$ in $\mathcal{X}$.

Due to the fact that $[\![A]\!]$ is finitely branching, i.e. $\textsc{succ}_{a,t}(\{c\})$ is finite for every fixed (a, t), all macro-configurations $X \in \mathcal{X}$ are finite. Let the final configurations of $\mathcal{X}$ be $F_{\mathcal{X}} = \{X \in \mathcal{X} \mid X \cap F \neq \emptyset\}$.

▷ Claim 4.4. $L_A(X) = L_{\mathcal{X}}(X)$ for every $X \in \mathcal{X}$. In particular $L(A) = L_{\mathcal{X}}(X_0)$.

For a macro-configuration X we write $\textsc{Val}(X) := \{u \mid (p, u, \textsc{now}(X)) \in X\} \cup \{\textsc{now}(X)\}$ to denote the reals appearing in X. Since A is greedily resetting, every macro-configuration $X \in \mathcal{X}$ satisfies $\textsc{Val}(X) \subseteq (\textsc{now}(X) - m, \textsc{now}(X)]$. Whenever a macro-configuration X satisfies this condition we say that *the span of X is bounded by m*.

Pre-states. By assumption (Point 2), $L_A(X)$ is S-invariant for some S of size at most k, but the macro-configuration X itself needs not be S-invariant in general. Indeed, a finite macro-configuration $X \in \mathcal{X}$ is S-invariant if, and only if, $\mathsf{fract}(\textsc{Val}(X)) \subseteq \mathsf{fract}(S)$, which is impossible in general when X is arbitrarily large, its span is bounded (by m), and size of S is bounded (by k). Intuitively, in order to assure S-invariance we will replace X by its S-closure $\Pi_S(X)$ (recall Fact 3.7).

A set $S \subseteq \mathbb{R}_{\geq 0}$ is *fraction-independent* if it contains no two reals with the same fractional part. A *pre-state* is a pair $Y = (X, S)$, where X is an S-invariant macro-state, and S is a finite fraction-independent subset of $\textsc{Val}(X)$ that contains $\textsc{now}(X)$. The intuitive rationale behind assuming the S-invariance of X is that it implies, together with the bounded span of X and bounded size of S, that there are only finitely many pre-states, up to timed automorphism. We define the deterministic timed transition system $\mathcal{Y}$ as the smallest set of pre-states and transitions between them such that:

- $\mathcal{Y}$ contains the initial pre-state: $Y_0 = (X_0, \{0\}) \in \mathcal{Y}$;
- $\mathcal{Y}$ is closed under the closure of successor: for every $(X, S) \in \mathcal{Y}$ and $(a, t) \in \Sigma \times \mathbb{R}_{\geq 0}$, there is a transition $(X, S) \xrightarrow{a,t} (X', S')$, where S' is the least, with respect to set inclusion, subset of $S \cup \{t\}$ containing t such that the language $L' = (a, t)^{-1} L_A(X) = L_A(\textsc{succ}_{a,t}(X))$ is S'-invariant, and $X' = \Pi_{S'}(\textsc{succ}_{a,t}(X))$.

▶ **Example 4.5.** Suppose $k = 3$, $m = 2$, $\textsc{succ}_{a,t}(X) = \{(p, 3.7, 5), (q, 3.9, 5), (r, 4.2, 5)\}$ and $S' = \{3.7, 4.2, 5\}$. Then $X' = \{(p, 3.7, 5)\} \cup \{(q, t, 5) \mid t \in (3.7, 4)\} \cup \{(r, 4.2, 5)\}$. $\textsc{now}(X') = 5$. A corresponding *state* is (X', μ'), where $\mu' = \{\mathtt{x}_1 \mapsto 3.7, \mathtt{x}_2 \mapsto 4.2, \mathtt{x}_3 \mapsto 5\}$.

Observe that the least such fraction-independent subset S' exists due to the following facts: as X is S-invariant, due to Fact 3.3 so is its language $L_A(X)$, and hence L' is necessarily $(S \cup \{t\})$-invariant; by assumption (Point 2), L' is R-invariant for some set $R \subseteq \mathbb{R}_{\geq 0}$ of size at most k containing t; let $T \subseteq \mathbb{R}_{\geq 0}$ be the least set given by Lemma 3.5, i.e., $\mathsf{fract}(T) \subseteq \mathsf{fract}(S) \cap \mathsf{fract}(R)$; and finally let $S' \subseteq S$ be chosen so that $\mathsf{fract}(S') = \mathsf{fract}(T \cup \{t\})$. Due to fraction-independence of S the choice is unique, S' is fraction-independent, and $t \in S'$. Furthermore, the size of S' is at most k. By Fact 3.3, we deduce:

▷ Claim 4.6 (Invariance of $\mathcal{Y}$). For every two transitions $(X_1, S_1) \xrightarrow{a,t_1} (X_1', S_1')$ and $(X_2, S_2) \xrightarrow{a,t_2} (X_2', S_2')$ in $\mathcal{Y}$ and a timed permutation π, if $\pi(X_1) = X_2$ and $\pi(S_1) = S_2$ and $\pi(t_1) = t_2$, then we have $\pi(X_1') = X_2'$ and $\pi(S_1') = S_2'$.

Let the final configurations of $\mathcal{Y}$ be $F_{\mathcal{Y}} = \{(X, S) \in \mathcal{Y} \mid X \cap \mathtt{F} \neq \emptyset\}$. By induction on the length of timed words it is easy to show:

▷ Claim 4.7. $L_{\mathcal{X}}(X_0) = L_{\mathcal{Y}}(Y_0)$.

Due to the assumption that A is greedily resetting and due to Point 2, in every pre-state $(X, S) \in \mathcal{Y}$ the span of X is bounded by m and the size of S is bounded by k.

States. We now introduce *states*, which are designed to be in one-to-one correspondence with configurations of the forthcoming DTA_k B'. Intuitively, a state differs from a pre-state (X, S) only by allocating the values from S into k clocks, thus while a pre-state contains a set S, the corresponding state contains a clock assignment $\mu : \mathtt{X} \to \mathbb{R}_{\geq 0}$ with image $\mu(\mathtt{X}) = S$.

Let $\mathtt{X} = \{\mathtt{x}_1, \ldots, \mathtt{x}_k\}$ be a set of k clocks. A *state* is a pair $Z = (X, \mu)$, where X is a macro-configuration, $\mu : \mathtt{X} \to \textsc{Val}(X)$ is a clock reset-point assignment, $\mu(\mathtt{X})$ is a fraction-independent set containing $\textsc{now}(X)$, and X is $\mu(\mathtt{X})$-invariant. Thus every state $Z = (X, \mu)$ determines uniquely a corresponding pre-state $\sigma(Z) = (X, S)$ with $S = \mu(\mathtt{X})$. We define the deterministic timed transition system $\mathcal{Z}$ consisting of those states Z s.t. $\sigma(Z) \in \mathcal{Y}$, and of transitions determined as follows: $(X, \mu) \xrightarrow{a,t} (X', \mu')$ if the corresponding pre-state has a transition $(X, S) \xrightarrow{a,t} (X', S')$ in $\mathcal{Y}$, where $S = \mu(\mathtt{X})$, and

$$\mu'(\mathtt{x}_i) \;:=\; \begin{cases} t & \text{if } \mu(\mathtt{x}_i) \notin S' \text{ or } \mu(\mathtt{x}_i) = \mu(\mathtt{x}_j) \text{ for some } j > i \\ \mu(\mathtt{x}_i) & \text{otherwise.} \end{cases} \tag{3}$$

Intuitively, the equation (3) defines a deterministic update of the clock reset-point assignment μ that amounts to resetting ($\mu'(\mathtt{x}_i) := t$) all clocks $\mathtt{x}_i$ whose value is either no longer needed (because $\mu(\mathtt{x}_i) \notin S'$), or is shared with some other clock x_j, for $j > i$ and is thus redundant. Due to this disciplined elimination of redundancy, knowing that $t \in S'$ and the size of S' is at most k, we ensure that at least one clock is reset in every step. In consequence, $\mu'(\mathtt{X}) = S'$, and the forthcoming DTA_k B' will be always resetting. Using Claim 4.6 we derive:

▷ Claim 4.8 (Invariance of $\mathcal{Z}$). For every two transitions $(X_1, \mu_1) \xrightarrow{a,t_1} (X_1', \mu_1')$ and $(X_2, \mu_2) \xrightarrow{a,t_2} (X_2', \mu_2')$ in $\mathcal{Z}$ and a timed permutation π, if $\pi(X_1) = X_2$ and $\pi \circ \mu_1 = \mu_2$ and $\pi(t_1) = t_2$, then we have $\pi(X_1') = X_2'$ and $\pi \circ \mu_1' = \mu_2'$.

Let the initial state be $Z_0 = (X_0, \mu_0)$, where $\mu_0(\mathtt{x}_i) = 0$ for all $\mathtt{x}_i \in \mathtt{X}$, and let final states be $F_{\mathcal{Z}} = \{(X, \mu) \in \mathcal{Z} \mid X \cap F \neq \emptyset\}$. By induction on the length of timed words one proves:

▷ Claim 4.9. $L_{\mathcal{Y}}(Y_0) = L_{\mathcal{Z}}(Z_0)$.

In the sequel we restrict $\mathcal{Z}$ to states reachable from Z_0. In every state $Z = (X, \mu)$ in $\mathcal{Z}$, we have $\textsc{now}(X) \in \mu(\mathtt{X})$. This will ensure the resulting DTA_k B' to be always resetting.

Orbits of states. While a state is designed to correspond to a configuration of the forthcoming DTA_k B', its orbit is designed to play the role of control location of B'. We therefore need to prove that the set of states in $\mathcal{Z}$ is orbit-finite, i.e., the set of orbits $\{\textsc{orbit}(Z) \mid Z \in \mathcal{Z}\}$ is finite and its size is bounded by $f(k, m, n)$. We start by deducing an analogue of Fact 3.6:

▷ Claim 4.10. For two states $Z = (X, \mu)$ and $Z' = (X', \mu')$ in $\mathcal{Z}$, their clock assignments are in the same orbit, i.e., $\pi \circ \mu = \mu'$ for some $\pi \in \Pi$, if, and only if, the corresponding clock valuations $\textsc{now}(X) - \mu$ and $\textsc{now}(X') - \mu'$ belong to the same k, m-region.

(In passing note that, since in every state (X, μ) in $\mathcal{Z}$ the span of X is bounded by m, only bounded k, m-regions can appear in the last claim. Moreover, in each of k, m-regions one of clocks equals 0.) The action of timed automorphisms on macro-configurations and clock assignments is extended to states as $\pi(X, \mu) = (\pi(X), \pi \circ \mu)$. Recall that the orbit of a state Z is defined as $\textsc{orbit}(Z) = \{\pi(Z) \mid \pi \in \Pi\}$.

▷ Claim 4.11. The number of orbits of states in $\mathcal{Z}$ is bounded by $f(k, m, n)$.

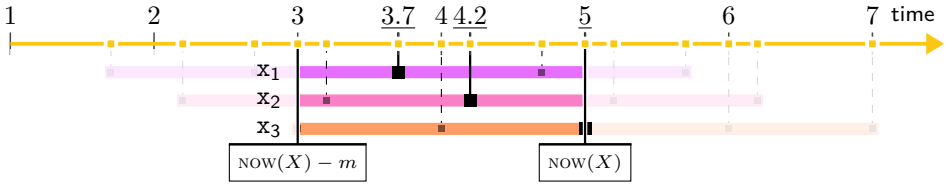
Proof. We finitely represent a state $Z = (X, \mu)$, relying on the following general fact.

▶ **Fact 4.12.** *For every $u \in \mathbb{R}_{\geq 0}$ and $S \subseteq \mathbb{R}_{\geq 0}$, the S-orbit[2] $\textsc{orbit}_S(u)$ is either the singleton $\{u\}$ (when $u \in S$) or an open interval with ends-points of the form $t + z$ where $t \in S$ and $z \in \mathbb{Z}$ (when $u \notin S$).*

[2] The orbits of states Z should not be confused with S-orbits of individual reals $u \in \mathbb{R}_{\geq 0}$.

We apply the fact above to $S = \mu(\mathtt{X})$. In our case the span of X is bounded by m, and thus the same holds for $\mu(\mathtt{X})$. Consequently, the integer z in the fact above always belongs to $\{-m, -m+1, \ldots, m\}$. In turn, X splits into disjoint $\mu(\mathtt{X})$-orbits $\textsc{orbit}_{\mu(\mathtt{X})}(u)$ consisting of open intervals separated by endpoints of the form $t + z$ where $t \in \mu(\mathtt{X})$ and $z \in \{-m, -m+1, \ldots, m\}$.

▶ **Example 4.13.** Continuing Example 4.5, the endpoints are $\{3, 3.2, 3.7, 4, 4.2, 4.7, 5\}$, as shown in the illustration:



Recall that $\mu(\mathtt{X})$ is fraction-independent. Let $e_1 < e_2 < \cdots < e_{l+1}$ be all the endpoints of open-interval orbits ($l \leq km$), and let $o_1, o_2, o_3, \ldots := \{e_1\}, (e_1, e_2), \{e_2\}, \ldots$ be the consecutive S-orbits $\textsc{orbit}_{\mu(\mathtt{X})}(u)$ of elements $u \in \mu(\mathtt{X})$. The number thereof is $2l + 1 \leq 2km + 1$. The finite representation of $Z = (X, \mu)$ consists of the pair (O, μ), where

$$O = \{(o_1, P_1), \ldots, (o_{2l+1}, P_{2l+1})\} \tag{4}$$

assigns to each orbit o_i the set of locations $P_i = \{p \mid (p, u, t_0) \in X \text{ for some } u \in o_i\} \subseteq \mathtt{L}$, (which is the same as $P_i = \{p \mid (p, u, t_0) \in X \text{ for all } u \in o_i\}$ since X is $\mu(\mathtt{X})$-invariant, and hence $\mu(\mathtt{X})$-closed). Thus a state $Z = (X, \mu)$ is uniquely determined by the sequence O as in (4) and the clock assignment μ.

We claim that the set of all the finite representations (O, μ), as defined above, is orbit-finite. Indeed, the orbit of (O, μ) is determined by the orbit of μ and the sequence

$$P_1,\ P_2,\ \ldots,\ P_{2km+1} \tag{5}$$

induced by the assignment O as in (4). Therefore, the number of orbits is bounded by the number of orbits of μ (which is bounded, due to Claim 4.10, by $\mathsf{Reg}(k, m)$) times the number of different sequences of the form (5) (which is bounded by $(2^n)^{2km+1}$). This yields the required bound $f(k, m, n) = \mathsf{Reg}(k, m) \cdot 2^{n(2km+1)}$. ◁

Construction of the DTA. As the last step we define a DTA_k $B' = (\Sigma, \mathtt{L}', \mathtt{X}, \{o_0\}, \mathtt{F}', \Delta')$ such that the reachable part of $[\![B']\!]$ is isomorphic to $\mathcal{Z}$. Let locations $\mathtt{L}' = \{\textsc{orbit}(Z) \mid Z \in \mathcal{Z}\}$ be orbits of states from $\mathcal{Z}$, the initial location be the orbit o_0 of Z_0, and final locations $\mathtt{F}' = \{\textsc{orbit}(Z) \mid Z \in F_{\mathcal{Z}}\}$ be orbits of final states. A transition $Z = (X, \mu) \xrightarrow{a,t} (X', \mu') = Z'$ in $\mathcal{Z}$ induces a transition rule in B'

$$(o, a, \psi, \mathtt{Y}, o') \in \Delta' \tag{6}$$

whenever $o = \textsc{orbit}(Z)$, $o' = \textsc{orbit}(Z')$, ψ is the unique k, m-region satisfying $t - \mu \in [\![\psi]\!]$, and $\mathtt{Y} = \{\mathtt{x}_i \in \mathtt{X} \mid \mu'(\mathtt{x}_i) = t\}$. The automaton B' is indeed a DTA since o, a and ψ uniquely determine $\mathtt{Y}$ and o':

▷ Claim 4.14. Suppose that two transitions $(X_1, \mu_1) \xrightarrow{a,t_1} (X_1', \mu_1')$ and $(X_2, \mu_2) \xrightarrow{a,t_2} (X_2', \mu_2')$ in $\mathcal{Z}$ induce transition rules $(o, a, \psi, \mathtt{Y}_1, o_1'), (o, a, \psi, \mathtt{Y}_2, o_2') \in \Delta'$ with the same source location o and constraint ψ, i.e,

$$t_1 - \mu_1 \in [\![\psi]\!] \qquad t_2 - \mu_2 \in [\![\psi]\!]\,. \tag{7}$$

Then the target locations are equal $o_1' = o_2'$, and the same for the reset sets $\mathtt{Y}_1 = \mathtt{Y}_2$.

Proof. We use the invariance of semantics of A and Claim 4.8. Let $o = \textsc{orbit}(X_1, \mu_1) = \textsc{orbit}(X_2, \mu_2)$. Thus there is a timed automorphism π such that

$$X_2 = \pi(X_1) \qquad \mu_2 = \pi \circ \mu_1. \tag{8}$$

It suffices to show that there is a (possibly different) timed permutation σ satisfying the following equalities:

$$t_2 = \sigma(t_1) \quad \{i \mid \mu_1'(\mathtt{x}_i) = t_1\} = \{i \mid \mu_2'(\mathtt{x}_i) = t_2\} \quad \mu_2' = \sigma \circ \mu_1' \quad X_2' = \sigma(X_1'). \tag{9}$$

We now rely the fact that both $t_{01} = \textsc{now}(X_1) \in \mu_1(\mathtt{X})$ and $t_{02} = \textsc{now}(X_2) \in \mu_2(\mathtt{X})$ are assigned to (the same) clock due to the second equality in (8): $t_{01} = \mu_1(\mathtt{x}_i)$ and $t_{02} = \mu_2(\mathtt{x}_i)$. We focus on the case when $t_1 - t_{01} \leq m$ (the other case is similar but easier as all clock are reset due to greedy resetting), which implies $t_2 - t_{02} \leq m$ due to (7). In this case we may assume w.l.o.g., due to (7) and the equalities (8), that π is chosen so that $\pi(t_1) = t_2$. We thus take $\sigma = \pi$ for proving the equalities (9). Being done with the first equality, we observe that the last two equalities in (9) hold due to the invariance of $\mathcal{Z}$ (cf. Claim 4.8). The remaining second equality in (9) is a consequence of the third one. ◁

▷ Claim 4.15. Let $Z = (X, \mu)$ and $Z' = (X', \mu)$ be two states in $\mathcal{Z}$ with the same clock assignment. If $\pi(X) = X'$ and $\pi \circ \mu = \mu$ for some timed automorphism π then $X = X'$.

▷ Claim 4.16. $\mathcal{Z}$ is isomorphic to the reachable part of $[\![B']\!]$.

Proof. For a state $Z = (X, \mu)$, let $c(Z) = (o, \mu, t)$, where $o = \textsc{orbit}(Z)$ and $t = \textsc{now}(X)$. By Claim 4.15, the mapping $c(\_)$ is a bijection between $\mathcal{Z}$ and its image $c(\mathcal{Z}) \subseteq [\![B']\!]$. By (6), $\mathcal{Z}$ is isomorphic to a subsystem of the reachable part of $[\![B']\!]$. The converse inclusion follows by the observation that $\mathcal{Z}$ is total: for every $(a_1, t_1) \dots (a_n, t_n) \in \mathbb{T}(\Sigma)$, there is a sequence of transitions $(X_0, \mu_0) \xrightarrow{a_1,t_1} \cdots \xrightarrow{a_n,t_n}$ in $\mathcal{Z}$. ◁

Claims 4.4, 4.7, 4.9, and 4.16 prove $L(A) = L(B')$.

5 Undecidability and hardness

In this section we complete the decidability status of the deterministic membership problem by providing matching undecidability and hardness results. In Section 5.1 we prove undecidability of the DTA_m embership problem for NTA_1 (cf. Theorem 1.2) and in Section 5.2 we prove HyperAckermann-hardness of the DTA_k membership problem for NTA_1 (cf. Theorem 1.3).

5.1 Undecidability of DTA and $\mathsf{DTA}_{\_,m}$ membership for NTA_1

It has been shown in [22, Theorem 1] that it is undecidable whether a NTA_k timed language can be recognised by some DTA, for any fixed $k \geq 2$. This was obtained by a reduction from the NTA_k universality problem, which is undecidable for any fixed $k \geq 2$. While the universality problem becomes decidable for $k = 1$, we show in this section that, as announced in Theorem 1.2, the DTA membership problem remains undecidable for NTA_1.

Since the universality problem for NTA_1 is decidable, we need to reduce from another (undecidable) problem. Our candidate is the finiteness problem of lossy counter machines, which is undecidable [35, Theorem 13]. A *k-counters lossy counter machine* (k-LCM) is a tuple $M = (C, Q, q_0, \Delta)$, where $C = \{c_1, \dots, c_k\}$ is a set of k counters, Q is a finite set of control locations, $q_0 \in Q$ is the initial control location, and Δ is a finite set of instructions

of the form (p, op, q), where op is one of $c\,\mathsf{++}$, $c\,\mathsf{-}$, and $c \stackrel{?}{=} 0$. A configuration of an LCM M is a pair (p, u), where $p \in Q$ is a control location, and $u \in \mathbb{N}^C$ is a counter valuation. For two counter valuations $u, v \in \mathbb{N}^C$, we write $u \leq v$ if $u(c) \leq v(c)$ for every counter $c \in C$. The semantics of an LCM M is given by a (potentially infinite) transition system over the configurations of M s.t. there is a transition $(p, u) \xrightarrow{\delta} (q, v)$, for $\delta = (p, \mathsf{op}, q) \in \Delta$, whenever

1) $\mathsf{op} = c\,\mathsf{++}$ and $v \leq u[c \mapsto u(c) + 1]$, or
2) $\mathsf{op} = c\,\mathsf{-}$ and $v \leq u[c \mapsto u(c) - 1]$, or
3) $\mathsf{op} = c \stackrel{?}{=} 0$ and $u(c) = 0$ and $v \leq u$.

The *finiteness problem* (a.k.a. space boundedness) for an LCM M asks to decide whether the reachability set $\mathrm{Reach}(M) = \{(p, u) \mid (q_0, u_0) \to^* (p, u)\}$ is finite, where u_0 is the constantly 0 counter valuation.

▶ **Theorem 5.1** ([35, Theorem 13]). *The 4-LCM finiteness problem is undecidable.*

We use the following encoding of LCM runs as timed words over the alphabet $\Sigma = Q \cup \Delta \cup C$ (cf. [33, Definition 4.6] for a similar encoding). We interpret a counter valuation $u \in \mathbb{N}^C$ as the word over Σ

$$u \;=\; \underbrace{c_1 c_1 \cdots c_1}_{u(c_1)\text{ letters}}\ \underbrace{c_2 c_2 \cdots c_2}_{u(c_2)\text{ letters}}\ \underbrace{c_3 c_3 \cdots c_3}_{u(c_3)\text{ letters}}\ \underbrace{c_4 c_4 \cdots c_4}_{u(c_4)\text{ letters}}.$$

With this interpretation, we encode an LCM run $\pi = (p_0, u_0) \xrightarrow{\delta_1} (p_1, u_1) \xrightarrow{\delta_2} \cdots \xrightarrow{\delta_n} (p_n, u_n)$ as the following timed word, called the *reversal-encoding* of π,

$$p_n \delta_n u_n \quad \cdots \quad p_1 \delta_1 u_1 \quad p_0 u_0,$$

s.t. p_n occurs at time 0, for every $1 \leq i < n$, p_i occurs exactly after one time unit since p_{i+1}, and if a "unit" of counter c_1 did not disappear due to lossiness when going from u_i to u_{i+1}, then the timestamps of the corresponding occurrences of letter c_1 in u_i and u_{i+1} are also at distance one (and similarly for the other counters). Under the encoding above, we can build a NTA_1 A recognising the complement of the set of reversal-encodings of the runs of M ([33] for more details about the construction of A). Intuitively, when reading the reversal-encoding of a run of M, the counters are allowed to spontaneously increase. Therefore, the only kind of error that A must verify is that some counter spontaneously decreases. This can be done by guessing an occurrence of letter (say) c_1 in the current configuration which does not have a corresponding occurrence in the next configuration after exactly one time unit. This check can be performed by an NTA with one clock.

▶ **Lemma 5.2.** *The set of reachable configurations $\mathrm{Reach}(M)$ is finite if, and only if, $L(A)$ is a deterministic timed language.*

Since the timed automaton constructed in the proof uses only constant 1, the reduction works also for the $\mathsf{DTA}_{\_,m}$ membership problem for every $m > 0$:

▶ **Corollary 5.3.** *For every fixed $m > 0$, the $\mathsf{DTA}_{\_,m}$ membership problem for NTA_1 languages is undecidable.*

This result is the best possible in terms of the parameter m since the problem becomes decidable for $m = 0$. In fact, the class of $\mathsf{DTA}_{k,0}$ languages coincides with the class of $\mathsf{DTA}_{1,0}$ languages (one clock is sufficient; cf. [37, Lemma 19]), and thus $\mathsf{DTA}_{\_,0}$ membership reduces to $\mathsf{DTA}_{1,0}$ membership, which is decidable for NTA_1 by Theorem 1.1.

▶ Remark 5.4. We observe that the reduction above uses a large alphabet Σ whose size depends on the input LCM M. In fact, an alternative encoding exists using a unary alphabet $\Sigma = \{a\}$. Let the input LCM M have control locations $Q = \{p_1, \ldots, p_m\}$ and instructions $\Delta = \{\delta_1, \ldots, \delta_n\}$. An LCM configuration $p_j \delta_k u$ is represented by the timed word consisting of 6 blocks $\underbrace{a \cdots a}_{j \text{ letters}}\ \underbrace{a \cdots a}_{k \text{ letters}}\ \underbrace{a \cdots a}_{u(c_1) \text{ letters}}\ \underbrace{a \cdots a}_{u(c_2) \text{ letters}}\ \underbrace{a \cdots a}_{u(c_3) \text{ letters}}\ \underbrace{a \cdots a}_{u(c_4) \text{ letters}}$ s.t. in each block the last a is at timed distance exactly one from the last a of the previous block. A unit of counter c_1 now repeats at distance 6 in the next configuration (instead of 1). This shows that the DTA membership problem is undecidable for NTA_1 using maximal constant $m = 6$ over a unary alphabet.

5.2 Undecidability and hardness for DTA_k and $\mathsf{DTA}_{k,m}$ membership

All the lower bounds in this section are obtained by a reduction from the universality problem for the respective language classes (does a given language $L \subseteq \mathbb{T}(\Sigma)$ satisfy $L = \mathbb{T}(\Sigma)$?). The reduction is a suitable adaptation, generalization, and simplification of [22, Theorem 1] showing undecidability of DTA membership for NTA languages.

A timed language L is *timeless* if $L = L(A)$ for $A \in \mathsf{NTA}_0$ a timed automaton with no clocks (hence timestamps appearing in input words are irrelevant for acceptance). For two languages $L \subseteq \mathbb{T}(\Sigma)$ and $M \subseteq \mathbb{T}(\Gamma)$, and a fresh alphabet symbol $\$ \notin \Sigma \cup \Gamma$, we define their *composition* $L \rhd \{\$\} \rhd M$ to be the following timed language over $\Sigma' = \Sigma \cup \{\$\} \cup \Gamma$:

$$L \rhd \{\$\} \rhd M \;=\; \{v(\$,t)(a_1, t_1 + t) \ldots (a_n, t_n + t) \in \mathbb{T}(\Sigma') \mid v \in L, (a_1, t_1) \ldots (a_n, t_n) \in M\}\,.$$

▶ **Lemma 5.5.** *Let $k, m \in \mathbb{N}$ and let $\mathcal{Y}$ be a class of timed languages that*

1. *contains all the timeless timed languages,*
2. *is closed under union and composition, and*
3. *contains some non-DTA_k (resp. non-$\mathsf{DTA}_{k,m}$) language.*

The universality problem for languages in $\mathcal{Y}$ reduces in polynomial time to the DTA_k (resp. $\mathsf{DTA}_{k,m}$) membership problem for languages in $\mathcal{Y}$.

We immediately obtain Theorem 1.3 as a corollary of Lemma 5.5, thanks to the following observations. First, the lemma is applicable by taking as $\mathcal{Y}$ the classes of languages recognised by NTA_1 since this class contains all timeless timed languages, is closed under union and composition, and is not included in DTA_k for any k nor in $\mathsf{DTA}_{k,m}$ for any k, m (cf. the NTA_1 language from Example 2.1 which is not recognised by any DTA). Second, HyperAckermann-hardness of the universality problem for NTA_1 follows form the same lower bound for the reachability problem in lossy channel systems [15, Theorem 5.5], together with the reduction from this problem to universality of NTA_1 given in [33, Theorem 4.1].

Since the universality problem is undecidable for NTA_2 [2, Theorem 5.2] and $\mathsf{NTA}_1^\varepsilon$ (NTA_1 with epsilon steps) [33, Theorem 5.3], using the same reasoning we can apply Lemma 5.5 to observe that the DTA_k and $\mathsf{DTA}_{k,m}$ membership problems are undecidable for NTA_2 and $\mathsf{NTA}_1^\varepsilon$, which refines the analysis of [22, Theorem 1].

6 Conclusions

We have shown decidability and undecidability results for several variants of the deterministic membership problem for timed automata. Regarding undecidability, we have extended the previously known results [22, 42] by proving that the DTA membership problem is undecidable already for NTA_1 (Theorem 1.2), and, over a unary input alphabet, it is undecidable for

$\mathsf{NTA}_{1,m}$ with $m \geq 6$ (Remark 5.4). We leave open the question of what is the minimal m guaranteeing undecidability. Regarding decidability, we have shown that when the resources available to the deterministic automaton are fixed (either just the number of clocks k, or both clocks k and maximal constant m), then the respective deterministic membership problem is decidable (Theorem 1.1) and HyperAckermann-hard (Theorem 1.3).

Our deterministic membership algorithm is based on a characterisation of NTA_1 languages which happen to be DTA_k (Lemma 4.1), which is proved using a semantic approach leveraging on notions from the theory of sets with atoms [10]. Analogous decidability results for register automata can be obtained with similar techniques. It would be interesting to compare this approach to the syntactic determinisation method of [6].

Finally, our decidability results extend to the slightly more expressive class of always resetting NTA_2, which have intermediate expressive power strictly between NTA_1 and NTA_2.

References

1. S. Akshay, Paul Gastin, and Shankara Narayanan Krishna. Analyzing Timed Systems Using Tree Automata. *Logical Methods in Computer Science*, Volume 14, Issue 2, May 2018. doi:10.23638/LMCS-14(2:8)2018.
2. Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126:183–235, 1994.
3. Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: a determinizable class of timed automata. *Theor. Comput. Sci.*, 211:253–273, January 1999.
4. Eugene Asarin and Oded Maler. As soon as possible: Time optimal control for timed automata. In *Proc. of HSCC'99*, HSCC '99, pages 19–30, London, UK, UK, 1999. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=646879.710314.
5. Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proc. of the 5th IFAC Conference on System Structure and Control (SSSC'98)*, volume 31 (18), pages 447–452, 1998. doi:10.1016/S1474-6670(17)42032-5.
6. Christel Baier, Nathalie Bertrand, Patricia Bouyer, and Thomas Brihaye. When are timed automata determinizable? In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris Nikoletseas, and Wolfgang Thomas, editors, *Proc of ICALP'09*, pages 43–54, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
7. Vince Bárány, Christof Löding, and Olivier Serre. Regularity problems for visibly pushdown languages. In *Proc. of STACS'06*, STACS'06, pages 420–431, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11672142_34.
8. Gerd Behrmann, Alexandre David, Kim G. Larsen, John Hakansson, Paul Petterson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. In *Proceedings of the 3rd International Conference on the Quantitative Evaluation of Systems*, QEST '06, pages 125–126, Washington, DC, USA, 2006. IEEE Computer Society. doi:10.1109/QEST.2006.59.
9. Nathalie Bertrand, Amélie Stainer, Thierry Jéron, and Moez Krichen. A game approach to determinize timed automata. *Formal Methods in System Design*, 46(1):42–80, 2015. doi:10.1007/s10703-014-0220-1.
10. Mikolaj Bojańczyk and Sławomir Lasota. A machine-independent characterization of timed languages. In *Proc. ICALP 2012*, pages 92–103, 2012.
11. Patricia Bouyer, Fabrice Chevalier, and Deepak D'Souza. Fault diagnosis using timed automata. In *Proc. of FOSSACS'05*, pages 219–233, Berlin, Heidelberg, 2005. Springer-Verlag. doi:10.1007/978-3-540-31982-5_14.
12. Thomas Brihaye, Thomas A. Henzinger, Vinayak S. Prabhu, and Jean-François Raskin. Minimum-time reachability in timed games. In Lars Arge, Christian Cachin, Tomasz Jurdziński, and Andrzej Tarlecki, editors, *In Proc. of ICALP'07*, pages 825–837, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

13 Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. On the expressiveness of Parikh automata and related models. In Rudolf Freund, Markus Holzer, Carlo Mereghetti, Friedrich Otto, and Beatrice Palano, editors, *Proc. of NCMA'11*, volume 282 of *books@ocg.at*, pages 103–119. Austrian Computer Society, 2011.

14 Franck Cassez, Alexandre David, Emmanuel Fleury, Kim G. Larsen, and Didier Lime. Efficient on-the-fly algorithms for the analysis of timed games. In Martín Abadi and Luca de Alfaro, editors, *Proc. of CONCUR'05*, pages 66–80, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

15 Pierre Chambart and Philippe Schnoebelen. The ordinal recursive complexity of lossy channel systems. In *Proc. of LICS'08*, pages 205–216, 2008.

16 Lorenzo Clemente, Piotr Hofman, and Patrick Totzke. Timed Basic Parallel Processes. In Wan Fokkink and Rob van Glabbeek, editors, *Proc. of CONCUR'19*, volume 140 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 15:1–15:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CONCUR.2019.15.

17 Lorenzo Clemente, Sławomir Lasota, and Radosław Piórkowski. Determinisability of one-clock timed automata. *arXiv e-prints*, July 2020. arXiv:2007.09340.

18 Lorenzo Clemente, Sławomir Lasota, and Radosław Piórkowski. Timed games and deterministic separability. In *Proc. of ICALP 2020*, pages 121:1–121:16, 2020.

19 Hubert Comon and Yan Jurski. Timed automata and the theory of real numbers. In *Proc. of CONCUR'99*, pages 242–257, London, UK, UK, 1999. Springer-Verlag.

20 C. Dima. Computing reachability relations in timed automata. In *Proc. of LICS'02*, pages 177–186, 2002.

21 John Fearnley and Marcin Jurdziński. Reachability in two-clock timed automata is PSPACE-complete. *Information and Computation*, 243:26–36, 2015. doi:10.1016/j.ic.2014.12.004.

22 Olivier Finkel. Undecidable problems about timed automata. In *Proc. of FORMATS'06*, pages 187–199, Berlin, Heidelberg, 2006. Springer-Verlag. doi:10.1007/11867340_14.

23 Martin Fränzle, Karin Quaas, Mahsa Shirmohammadi, and James Worrell. Effective definability of the reachability relation in timed automata. *Information Processing Letters*, 153:105871, 2020. doi:10.1016/j.ipl.2019.105871.

24 Laurent Fribourg. A closed-form evaluation for extended timed automata. Technical report, CNRS & ECOLE NORMALE SUPERIEURE DE CACHAN, 1998.

25 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Reachability in Timed Automata with Diagonal Constraints. In Sven Schewe and Lijun Zhang, editors, *Proc. of CONCUR'18*, volume 118 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:17, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CONCUR.2018.28.

26 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Fast algorithms for handling diagonal constraints in timed automata. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification*, pages 41–59, Cham, 2019. Springer International Publishing.

27 Stefan Göller and Paweł Parys. Bisimulation finiteness of pushdown systems is elementary. In *Proc. of LICS'20*, pages 521–534, 2020.

28 R. Govind, Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Revisiting Local Time Semantics for Networks of Timed Automata. In Wan Fokkink and Rob van Glabbeek, editors, *Proc. of CONCUR 2019*, volume 140 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.CONCUR.2019.16.

29 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. *Information and Computation*, 251:67–90, 2016. doi:10.1016/j.ic.2016.07.004.

30 Marcin Jurdziński and Ashutosh Trivedi. Reachability-time games on timed automata. In *In Proc. of ICALP'07*, pages 838–849, Berlin, Heidelberg, 2007. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=2394539.2394637.

31 Pavel Krčál and Radek Pelánek. On sampled semantics of timed systems. In Sundar Sarukkai and Sandeep Sen, editors, *In Proc. of FSTTCS'05*, volume 3821 of *LNCS*, pages 310–321. Springer, 2005. doi:10.1007/11590156_25.

32 M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. of CAV'11*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

33 Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Trans. Comput. Logic*, 9(2):10:1–10:27, 2008. doi:10.1145/1342991.1342994.

34 Oded Maler and Amir Pnueli. On recognizable timed languages. In Igor Walukiewicz, editor, *Proc. of FOSSACS'04*, volume 2987 of *LNCS*, pages 348–362. Springer Berlin Heidelberg, 2004. doi:10.1007/978-3-540-24727-2_25.

35 Richard Mayr. Undecidable problems in unreliable computations. *Theor. Comput. Sci.*, 297(1-3):337–354, March 2003. doi:10.1016/S0304-3975(02)00646-1.

36 Brian Nielsen and Arne Skou. Automated test generation from timed automata. *International Journal on Software Tools for Technology Transfer*, 5(1):59–77, November 2003. doi:10.1007/s10009-002-0094-1.

37 Joël Ouaknine and James Worrell. On the language inclusion problem for timed automata: Closing a decidability gap. In *Proc. of LICS'04*, pages 54–63, 2004. doi:10.1109/LICS.2004.1319600.

38 Joel Ouaknine and James Worrell. On the decidability and complexity of Metric Temporal Logic over finite words. *Logical Methods in Computer Science*, Volume 3, Issue 1, February 2007. doi:10.2168/LMCS-3(1:8)2007.

39 Jeffrey Shallit. *A Second Course in Formal Languages and Automata Theory*. Cambridge University Press, 2008. doi:10.1017/CBO9780511808876.

40 P. Vijay Suman, Paritosh K. Pandya, Shankara Narayanan Krishna, and Lakshmi Manasa. Timed automata with integer resets: Language inclusion and expressiveness. In *Proc. of FORMATS'08*, pages 78–92, Berlin, Heidelberg, 2008. Springer-Verlag. doi:10.1007/978-3-540-85778-5_7.

41 Martin Tappler, Bernhard K. Aichernig, Kim Guldstrand Larsen, and Florian Lorber. Time to learn - learning timed automata from tests. In Étienne André and Mariëlle Stoelinga, editors, *Proc. of FORMATS'19*, pages 216–235, Cham, 2019. Springer International Publishing.

42 Stavros Tripakis. Folk theorems on the determinization and minimization of timed automata. *Inf. Process. Lett.*, 99(6):222–226, September 2006.

43 Leslie G. Valiant. Regularity and related problems for deterministic pushdown automata. *J. ACM*, 22(1):1–10, January 1975. doi:10.1145/321864.321865.

44 Rüdiger Valk and Guy Vidal-Naquet. Petri nets and regular languages. *Journal of Computer and System Sciences*, 23(3):299–325, 1981. doi:10.1016/0022-0000(81)90067-2.

45 Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. An algorithm for learning real-time automata. In *Proc of. the Annual Belgian-Dutch Machine Learning Conference (Benelearn'078)*, 2007.

Synthesis of Computable Regular Functions of Infinite Words

Vrunda Dave
IIT Bombay, India
vrunda@cse.iitb.ac.in

Emmanuel Filiot
Université Libre de Bruxelles, Belgium
efiliot@ulb.ac.be

Shankara Narayanan Krishna
IIT Bombay, India
krishnas@cse.iitb.ac.in

Nathan Lhote
MIMUW, University of Warsaw, Poland
nlhote@mimuw.edu.pl

Abstract

Regular functions from infinite words to infinite words can be equivalently specified by MSO-transducers, streaming ω-string transducers as well as deterministic two-way transducers with look-ahead. In their one-way restriction, the latter transducers define the class of rational functions. Even though regular functions are robustly characterised by several finite-state devices, even the subclass of rational functions may contain functions which are not computable (by a Turing machine with infinite input). This paper proposes a decision procedure for the following synthesis problem: given a regular function f (equivalently specified by one of the aforementioned transducer model), is f computable and if it is, synthesize a Turing machine computing it.

For regular functions, we show that computability is equivalent to continuity, and therefore the problem boils down to deciding continuity. We establish a generic characterisation of continuity for functions preserving regular languages under inverse image (such as regular functions). We exploit this characterisation to show the decidability of continuity (and hence computability) of rational and regular functions. For rational functions, we show that this can be done in NLogSpace (it was already known to be in PTime by Prieur). In a similar fashion, we also effectively characterise uniform continuity of regular functions, and relate it to the notion of uniform computability, which offers stronger efficiency guarantees.

2012 ACM Subject Classification Theory of computation → Transducers; Theory of computation → Automata over infinite objects; Theory of computation → Computability

Keywords and phrases transducers, infinite words, computability, continuity, synthesis

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.43

Related Version A full version of the paper is available at `https://arxiv.org/abs/1906.04199`.

Funding *Emmanuel Filiot*: This work is partially supported by the MIS project F451019F (F.R.S.-FNRS) and the EOS project Verifying Learning Artificial Intelligence Systems (F.R.S.-FNRS and FWO). Emmanuel Filiot is research associate at F.R.S.-FNRS.

1 Introduction

Let Inputs and $\mathsf{Outputs}$ be two arbitrary sets of elements called inputs and outputs respectively. A general formulation of the synthesis problem is as follows: for a given specification of a function $S\colon \mathsf{Inputs} \to 2^{\mathsf{Outputs}}$ relating any input $u \in \mathsf{dom}(S)$[1] to a set of outputs $S(u) \subseteq \mathsf{Outputs}$, decide whether there exists a (total) function $f\colon \mathsf{dom}(S) \to \mathsf{Outputs}$ such that (i) for all $u \in \mathsf{dom}(S)$, $f(u) \in S(u)$ and (ii) f satisfies some additional constraints such as being computable in some way, by some device which is effectively returned by the synthesis procedure. Assuming the axiom of choice, the relaxation of this problem without constraint (ii) always has a positive answer. However with additional requirement (ii), a function f realising S may not exist in general. In this paper, we consider the particular case where the specification S is *functional*,[2] in the sense that $S(u)$ is singleton set for all $u \in \mathsf{dom}(S)$. Even in this particular case, S may not be realisable while satisfying requirement (ii).

The latter observation on the functional case can already be made in the Church approach to synthesis [1, 18], for which $\mathsf{Inputs}, \mathsf{Outputs}$ are sets of infinite words and f is required to be implementable by a Mealy machine (a deterministic automaton which can output symbols). More precisely, an infinite word α over a finite alphabet Σ is a function $\alpha\colon \mathbb{N} \to \Sigma$ and is written as $\alpha = \alpha(0)\alpha(1)\ldots$. The set of infinite words over Σ is denoted by Σ^ω. In Church ω-regular synthesis, we have $\mathsf{Inputs} = \Sigma_i^\omega$ and $\mathsf{Outputs} = \Sigma_o^\omega$, and functions S are specified by ω-automata over $\Sigma_i.\Sigma_o$. Thus, such an automaton defines a language $L \subseteq (\Sigma_i.\Sigma_o)^\omega$ and in turn, through projection, a function S_L defined by $S_L(i_1 i_2 \ldots) = \{o_1 o_2 \cdots \mid i_1 o_1 i_2 o_2 \cdots \in L\}$. Such a specification is said to be synchronous, meaning that they alternatively read an input symbol and produce an output symbol deterministically. It is also ω-regular because it can be represented as an automaton over $\Sigma_i.\Sigma_o$. As an example, consider $\Sigma_i = \Sigma_o = \{a, b, @\}$ and the function S_{swap} defined only for all words of the form $u_1\sigma@u_2$ such that $u_1 \in \{a, b\}^*$ and $\sigma \in \{a, b\}$ by $S(u_1\sigma@u_2) = \sigma u_1@u_2$. The specification S is easily seen to be synchronous and ω-regular, but not realisable by any Mealy machine. This is because a Mealy machine is an input deterministic model and so cannot guess the last symbol before the @ symbol.

Computability of functions over infinite words. In Church synthesis, the notion of computability used for requirement (ii) is that of being computable by a Mealy machine. While this makes sense in a reactive scenario where output symbols (reactions) have to be produced immediately after input symbols are received, this computability notion is too strong in a more relaxed scenario where reactivity is not required. Instead, we propose here to investigate the synthesis problem for functional specifications over infinite words where the computability assumption (ii) for f is just being computable by some algorithm (formally a Turing machine) running on infinite inputs. In other words, our goal is to synthesize *algorithms* from specifications of functions of infinite words. There are classical computability notions for infinite objects, like infinite sequences of natural numbers, motivated by real analysis, or computation of functions of real numbers. The model of computation we consider for infinite words is a deterministic machine with 3 tapes : a read-only one-way tape holding the input, a two-way working tape with no restrictions and a write-only one-way output tape. All three tapes are infinite on the right. A function f is computable if there exists such a machine M such that, if its input tape is fed with an infinite word x in the domain of f, then M outputs longer and longer prefixes of $f(x)$ when reading longer and longer prefixes

[1] $\mathsf{dom}(S)$ is the domain of S, *i.e.* the set of inputs that have a non-empty image by S.
[2] In this case, we just write $S(u) = v$ instead of $S(u) = \{v\}$.

of x. This machine model has been defined in [25, Chap. 2]. If additionally one requires the existence of a computable function $m\colon \mathbb{N} \to \mathbb{N}$ such that for all $i \in \mathbb{N}$, for all infinite input x, M writes at least i output symbols when reading $m(i)$ input symbols of x, we obtain the notion of uniform computability. This offers promptness guarantees on the production of output symbols with respect to the number of symbols read on input.

Obviously, not all functions are computable. In this paper, we aim to solve the following synthesis problem: given a (finite) specification of a (partial) function f from infinite words to infinite words, is f computable (respectively uniformly computable)? If it is the case, then the procedure should return a Turing machine computing f (respectively uniformly computing f).

Examples. The function S_{swap} is computable. Since it is defined over all inputs containing at least one @ symbol, if a Turing machine is fed with such a word, it suffices for it to read its input until the first @ symbol is met, store in memory the symbol $\sigma \in \{a, b\}$ just before @, come back to the beginning of the tape and start producing the output infinite word $\sigma u_1 @ u_2$.

Over the alphabet $\Sigma = \{a, b\}$, consider the function f_∞ defined by $f_\infty(u) = a^\omega$ if u contains infinitely many as, and by b^ω otherwise. This simple function is not computable, as it requires to read the whole infinite input to produce even the very first output symbol. For any word $u \in \Sigma^*$, we denote by $\overline{u}$ its mirror (e.g. $\overline{abaa} = aaba$). Consider the (partial) function f_{mir} defined on $(\Sigma^*\sharp)^\omega$ by $f(u_1\sharp u_2\sharp\dots) = \overline{u_1}\sharp\overline{u_2}\sharp\dots$. It is computable by a machine that stores its input u_1 in memory until the first $\sharp$ is read, then outputs $\overline{u_1}$, and proceeds with u_2, and so on. It is however not uniformly computable. Indeed, if it were, with some $m\colon \mathbb{N} \to \mathbb{N}$, then, for inputs $u_1\sharp u_2\sharp\cdots$ such that $|u_1| > m(1)$, it is impossible to determine the first output symbol (which is the last of u_1) by reading only a prefix of length $m(1)$ of u_1.

Finally, consider the (partial) function f_{dbl} defined on $(\Sigma^*\sharp)^\omega$ by $f(u_1\sharp u_2\sharp\dots) = u_1u_1\sharp u_2u_2\sharp\dots$. Similarly as before, it is computable but also uniformly computable: to determine the ith output symbol, it suffices to read an input prefix of length at most i. Indeed, let $u_1\sharp u_2\sharp\dots\sharp u$ be a prefix of length i of the input x, then $u_1u_1\sharp u_2u_2\sharp\dots\sharp u$ is a prefix of $f(x)$ of length $\geq i$.

Computability and continuity. There are strong connections between computability and continuity: computable functions are continuous for the Cantor topology, that is where words are close to each other if they share a long common prefix. Intuitively, it is because the very definition of continuity asks that input words sharing longer and longer prefixes also share longer and longer output prefixes. It is the case of the functions f_{mir} and f_{dbl} seen before. Likewise, uniformly computable functions are uniformly continuous. The reverse direction does not hold in general: assuming an effective enumeration $M_1, M_2, \dots$ of Turing machines (on finite word inputs), the function f_{halt} defined as $f_{\mathsf{halt}}(a^\omega) = b_1b_2b_3\dots$ where $b_i \in \{0, 1\}$ is such that $b_i = 1$ iff M_i halts on input ϵ, is not computable but (uniformly) continuous (as it is defined on a single point).

Beyond synchronous functions: regular functions. functional specifications in Church ω-regular synthesis problem range over the class of *synchronous* functions as described before: they can be specified using automata over $\Sigma_i.\Sigma_o$. For example, while S_{swap} and f_∞ are synchronous, f_{mir} and f_{dbl} are not. In this paper, we intend to go much beyond this class by dropping the synchronicity assumption and consider the so-called class of regular functions. It is a well-behaved class, captured by several models such as streaming ω-string

transducers (SST), deterministic two-way Muller transducers with look around ($2\mathsf{DMT}_{\mathsf{la}}$), and also by MSO-transducers [2, Thm. 1, Prop. 1]. We propose the model of deterministic two-way transducers with a prophetic Büchi look-ahead ($2\mathsf{DFT}_{\mathsf{pla}}$) and show that they are equivalent to $2\mathsf{DMT}_{\mathsf{la}}$. This kind of transducer is defined by a *deterministic* two-way automaton without accepting states, extended with output words on the transitions, and which can consult another automaton, called the look-ahead automaton, to check whether an infinite suffix satisfies some regular property. We assume this automaton to be a prophetic Büchi automaton [7, Sec. 7], because this class is naturally suited to implement a regular look-ahead, while capturing all regular languages of infinite words. Look-ahead is necessary to capture functions such as f_∞. Two-wayness is needed to capture, for instance, functions f_{dbl} and f_{mir}.

Contributions. We call *effectively reg-preserving functions* those functions that effectively preserve regular languages by inverse image [24, 19, 23, 20]. This includes for instance rational functions, regular functions and the more general class of polyregular functions [4, 11]. We first show that for effectively reg-preserving functions, computability and continuity coincide, respectively, uniform computability and uniform continuity (Section 3, Theorem 6). To the best of our knowledge, this connection was not made before. The connection is effective, in the sense that when f is effectively reg-preserving and continuous (resp. uniformly continuous), we can effectively construct a Turing machine computing f (resp. uniformly computing f).

For rational functions (functions defined by non-deterministic one-way Büchi transducers), we show that continuity and uniform continuity are decidable in NLogSpace (Section 5, Theorem 12). Continuity and uniform continuity for rational functions were already known to be decidable in PTime, from Prieur [21, Prop. 4]. However, Prieur's proof techniques do not transfer to the two-way case. We then prove that continuity (and hence computability) is decidable for regular functions given by deterministic Büchi two-way transducers with look-ahead (Section 5, Theorem 16). Using our techniques, we also get the decidability of uniform continuity for regular functions (also Theorem 16). Our proof technique relies on a characterisation of non-continuous reg-preserving functions by the existence of pairs of sequences of words which have a nice regular structure (Section 4, Corollary 10). Based on this, we derive a decision procedure for continuity of rational functions by checking in NLogSpace a structural transducer pattern. For regular functions, we rely on a characterisation of the form of output words produced by idempotent loops in two-way transducers [3]. *Most of the proofs have been sketched and the full proofs can be found in full version [10].*

Related work. To the best of our knowledge, our results are new and the notion of continuity has not been extensively studied in the transducers literature over infinite words. The work by Prieur [21] is the closest to ours, while [8] looks at continuity of regular functions encoded by ω-automata.

Notions of continuity with respect to language varieties have been studied for rational functions of *finite words* in [5]. Our notion of uniform continuity can be linked to continuity with respect to a particular language variety which was *not* studied in [5] (namely the non-erasing variety generated by languages of the shape uA^*). A quite strong Lipschitz continuity notion, called *bounded variation* due to Choffrut (*e.g.* [9]), was shown to capture, over finite words, the sequential functions (the corresponding topology is however trivial, hence simple continuity is not very interesting in this context).

Another result connecting computability and continuity is from [6] where the authors find that some notion of computability by AC^0 circuits corresponds, over sequential functions, to continuity with respect to some language variety.

Our result on rational functions has been extended recently to rational functions of infinite words over an infinite alphabet in [12]. More precisely, continuity for functions of infinite data words defined by (one-way) transducers with registers has been shown to be decidable. The proof of [12] goes by reduction to the finite alphabet setting and uses the result presented in this paper, which is publicly available on Arxiv [10], to decide continuity.

Finally, a discussion and comparison of Church ω-synthesis with our work is given in the conclusion of this paper.

2 Languages, Automata and Transducers over ω-Words

Given a finite set Σ, we denote by Σ^* (resp. Σ^ω) the set of finite (resp. infinite) words over Σ, and by Σ^∞ the set of finite and infinite words. Let Σ^j represent the set of all words over Σ with length j. We denote by $|u| \in \mathbb{N} \cup \{\infty\}$ the length of $u \in \Sigma^\infty$ (in particular $|u| = \infty$ if $u \in \Sigma^\omega$). For a word $w = a_1a_2a_3\ldots$, $w[:j]$ denotes the prefix $a_1a_2\ldots a_j$ of w. Let $w[j]$ denote a_j, the j^{th} symbol of w and $w[j:]$ denote the suffix $a_{j+1}a_{j+2}\ldots$ of w. For a word w and $i \leq j$, $w[i:j]$ denotes the factor of w with positions from i to j, both included. For two words $u, v \in \Sigma^\infty$, $u \preceq v$ (resp. $u \prec v$) denotes that u is a prefix (resp. strict prefix) of v (in particular if $u, v \in \Sigma^\omega$, $u \preceq v$ iff $u = v$). For $u \in \Sigma^*$, let $\uparrow u$ denote the set of words $w \in \Sigma^\infty$ having u as prefix *i.e.* $u \preceq w$. Let $\mathsf{mismatch}$ be a function which takes two words, and returns a boolean value, denoted by $\mathsf{mismatch}(u, v)$ for u and v; it returns true if there exists a position $i \leq |u|, |v|$ such that $u[i] \neq v[i]$, and returns false otherwise. The longest common prefix between two words u and v is denoted by $u \wedge v$ and their distance is defined as $d(u, v) = 0$ if $u = v$, and $2^{-|u\wedge v|}$ if $u \neq v$.

A Büchi automaton is a tuple $B = (Q, \Sigma, \delta, Q_0, F)$ consisting of a finite set of states Q, a finite alphabet Σ, a set $Q_0 \subseteq Q$ of initial states, a set $F \subseteq Q$ of accepting states, and a transition relation $\delta \subseteq Q \times \Sigma \times Q$. A run ρ on a word $w = a_1a_2\ldots \in \Sigma^\omega$ starting in a state q_1 in B is an infinite sequence $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \ldots$ such that $(q_i, a_i, q_{i+1}) \in \delta$ for all $i \in \mathbb{N}$. Let $\mathsf{Inf}(\rho)$ denote the set of states visited infinitely often along ρ. The run ρ is a final run iff $\mathsf{Inf}(\rho) \cap F \neq \emptyset$. A run is *accepting* if it is final and starts from an initial state. A word $w \in \Sigma^\omega$ is accepted ($w \in L(B)$) iff it has an accepting run. A language L of ω-words is called *ω-regular* if $L = L(B)$ for some Büchi automaton B.

An automaton is co-deterministic if any two final runs on any word w are the same [7, Sec. 7.1]. Likewise, an automaton is co-complete if every word has at least one final run. A prophetic automaton $P = (Q_P, \Sigma, \delta_P, Q_0, F_P)$ is a Büchi automaton which is co-deterministic and co-complete. Equivalently, a Büchi automaton is prophetic iff each word admits a unique final run. The states of the prophetic automaton partition Σ^ω: each state q defines a set of words w such that w has a final run starting from q. For any state q, let $L(P, q)$ be the set of words having a final run starting at q. Then $\Sigma^\omega = \uplus_{q\in Q_P} L(P, q)$. It is known [7, Thm. 7.2] that prophetic automata capture ω-regular languages.

Transducers. We recall the definitions of one-way and two-way transducers over infinite words. A one-way transducer $\mathcal{A}$ is a tuple $(Q, \Sigma, \Gamma, \delta, Q_0, F)$ where Q is a finite set of states, Q_0, F respectively are sets of initial and accepting states; Σ, Γ respectively are the input and output alphabets; $\delta \subseteq (Q \times \Sigma \times Q \times \Gamma^*)$ is the transition relation. We equip $\mathcal{A}$ with a Büchi acceptance condition. A transition in δ of the form (q, a, q', γ) represents that from state q, on reading a symbol a, the transducer moves to state q', producing the output γ. Runs, final runs and accepting runs are defined exactly as in Büchi automata, with the addition that each transition produces some output $\in \Gamma^*$.

The output produced by a run ρ, denoted $\mathsf{out}(\rho)$, is obtained by concatenating the outputs generated by transitions along ρ. Let $\mathsf{dom}(\mathcal{A})$ represent the language accepted by the underlying automaton of $\mathcal{A}$, ignoring the outputs. The relation computed by $\mathcal{A}$ is defined as $[\![\mathcal{A}]\!] = \{(u, v) \in \Sigma^\omega \times \Gamma^\omega \mid u \in \mathsf{dom}(\mathcal{A}), \rho \text{ is an accepting run of } u, \mathsf{out}(\rho) = v\}$.[3] We say that $\mathcal{A}$ is functional if $[\![\mathcal{A}]\!]$ is a function. A relation (function) is *rational* iff it is recognised by a one-way (functional) transducer.

Two-way transducers extend one-way transducers and two-way finite state automata. A two-way transducer is a two-way automaton with outputs. In [2, Prop. 1], regular functions are shown to be those definable by a two-way deterministic transducer with Muller acceptance condition, along with a regular look-around ($2\mathsf{DMT}_{\mathsf{la}}$). In this paper, we propose an alternative machine model for regular functions, namely, $2\mathsf{DFT}_{\mathsf{pla}}$. A $2\mathsf{DFT}_{\mathsf{pla}}$ is a deterministic two-way automaton with outputs, along with a look-ahead given by a prophetic automaton.

Let $\Sigma_\vdash = \Sigma \uplus \{\vdash\}$. Formally, a $2\mathsf{DFT}_{\mathsf{pla}}$ is a pair $(\mathcal{T}, A)$ where $A = (Q_A, \Sigma, \delta_A, S_A, F_A)$ is a prophetic Büchi automaton and $\mathcal{T} = (Q, \Sigma, \Gamma, \delta, q_0)$ is a two-way transducer s.t. Σ and Γ are finite input and output alphabets, Q is a finite set of states, $q_0 \in Q$ is a unique initial state, $\delta \colon Q \times \Sigma_\vdash \times Q_A \to Q \times \Gamma^* \times \{-1, +1\}$ is a partial transition function. $\mathcal{T}$ has no acceptance condition: every infinite run in $\mathcal{T}$ is a final run. A two-way transducer stores its input $\vdash a_1 a_2 \dots$ on a two-way tape, and each index of the input can be read multiple times. A configuration of a two-way transducer is a tuple $(q, i) \in Q \times \mathbb{N}$ where $q \in Q$ is a state and $i \in \mathbb{N}$ is the current position on the input tape. The position is an integer representing the gap between consecutive symbols. Thus, before $\vdash$, the position is 0, between $\vdash$ and a_1, the position is 1, between a_i and a_{i+1}, the position is $i + 1$ and so on. The $2\mathsf{DFT}_{\mathsf{pla}}$ is deterministic: for every word $w = \vdash a_1 a_2 a_3 \dots \in \vdash\Sigma^\omega$, every input position $i \in \mathbb{N}$, and state $q \in Q$, there is a unique state $p \in Q_A$ such that $a_i a_{i+1} \dots \in L(A, p)$. Given $w = a_1 a_2 \dots$, from a configuration (q, i), on a transition $\delta(q, a_i, p) = (q', \gamma, d)$, $d \in \{-1, +1\}$, such that $a_i a_{i+1} \dots \in L(A, p)$, we obtain the configuration $(q', i + d)$ and the output γ is appended to the output produced so far. This transition is denoted as $(q, i) \xrightarrow{a_i, p/\gamma} (q', i + d)$. A run ρ of a $2\mathsf{DFT}_{\mathsf{pla}}$ $(\mathcal{T}, A)$ is a sequence of transitions $(q_0, i_0 = 0) \xrightarrow{a_{i_0}, p_1/\gamma_1} (q_1, i_1) \xrightarrow{a_{i_1}, p_2/\gamma_2} \cdots$. The output of ρ, denoted $\mathsf{out}(\rho)$ is then $\gamma_1 \gamma_2 \cdots$. The run ρ reads the whole word w if $\mathsf{sup}\{i_n \mid 0 \le n < |\rho|\} = \infty$. The output $[\![(\mathcal{T}, A)]\!](w)$ of a word w on run ρ is defined only when $\mathsf{sup}\{i_n \mid 0 \le n < |\rho|\} = \infty$, and equals $\mathsf{out}(\rho)$. $2\mathsf{DFT}_{\mathsf{pla}}$ are equivalent to $2\mathsf{DMT}_{\mathsf{la}}$, and capture all regular functions (see full version [10] for the proof).

▶ **Theorem 1.** *A function $f \colon \Sigma^\omega \to \Gamma^\omega$ is regular iff it is $2\mathsf{DFT}_{\mathsf{pla}}$ definable.*

▶ **Example 2.** Consider the function $g \colon \Sigma^\omega \to \Gamma^\omega$ over $\Sigma = \Gamma = \{a, b\}$ such that $g(uab^\omega) = uub^\omega$ for $u \in \Sigma^*$ and $g(b^\omega) = b^\omega$. The $2\mathsf{DFT}_{\mathsf{pla}}$ is shown in Figure 1 with the prophetic look-ahead automaton A on the right. The transitions are decorated as $\alpha, p \mid \gamma, d$ where $\alpha \in \{a, b\}$, p is a state of A, γ is the output and d is the direction. In transitions not using the look-ahead information, the decoration is simply $\alpha \mid \gamma, d$. Notice that $\mathcal{L}(A, p_1) = \vdash\Sigma^* ab^\omega$, $\mathcal{L}(A, p_2) = \vdash b^\omega$, $\mathcal{L}(A, p_3) = \Sigma^+ ab^\omega$, $\mathcal{L}(A, p_4) = ab^\omega$. Each word in $\vdash\Sigma^\omega$ has a unique final run; $\mathcal{L}(A, p_1) \cup \mathcal{L}(A, p_2) = \vdash\mathsf{dom}(g)$. The remaining states ensure that each word in $(\vdash\Sigma^\omega \setminus \vdash\mathsf{dom}(g)) \uplus \Sigma^\omega$ has a unique final run. ⌟

[3] We assume that final runs always produce infinite words, which can be enforced syntactically by a Büchi condition such that any input word produces non-empty output in a single loop execution containing Büchi accepting state.

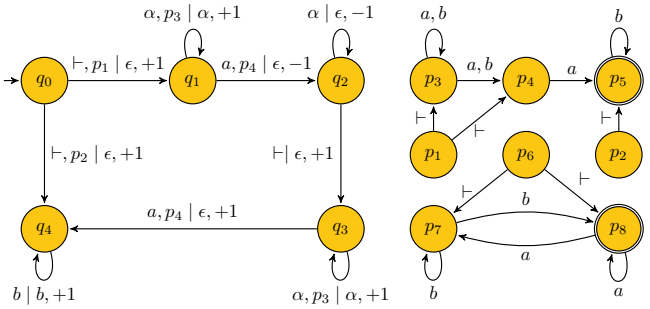


Figure 1 A $2\mathsf{DFT}_{\mathsf{pla}}$ with automaton on the right implementing the look-ahead.

We also use a look-ahead-free version of two-way transducers in some of the proofs, where we also have a Büchi acceptance condition given by a set of states F, just as for Büchi automata. The resulting model is called two-way deterministic Büchi transducer (2DBT). The definitions of configuration, run, and the semantics are done just like for $2\mathsf{DFT}_{\mathsf{pla}}$.

3 Computability versus Continuity

Computability of a function on infinite words can be described intuitively in the following way: there is an algorithm which, given access to the input word, can enumerate the letters in the output word. We also investigate a stronger notion of computability, which we call *uniform computability*. The main idea is that given some input word x and some position j one can compute the j^{th} position of the output in time that depends on j but not on x. An appealing aspect of uniform computability is that it offers a uniform bound on the number of input symbols one needs to read in order to produce the output at some fixed precision.

▶ **Definition 3** (Computability/Uniform computability). *A function $f\colon \Sigma^\omega \to \Gamma^\omega$ is* computable *if there exists a deterministic multitape Turing machine M computing it in the following sense. The machine M has a read-only one-way input tape, a two-way working tape, and a write-only one-way output tape. All tapes have a left delimiter $\vdash$ and are infinite to the right. Let $x \in \mathsf{dom}(f)$. For any $j \in \mathbb{N}$, let $M(x, j)$ denote the output produced by M till the time it moves to the right of position j, onto position $j+1$ in the input (or ϵ if this move never happens). The function f is computable by M if for all $x \in \mathsf{dom}(f)$, for all $i \geq 0$, there exists $j \geq 0$ such that $f(x)[:i] \preceq M(x, j)$.*

Moreover if there exists a computable function $m\colon \mathbb{N} \to \mathbb{N}$ (called a modulus of continuity *for M) such that for all $x \in \mathsf{dom}(f)$, for all $i \geq 0$, $f(x)[:i] \preceq M(x, m(i))$, f is called* uniformly computable.

It turns out that there is a quite strong connection between computability and continuity of functions. In particular computable functions are always continuous. This can be seen intuitively since given a deterministic Turing machine, it must behave the same on the common prefixes of two words. Hence two words with a very long common prefix must have images by the machine that have a somewhat long common prefix. This connection also transfers to uniform computability and uniform continuity. We start by formally defining continuity and uniform continuity. We interchangeably use the following two definitions [22] of continuity.

▶ **Definition 4** (Continuity/Uniform continuity)**.**

1. *A function* $f\colon \Sigma^\omega \to \Gamma^\omega$ *is continuous at* $x \in \mathsf{dom}(f)$ *if (equivalently)*
   - **(a)** *for all* $(x_n)_{n\in\mathbb{N}}$ *converging to* x*, where* $x_i \in \mathsf{dom}(f)$ *for all* $i \in \mathbb{N}$*,* $(f(x_n))_{n\in\mathbb{N}}$ *converges.*
   - **(b)** $\forall i \geq 0\ \exists j \geq 0\ \forall y \in \mathsf{dom}(f),\ |x \wedge y| \geq j \Rightarrow |f(x) \wedge f(y)| \geq i$
2. *A function is continuous if it is continuous at every* $x \in \mathsf{dom}(f)$*.*
3. *A function* $f\colon \Sigma^\omega \to \Gamma^\omega$ *is uniformly continuous if:*
   there exists $m\colon \mathbb{N} \to \mathbb{N}$*, called a* modulus of continuity *for* f *such that,*
   $\forall i \geq 0,\ \forall x, y \in \mathsf{dom}(f),\ |x \wedge y| \geq m(i) \Rightarrow |f(x) \wedge f(y)| \geq i$*.*

▶ **Example 5.** As explained in the introduction, the function f_∞ is not continuous, and, as we will see later, is thus not computable. The function f_{halt} is continuous, even uniformly continuous (it is constant) yet is obviously not computable. The function f_{mir} is computable, however is not uniformly continuous, two words can be arbitrarily close but with far away outputs: consider $a^n\sharp^\omega$ and $a^n b\sharp^\omega$. Finally, the function f_{dbl} is uniformly computable. ⌟

We now investigate the relationship between continuity and computability for functions that are effectively reg-preserving. More precisely, we say that a function $f\colon \Sigma^\omega \to \Gamma^\omega$ is *effectively reg-preserving* if there is an algorithm which, for any automaton recognizing a regular language $L \subseteq \Gamma^\omega$, produces an automaton recognizing the language $f^{-1}(L) = \{u \mid f(u) \in L\}$.

Two well-studied classes (see *e.g.* [13]) of reg-preserving functions are the rational and the regular functions, which we will study in Section 5. As announced, continuity and computability coincide for effectively reg-preserving functions:

▶ **Theorem 6.** *An effectively reg-preserving function* $f\colon \Sigma^\omega \to \Gamma^\omega$ *is computable (resp. uniformly computable) if and only if it is continuous (resp. uniformly continuous).*

Proof. $\Rightarrow$) This implication is easy and actually holds without the reg-preserving assumption. If f is computable by some machine M, then it is not difficult to see that it is continuous. Intuitively, the longer the prefix of input $x \in \mathsf{dom}(f)$ is processed by M, the longer the output produced by M on that prefix, which converges to $f(x)$, according to the definition of computability. More details of this proof can be found in full version. Moreover, if f is uniformly computable, then the modulus of continuity of M is in particular a modulus of continuity for f and is thus uniformly continuous.

Algorithm 1 Algorithm describing M.

```
   Input: x ∈ Σ^ω
1  out := ϵ ;   ;                          // this is written on the working tape
2  for i = 0 to +∞ do
3  |  for γ ∈ Γ do
4  |  |  if f(↑x[:i]) ⊆ ↑out.γ then
5  |  |  |  out := out.γ ; ;                        // append to the working tape
6  |  |  |  output γ ;   ;              // this is written on the output tape
```

$\Leftarrow$) The converse direction is less trivial and makes use of the reg-preserving assumption. Suppose that f is continuous. We design the machine M, represented as Algorithm 1, which is shown to compute f. This machine processes longer and longer prefixes $x[:i]$ of its input x (for loop at line 2), and tests (line 4) whether a symbol γ can be safely appended to the output. The test ensures that the invariant $\mathsf{out} \preceq f(x)$ is preserved at any point. Moreover, the continuity of f at x ensures that out is updated infinitely often. The only thing left to

obtain computability is that the test of line 4 is decidable. Let u and v be two words, deciding $f(\uparrow u) \subseteq \uparrow v$ is equivalent to deciding if $\mathsf{dom}(f) \cap \uparrow u \subseteq f^{-1}(\uparrow v)$. These sets are effectively regular since u, v are given and f is effectively reg-preserving, and $\mathsf{dom}(f) = f^{-1}(\Gamma^\omega)$. Since the constructions are effective, and the languages are regular, the inclusion is decidable.

We only have left to show that if f is moreover uniformly continuous, then M has a computable modulus of continuity. We start by showing that f has a computable modulus of continuity. Let us consider the predicate $P(i,j)\colon \forall x, y\ |x \wedge y| \geq j \Rightarrow |f(x) \wedge f(y)| \geq i$. Then we define $m\colon i \mapsto \min\{j \mid P(i,j)\}$. Since f is uniformly continuous, m is indeed well defined and is a modulus of continuity of f. To show that m is computable, we only have to show that $P(i,j)$ is decidable.

Let us consider the negation of $P(i,j)$: there exist $u, x_1, x_2, v_1, v_2, w_1, w_2$ such that $|u| = j$, and $f(ux_k) = v_k w_k$ for $k \in \{1,2\}$ with $|v_1| = |v_2| = i$ and $v_1 \neq v_2$. Hence to decide $\neg P(i,j)$, we only have to find two words $v_1 \neq v_2$ in Γ^i, such that $S \neq \emptyset$ where $S = \{vw \mid v \in \Sigma^j,\ \exists w_1, w_2,\ \text{s.t. } vw_1 \in f^{-1}(\uparrow v_1),\ vw_2 \in f^{-1}(\uparrow v_2)\}$. Since f is effectively reg-preserving, S is effectively regular. By searching exhaustively for words $v_1, v_2 \in \Gamma^i$ we get decidability of $P(i,j)$. We only have left to define a modulus of continuity for M. Let $m'\colon \mathbb{N} \to \mathbb{N}$ be defined by $m'(i) = m(i) + i$. If we read $m(i)$ symbols, we know we can output at least i symbols. Hence in each of the next i steps, we are guaranteed to output a letter. Hence m' is a modulus of continuity for M and f is uniformly computable. ◀

▶ Remark 7. Note that we focus on functions that are effectively reg-preserving, but Algorithm 1 is actually more general than that. The continuity-computability equivalence indeed carries over to any class of functions for which the test in line 4 is decidable.

4 A Characterisation of Continuity and Uniform Continuity

We provide here a characterisation of continuity (and uniform continuity) for reg-preserving functions (we don't need effectiveness here). The characterisation is based on a study of some particular properties of sequences and pairs of sequences which we define below:

▶ **Definition 8.** *Let $f\colon \Sigma^\omega \to \Gamma^\omega$.*

Let $(x_n)_{n\in\mathbb{N}}$ be a sequence of words in $\mathsf{dom}(f)$ converging to $x \in \Sigma^\omega$, such that $(f(x_n))_{n\in\mathbb{N}}$ is not convergent. Such a sequence is called a bad sequence *at x for f.*

Let $(x_n)_{n\in\mathbb{N}}$ and $(x'_n)_{n\in\mathbb{N}}$ be two sequences in $\mathsf{dom}(f)$ both converging to $x \in \Sigma^\omega$, such that either $(f(x_n))_{n\in\mathbb{N}}$ is not convergent, $(f(x'_n))_{n\in\mathbb{N}}$ is not convergent, or $\lim_n f(x_n) \neq \lim_n f(x'_n)$. Such a pair of sequences is called a bad pair *of sequences at x for f.*

A pair of sequences is synchronised *if it is of the form: $((uv^n wz^\omega)_n, (uv^n w' z'^\omega)_n)$*

▶ **Proposition 9.** *A function is* not *continuous if and only if it has a bad pair at some point of its domain. A function is* not *uniformly continuous if and only if it has a bad pair.*

Proof. The case of continuous functions is obtained just by definition. For uniform continuity, consider a function f with a bad pair $((x_n)_{n\in\mathbb{N}}, (x'_n)_{n\in\mathbb{N}})$, and let us show that it is not uniformly continuous. We can assume that both $(f(x_n))_{n\in\mathbb{N}}$ and $(f(x'_n))_{n\in\mathbb{N}}$ converge. Otherwise we can extract subsequences that converge, by compactness of Γ^ω. Moreover, since the pair is bad, one can assume that they converge to different limits $y \neq y'$. Let i be such that $y[i] \neq y'[i]$. For any j, one can find N such that for all $n \geq N$, $|x_n \wedge x'_n| \geq j$ since both sequences converge to x. Since $(f(x_n))_{n\in\mathbb{N}}$ converges to y, we can ensure that N is large enough so that for all $n \geq N$, $|f(x_n) \wedge y| \geq i$. We can also ensure that for $n \geq N$, $|f(x'_n) \wedge y'| \geq i$ holds. Let $n \geq N$, we have both $|x_n \wedge x'_n| \geq j$ and $|f(x_n) \wedge f(x'_n)| < i$, which means that f is not uniformly continuous.

Let f be a function which is not uniformly continuous, we want to exhibit a bad pair of f. According to the definition, there exists i such that for all j there exist x_j, x'_j with $|x_j \wedge x'_j| \geq j$ but $|f(x_j) \wedge f(x'_j)| < i$. By compactness of Σ^ω, there exists a subsequence of $(x_j)_{j\in\mathbb{N}}$ which is convergent. Let $\left(x_{\tau(j)}\right)_{j\in\mathbb{N}}$, with $\tau\colon \mathbb{N} \to \mathbb{N}$ increasing, denote such a subsequence. Then we have for all j that $|x_{\tau(j)} \wedge x'_{\tau(j)}| \geq \tau(j) \geq j$ and $|f(x_{\tau(j)}) \wedge f(x'_{\tau(j)})| < i$. Therefore up to renaming the sequences, we can assume that for all j, $|x_j \wedge x'_j| \geq j$ and $|f(x_j) \wedge f(x'_j)| < i$, with $(x_j)_{j\in\mathbb{N}}$ being convergent. By repeating the process of extracting subsequences, we can assume that $\left(x'_j\right)_{j\in\mathbb{N}}$, $(f(x_j))_{j\in\mathbb{N}}$, $\left(f(x'_j)\right)_{j\in\mathbb{N}}$ are also convergent. Since for any j, $|x_j \wedge x'_j| \geq j$, the two sequences converge to the same limit. In the end we obtain that $\left((x_j)_{j\in\mathbb{N}}, \left(x'_j\right)_{j\in\mathbb{N}}\right)$ is a bad pair for f at $\lim_j x_j = \lim_j x'_j$.

Note that in case $\mathsf{dom}(f)$ is not compact, then we may not have a subsequence of $(x_j)_{j\in\mathbb{N}}$ which converges in $\mathsf{dom}(f)$. However, the definition of bad pairs does not require convergence in the domain; it only asks for convergence to some x, which need not be in $\mathsf{dom}(f)$. ◀

The main result of this section is the following lemma which says that one can restrict to considering only *synchronised* bad pairs. In the following sections this characterisation will be used to decide continuity/uniform continuity.

▶ **Lemma 10** (Characterisation). *A reg-preserving function is not continuous if and only if it has a synchronised bad pair at some point of its domain. A reg-preserving function is not uniformly continuous if and only if it has a synchronised bad pair.*

Sketch of Proof. This lemma extends Proposition 9. It shows that, in the case of reg-preserving functions, one can restrict to considering synchronised pairs, which are much easier to deal with. The proof is done in several steps but due to a lack of space, we only sketch these steps, the full proof being given in full version [10].

First we show that for a reg-preserving function f, if there is a bad pair at some x, then there is one at some *regular* z, *i.e.* $z = uv^\omega$ for some finite words u, v. Moreover, for the case of non-uniform continuity, we show that z can be chosen so that $x \in \mathsf{dom}(f) \Leftrightarrow z \in \mathsf{dom}(f)$.

In the second step, since we have two sequences converging to regular z, we show how to replace the bad pair by a bad pair of *regular* sequences, still using the fact that f is reg-preserving. Finally, we prove that we can *synchronise* these two regular sequences and end up with a synchronised bad pair at z. ◀

5 Deciding Continuity and Uniform Continuity

We first show how to decide (uniform) continuity for rational and then for regular functions.

Rational case. We exhibit structural patterns which are shown to be satisfied by a one-way Büchi transducer iff the rational function it defines is not continuous (resp. not uniformly continuous). We express those patterns in the *pattern logic* defined in [15, Sec. 6], which is based on existential run quantifiers of the form $\exists \pi\colon p \xrightarrow{u|v} q$ where π is a run variable, p, q are state variables and u, v are word variables. Intuitively, there exists a run π from state p to state q on input u, producing output v. A one-way transducer is called *trim* if each of its states appears in some accepting run. Any one-way Büchi transducer can be trimmed in polynomial time. The structural patterns for trim transducers are given in Figures 2 and 3. The predicate $\mathsf{init}(p)$ expresses that p is initial while $\mathsf{acc}(p)$ expresses that it is accepting. The predicate $\mathsf{mismatch}$ expresses the existence of a mismatch between two words, as defined in Section 2.

$$\phi_{\text{cont}} = \begin{array}{l} \exists \pi_1 \colon p_1 \xrightarrow{u|u_1} q_1,\ \exists \pi_1' \colon q_1 \xrightarrow{v|v_1} q_1 \\ \exists \pi_2 \colon p_2 \xrightarrow{u|u_2} q_2,\ \exists \pi_2' \colon q_2 \xrightarrow{v|v_2} q_2,\ \exists \pi_2'' \colon q_2 \xrightarrow{w|w_2} r_2 \\ \big(\mathsf{init}(p_1) \wedge \mathsf{init}(p_2) \wedge \mathsf{acc}(q_1)\big) \wedge \\ \big(\mathsf{mismatch}(u_1, u_2) \vee (v_2 = \epsilon \wedge \mathsf{mismatch}(u_1, u_2 w_2))\big) \end{array}$$

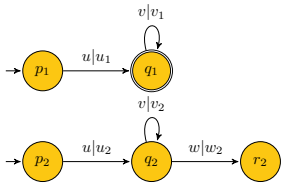


Figure 2 Pattern characterising non-continuity of rational functions given by *trim* one-way Büchi transducers.

$$\phi_{\text{u-cont}} = \begin{array}{l} \exists \pi_1 \colon p_1 \xrightarrow{u|u_1} q_1,\ \exists \pi_1' \colon q_1 \xrightarrow{v|v_1} q_1,\ \exists \pi_1'' \colon q_1 \xrightarrow{w|w_1} r_1 \\ \exists \pi_2 \colon p_2 \xrightarrow{u|u_2} q_2,\ \exists \pi_2' \colon q_2 \xrightarrow{v|v_2} q_2,\ \exists \pi_2'' \colon q_2 \xrightarrow{w|w_2} r_2 \\ \big(\mathsf{init}(p_1) \wedge \mathsf{init}(p_2)\big) \wedge \\ \big(\mathsf{mismatch}(u_1, u_2) \vee (v_1 = \epsilon \wedge \mathsf{mismatch}(u_1 w_1, u_2)) \\ \qquad \vee (v_1 = v_2 = \epsilon \wedge \mathsf{mismatch}(u_1 w_1, u_2 w_2))\big) \end{array}$$

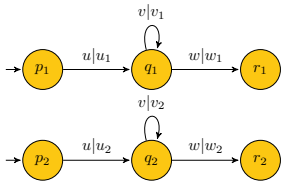


Figure 3 Pattern characterising non-*uniform* continuity of rational functions given by *trim* one-way Büchi transducers.

▶ **Lemma 11.** *A trim one-way Büchi transducer defines a non-continuous (resp. non-uniformly continuous) function if and only if it satisfies the formula ϕ_{cont} of Fig. 2 (resp. the formula $\phi_{u\text{-}cont}$ of Fig. 3).*

Sketch of Proof. Showing that the patterns of Figure 2 and Figure 3 induce non-continuity and non-uniform continuity, respectively, is quite simple. Indeed, the first pattern ϕ_{cont} is a witness that $(uv^n wz)_{n\in\mathbb{N}}$ is a bad sequence at a point uv^ω of its domain, for z a word with a final run from r_2, which entails non-continuity by Proposition 9 (if a sequence s is bad then (s, s) is bad). Similarly, the pattern $\phi_{\text{u-cont}}$ witnesses that the pair $\left((uv^n wz)_{n\in\mathbb{N}}, (uv^n w' z')_{n\in\mathbb{N}}\right)$ is synchronised and bad (with z, z' words that have a final run from r_1, r_2, respectively), which entails non-uniform continuity by Lemma 10.

For the other direction, we again use Lemma 10. From a synchronised bad pair, we can find a pair of runs with a synchronised loop, such that iterating the loop does not affect the existing mismatch between the outputs of the two runs, which is in essence what the pattern formulas of Figure 2 and Figure 3 state. The full proof is available in full version [10]. ◀

▶ **Theorem 12.** *Deciding if a one way Büchi transducer defines a continuous (resp. uniformly continuous) function can be done in NLogSpace.*

Proof. Let T be a one way Büchi transducer defining a function f. From Lemma 11, if T is trim, non-continuity of f is equivalent to T satisfying the formula ϕ_{cont} of Fig. 2. This formula is expressed in the syntax of the pattern logic from [15], where it is proved that model-checking pattern formulas against transducers can be done in NLogSpace [15, Thm. 6]. This yields the result.

If T is not trim, then we modify the formula ϕ_{cont} to additionally express that there must be some accepting run from r_2 on some input. Equivalently, we express that there exists a run from r_2 to some accepting state s, and a run looping in s, in the following way: we just add the quantifiers $\exists \pi_3 \colon r_2 \xrightarrow{\alpha|\beta} s\ \exists \pi_4 \colon s \xrightarrow{\gamma|\tau} s$ to ϕ_{cont} and the constraint $\mathsf{acc}(s)$ which requires s to be accepting.

The proof for non-uniform continuity, using formula $\phi_{\text{u-cont}}$ is similar. ◀

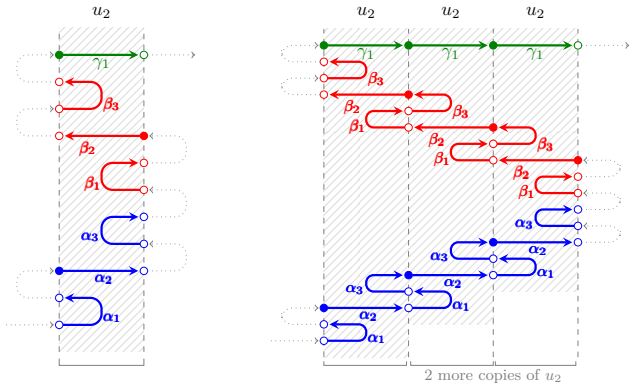


Figure 4 Pumping u_2. $\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3, \gamma_1$ are the outputs seen on u_2. The blue, red and green arrows are part of the run r while reading u_2.

Regular case. The case of regular functions is more intricate. We have to exploit the form of the output words produced by idempotent loops of two-way transducer runs. Idempotent loops always exist for sufficiently long inputs and indeed have a nice structure which allows one to characterise the form of the output words produced when iterating such loops [3]. A detailed definition of idempotent loops, based on the traversal monoid is in [3]. We have abstracted the main property of idempotent loops which is sufficient in our context, and for which it is not necessary to know the precise definition of idempotency. So, given a deterministic two-way transducer T on finite words (we need the notion only for finite words) and an input word $u_1u_2u_3$, we will say that u_2 is idempotent in (u_1, u_2, u_3) (or just idempotent when u_1, u_3 are clear from the context), if in the run r of T on $u_1u_2u_3$, the restriction of r to u_2 (which is a sequence of possibly disconnected runs on u_2) is idempotent *i.e.* we can pump u_2 any number of times in the context of u_1 and u_3 and still get a valid run of T [3]. See Figure 4, if the sequence of states visited before reading first position of u_2 (at first vertical dashed line in figure) and the sequence of states visited after reading u_2 (at second vertical dashed line), we can pump u_2, in other words concatenate the run shown in left to itself multiple times. In right side figure, the partial run is shown where u_2 is concatenated with itself twice. Observe that the outputs on the factors that is shown on each blue, red and green arrow remain same and the output words are concatenated in pumped word in a systematic manner.

Given a language of ω-words $L \subseteq \Sigma^\omega$, we denote by $\mathrm{Pref}(L)$ the set of finite prefixes of words in L, *i.e.* $\mathrm{Pref}(L) = \{u \in \Sigma^* \mid \exists v \in L, \text{ with } u \preceq v\}$. In order to deal with look-aheads more easily, we remove look-aheads by considering words annotated with look-ahead information. Given a $\mathsf{2DFT_{pla}}$ $(\mathcal{T}, P)$ over alphabet Σ and with a set of look-ahead states Q_P, realising a function f, we define $\widetilde{\mathcal{T}}$, a $\mathsf{2DBT}$ over $\Sigma \times Q_P$ which simulates $(\mathcal{T}, P)$ over words annotated with look-ahead states, and which accepts only words with a correct look-ahead annotation with respect to P (the formal definition can be found in full version [10]). We denote by $\tilde{f}$ the function it realises, in particular for all words $u \in \mathsf{dom}(f)$, there exists a unique annotated word $\tilde{u} \in \mathsf{dom}(\tilde{f})$ such that $\tilde{f}(\tilde{u}) = f(u)$, as P is prophetic. For any annotated word $\tilde{u}$, $\pi(\tilde{u}) = u$ stands for its Σ-projection.

From $\widetilde{\mathcal{T}}$, we define $\mathcal{T}_*$, a deterministic two-way transducer of finite words over the input alphabet $\Sigma \times Q_P$. Its domain is restricted to $\mathrm{Pref}(\mathsf{dom}(\tilde{f}))$ (which is a regular set) and it behaves just as $\widetilde{\mathcal{T}}$ until it reaches the right border of its input for the first time, after which it

accepts iff the input was indeed a prefix of $\mathsf{dom}(\tilde{f})$ which, as said before, is a regular property which can be checked by $\mathcal{T}_*$ while simulating $\widetilde{\mathcal{T}}$. We let f_* be the function realised by $\mathcal{T}_*$ (which depends on $\mathcal{T}$). We have that, for any infinite word $x \in \mathsf{dom}(\tilde{f})$, $\tilde{f}(x) = \lim_{u \prec x} f_*(u)$.

The following lemma is a first characterisation of non-continuity which we can get by exploiting the existence of synchronised bad pairs.

▶ **Lemma 13.** *Let $f\colon \Sigma^\omega \to \Gamma^\omega$ be a regular function defined by some deterministic two-way transducer $\mathcal{T}$ with look-ahead and let Q_P be the set of look-ahead states. Then f is not continuous (resp. uniformly continuous) iff there exist finite words $u_1, u_1', u_2, u_2', u_3, u_3' \in (\Sigma \times Q_P)^*$ such that $u_1u_2u_3, u_1'u_2'u_3' \in \mathsf{dom}(f_*)$ and*

1. *$\pi(u_1) = \pi(u_1')$, $\pi(u_2) = \pi(u_2')$, and $x = \pi(u_1)\pi(u_2)^\omega \in \mathsf{dom}(f)$ (resp. $x \in \Sigma^\omega$),*
2. *u_2 and u_2' are idempotent in (u_1, u_2, u_3) and (u_1', u_2', u_3') respectively (for $\mathcal{T}_*$),*
3. *there exists i such that for all $n \geq 1$, $f_*(u_1u_2^nu_3)[i] \neq f_*(u_1'{u_2'}^nu_3')[i]$.*

Sketch of Proof. For the if direction, since $\mathsf{dom}(f_*) = \mathrm{Pref}(\mathsf{dom}(\tilde{f}))$, for any n there are some $u_{4,n}, u_{4,n}'$ such that the words $x_n = u_1u_2^nu_{4,n}$ and $x_n' = u_1'{u_2'}^nu_{4,n}'$ are both in $\mathsf{dom}(\tilde{f})$. Moreover, the sequences $(\pi(x_n))_{n\in\mathbb{N}}$ and $(\pi(x_n'))_{n\in\mathbb{N}}$ both converge to x. However, since there is a mismatch between $f_*(u_1u_2^nu_3)$ and $f_*(u_1'{u_2'}^nu_3')$ at position i, and by definition of f_* we have $f_*(u_1u_2^nu_3) \preceq \tilde{f}(x_n)$ and $f_*(u_1'{u_2'}^nu_3') \preceq \tilde{f}(x_n')$, there is also one between $\tilde{f}(x_n)$ and $\tilde{f}(x_n')$ at position i. Thus the pair $\left((\pi(x_n))_{n\in\mathbb{N}}, (\pi(x_n'))_{n\in\mathbb{N}}\right)$ is a bad pair and we conclude by Lemma 10.

In the other direction, as for the rational case, we start from Lemma 10 stating that it suffices to check for a synchronised bad pair. Like in the rational case, we successively extract subsequences of the synchronised bad pair and at each step we need to preserve synchronicity as well as badness. The main idea is that if we iterate enough times the loop in the synchronised bad pair, we will end up with synchronised idempotent loops. The more detailed version is available in [10]. ◀

Given a deterministic two-way transducer T (*i.e.* with a trivial look-ahead) defining a function f and words $u_1, u_2, u_3 \in \Sigma^*$ such that $u_1u_2u_3 \in \mathrm{Pref}(\mathsf{dom}(T))$ and u_2 is idempotent for T, we say that u_2 is "producing" in (u_1, u_2, u_3) if the run of T on $u_1u_2u_3$ produces some output when reading at least one symbol of u_2, at some point in the run. If u_2 is producing, then $|f_*(u_1u_2^iu_3)| < |f_*(u_1u_2^{i+1}u_3)|$ for all $i \geq 1$.

Our goal is now to give another characterisation of (non-) continuity, which replaces the quantification on n in Lemma 13 by a property which does not need iteration, and therefore which is more amenable to an algorithmic check. It is based on the following key result.

▶ **Lemma 14.** *Let Σ be an alphabet such that $\# \notin \Sigma$. Let $g\colon \Sigma^\omega \to \Gamma^\omega$ be a regular function defined by some deterministic two-way transducer U. There exists a function $\rho_U\colon (\Sigma^*)^3 \to \Gamma^*$ defined on all tuples (u_1, u_2, u_3) such that u_2 is idempotent and $u_1u_2u_3 \in \mathsf{Pref}(\mathsf{dom}(g))$, and which satisfies the following conditions:*

1. *if u_2 is producing in (u_1, u_2, u_3), then $\rho_U(u_1, u_2, u_3) \prec \rho_U(u_1u_2, u_2, u_2u_3)$*
2. *for all $n \geq 1$, $\rho_U(u_1, u_2, u_3) \preceq g_*(u_1u_2^nu_3)$*
3. *for all $n \geq 1$, $\rho_U(u_1, u_2, u_3) = g_*(u_1u_2^nu_3)$ if u_2 is* not *producing in (u_1, u_2, u_3)*
4. *the finite word function $\rho_U'\colon u_1\#u_2\#u_3 \mapsto \rho_U(u_1, u_2, u_3)$ is (effectively) regular.*

Proof. The proof of Lemma 14 is based on a thorough study of the form of the output words produced by idempotent loops [3]. The whole proof, which requires technical notions, can be found in full version [10]. ◀

Note that the previous lemma is stated for transducers without look-ahead. It is however sufficient as we apply it to transducers of the form $\widetilde{\mathcal{T}}$. In particular, we use this lemma to characterise the continuity of a function defined by a $2\mathsf{DFT}_{\mathsf{pla}}$ $\mathcal{T}$ by using the function $\rho_{\widetilde{\mathcal{T}}}$.

Unlike in Lemma 13, in the following characterisation, we do not need to iterate the loop to check existence of a mismatch for all iterations, as we just need to inspect $\rho_{\widetilde{\mathcal{T}}}(u_1, u_2, u_3)$.

▶ **Lemma 15.** *Let $f\colon \Sigma^\omega \to \Gamma^\omega$ be a function defined by some deterministic two-way transducer $\mathcal{T}$ with look-ahead and let Q_P be the set of look-ahead states. The function f is not continuous (resp. not uniformly continuous) iff there exist $u_1, u_1', u_2, u_2', u_3, u_3' \in (\Sigma \times Q_P)^*$ such that $u_1u_2u_3, u_1'u_2'u_3' \in \mathsf{dom}(f_*)$ and*

1. *$\pi(u_1) = \pi(u_1')$, $\pi(u_2) = \pi(u_2')$, and $x = \pi(u_1)\pi(u_2)^\omega \in \mathsf{dom}(f)$ (resp. $x \in \Sigma^\omega$)*
2. *u_2 and u_2' are idempotent in (u_1, u_2, u_3) and (u_1', u_2', u_3') respectively (for $\widetilde{\mathcal{T}}$)*
3. *there is a mismatch between $\rho_{\widetilde{\mathcal{T}}}(u_1, u_2, u_3)$ and $\rho_{\widetilde{\mathcal{T}}}(u_1', u_2', u_3')$.*

Sketch of proof. We show how to replace condition 3 of Lemma 13 by condition 3 of this lemma. One direction is easy: if $\rho_{\widetilde{\mathcal{T}}}(u_1, u_2, u_3)[i] \neq \rho_{\widetilde{\mathcal{T}}}(u_1', u_2', u_3')[i]$ for some i, then by Condition 2 of Lemma 14, we get the result. Conversely, assume there is i such that $f_*(u_1u_2^nu_3)[i] \neq f_*(u_1'(u_2')^nu_3')[i]$ for all $n \geq 1$ and u_2, u_2' are both producing (the other cases are similar and done in full version). By Condition 1 of Lemma 14, $\rho_{\widetilde{\mathcal{T}}}(u_1, u_2, u_3) \prec \rho_{\widetilde{\mathcal{T}}}(u_1u_2, u_2, u_2u_3) \prec \cdots \prec \rho_{\widetilde{\mathcal{T}}}(u_1u_2^k, u_2, u_2^ku_3)$ for all $k \geq 1$, and similarly for the u_i'. Therefore, for large enough k, $\rho_{\widetilde{\mathcal{T}}}(u_1u_2^k, u_2, u_2^ku_3)$ and $\rho_{\widetilde{\mathcal{T}}}({u_1'u_2'}^k, u_2', {u_2'}^ku_3')$, have length at least i. By Condition 2, $x = \rho_{\widetilde{\mathcal{T}}}(u_1u_2^k, u_2, u_2^ku_3) \preceq f_*(u_1u_2^nu_3)$ and $x' = \rho_{\widetilde{\mathcal{T}}}({u_1'u_2'}^k, u_2', {u_2'}^ku_3') \preceq f_*({u_1'u_2'}^nu_3')$ for all $n \geq 2k+1$, from which we get $x[i] \neq x'[i]$. ◀

Finally, we show how to decide continuity by reduction to the emptiness problem of bounded-visit two-way Parikh automata [17, 14]. (Full details are in full version [10])

▶ **Theorem 16.** *Continuity and uniform continuity are decidable for regular functions.*

Sketch of proof. The proof is based on Lemma 15. First, we encode words u_1, u_1', u_2, u_2' as words over the alphabet $(\Sigma \times {Q_P}^2)$ to hard-code condition 1 of the lemma. In particular, we define the language L of words of the form $w_1\#w_2\#u_3\#u_3'$ such that $w_1, w_2 \in (\Sigma \times {Q_P}^2)^*$ represent u_1, u_1', u_2, u_2' and such that conditions 1 and 2 of the lemma are satisfied. Condition 2 and condition $\pi(u_1)\pi(u_2)^\omega \in \mathsf{dom}(f)$ are simple because they are regular properties of words, the domain of f being regular. For condition 3, we need counters to identify positions i and j such that $\rho_{\widetilde{\mathcal{T}}}(u_1, u_2, u_3)[i] \neq \rho_{\widetilde{\mathcal{T}}}(u_1, u_2, u_3)[j]$, and later on check that $i = j$. In particular, we rely on the model of two-way Parikh automata which extend two-way automata with counters which can be only incremented and tested at the end of the computation. If such automata visit any input position a bounded number of times, their emptiness is decidable [17, 14]. We show that L is definable by an automaton which (1) visits any input boundedly many times, and (2) simulates the transducer obtained by Lemma 14.4. ◀

6 Discussion and Further Directions

Summary. In this paper, we have studied two notions of computability for rational and regular functions, shown their correspondences to continuity notions which we proved to be decidable. The notion of uniform computability asks for the existence of a modulus of continuity, which tells how far one has to go in the input to produce a certain amount of output. It would be interesting to give a tight upper bound on modulus of continuity for regular functions, and we conjecture that it is always a linear (affine) function in that case.

Discussion on Church synthesis. This work is motivated by a synthesis problem: given a specification of a function of infinite words (as a transducer), does there exist an algorithm to compute it and if true, synthesize such an algorithm. We have established in the introduction that even in the setting of Church ω-regular synthesis, this question makes sense as some (functional and synchronous) ω-regular specifications may describe functions which are not even computable. Here we compare our work with Church synthesis and address some open question. The Church ω-regular synthesis problem is known be decidable [18]. The setting we consider in this paper is orthogonal: Church ω-regular synthesis considers *non*-functional specifications but they have to be synchronous, while we consider functional specifications but they can be represented by way more expressive automata devices (two-way transducers with look-ahead). Moreover, Church synthesis asks for computability by Mealy machines while our goal is to relax this notion to more general computability notions. A corollary of our results is that the Church ω-regular synthesis problem when the specification is functional and the function realising the specification is only required to be computable, can be decided in NLogSpace. An interesting open question that we do not solve here is the extension of this latter result to non-functional specifications. More precisely, we leave the following problem open: given an automaton over $\Sigma_i.\Sigma_o$ defining an ω-regular synchronous specification S, is S realisable by a computable function? This question was partially answered in [16], where S is assumed to be *total*, *i.e.* $\mathsf{dom}(S) = \Sigma_i^\omega$. Intuitively, it is shown that in this case, if S is realisable, then it is realisable by a bounded delay function, *i.e.* a function which can be implemented by a deterministic transducer which needs to read at most $i + K$ input symbols before outputting the ith output symbol, where K is a constant that depends only on f and not on the input. The open case where S is partial is more challenging. For example, the function S_{swap} has partial domain, is computable, but not bounded delay computable.

Other future directions. Another interesting direction is to find a transducer model which captures exactly the computable, and uniformly computable, rational and regular functions. For rational functions, the deterministic (one-way) transducers are not sufficient, already for uniform computability, as witnessed by the rational function which maps any word of the form $a^n b^\omega$ to itself, and any word $a^n c^\omega$ to $a^{2n} c^\omega$. For regular functions, we conjecture that 2DFT characterise the computable ones, but we have not been able to show it yet.

Finally, much of our work deals with reg-preserving functions in general. An interesting line of research would be to investigate continuity and uniform continuity for different classes of functions which have this property. One natural candidate is the class of *polyregular functions* introduced in [4] which enjoy several different characterisations and many nice properties, including being effectively reg-preserving. This means that continuity and computability also coincide, however deciding continuity seems challenging.

References

1 Alonzo Church. Logic, arithmetic and automata. In *Int. Congr. Math.*, pages 23–35, Stockholm, 1962.

2 Rajeev Alur, Emmanuel Filiot, and Ashutosh Trivedi. Regular transformations of infinite strings. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 65–74. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.18.

3 Félix Baschenis, Olivier Gauwin, Anca Muscholl, and Gabriele Puppis. One-way definability of two-way word transducers. *Log. Methods Comput. Sci.*, 14(4), 2018. doi:10.23638/LMCS-14(4:22)2018.

4 Mikolaj Bojanczyk. Polyregular functions. *CoRR*, abs/1810.08760, 2018. arXiv:1810.08760.

5 Michaël Cadilhac, Olivier Carton, and Charles Paperman. Continuity and rational functions. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 115:1–115:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.115.

6 Michaël Cadilhac, Andreas Krebs, Michael Ludwig, and Charles Paperman. A circuit complexity approach to transductions. In Giuseppe F. Italiano, Giovanni Pighizzini, and Donald Sannella, editors, *Mathematical Foundations of Computer Science 2015 – 40th International Symposium, MFCS 2015, Milan, Italy, August 24-28, 2015, Proceedings, Part I*, volume 9234 of *Lecture Notes in Computer Science*, pages 141–153. Springer, 2015. doi:10.1007/978-3-662-48057-1_11.

7 Olivier Carton, Dominique Perrin, and Jean-Eric Pin. Automata and semigroups recognizing infinite words. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, volume 2 of *Texts in Logic and Games*, pages 133–168. Amsterdam University Press, 2008.

8 Swarat Chaudhuri, Sriram Sankaranarayanan, and Moshe Y. Vardi. Regular real analysis. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 509–518. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.57.

9 Christian Choffrut. Minimizing subsequential transducers: a survey. *Theor. Comput. Sci.*, 292(1):131–143, 2003. doi:10.1016/S0304-3975(01)00219-5.

10 Vrunda Dave, Emmanuel Filiot, Shankara Narayanan Krishna, and Nathan Lhote. Deciding the computability of regular functions over infinite words. *CoRR*, abs/1906.04199, 2019. arXiv:1906.04199.

11 Joost Engelfriet, Hendrik Jan Hoogeboom, and Bart Samwel. XML navigation and transformation by tree-walking automata and transducers with visible and invisible pebbles. *CoRR*, abs/1809.05730, 2018. arXiv:1809.05730.

12 Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. On computability of data word functions defined by transducers. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures – 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12077 of *Lecture Notes in Computer Science*, pages 217–236. Springer, 2020. doi:10.1007/978-3-030-45231-5_12.

13 Emmanuel Filiot. Logic-automata connections for transformations. In Mohua Banerjee and Shankara Narayanan Krishna, editors, *Logic and Its Applications – 6th Indian Conference, ICLA 2015, Mumbai, India, January 8-10, 2015. Proceedings*, volume 8923 of *Lecture Notes in Computer Science*, pages 30–57. Springer, 2015. doi:10.1007/978-3-662-45824-2_3.

14 Emmanuel Filiot, Shibashis Guha, and Nicolas Mazzocchi. Two-way parikh automata. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India*, volume 150 of *LIPIcs*, pages 40:1–40:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.40.

15 Emmanuel Filiot, Nicolas Mazzocchi, and Jean-François Raskin. A pattern logic for automata with outputs. In Mizuho Hoshi and Shinnosuke Seki, editors, *Developments in Language Theory – 22nd International Conference, DLT 2018, Tokyo, Japan, September 10-14, 2018, Proceedings*, volume 11088 of *Lecture Notes in Computer Science*, pages 304–317. Springer, 2018. doi:10.1007/978-3-319-98654-8_25.

16 Michael Holtmann, Lukasz Kaiser, and Wolfgang Thomas. Degrees of lookahead in regular infinite games. *Log. Methods Comput. Sci.*, 8(3), 2012. doi:10.2168/LMCS-8(3:24)2012.

17 Oscar H. Ibarra. Automata with reversal-bounded counters: A survey. In Helmut Jürgensen, Juhani Karhumäki, and Alexander Okhotin, editors, *Descriptional Complexity of Formal Systems – 16th International Workshop, DCFS 2014, Turku, Finland, August 5-8, 2014. Proceedings*, volume 8614 of *Lecture Notes in Computer Science*, pages 5–22. Springer, 2014. doi:10.1007/978-3-319-09704-6_2.

18 J.R. Büchi and L.H. Landweber. Solving sequential conditions finite-state strategies. *Trans. Ameri. Math. Soc.*, 138:295–311, 1969.

19 S. Rao Kosaraju. Regularity preserving functions. *SIGACT News*, 6(2):16–17, April 1974. doi:10.1145/1008304.1008306.

20 Jean-Éric Pin and Pedro V. Silva. On uniformly continuous functions for some profinite topologies. *Theor. Comput. Sci.*, 658:246–262, 2017. doi:10.1016/j.tcs.2016.06.013.

21 Christophe Prieur. How to decide continuity of rational functions on infinite words. *Theor. Comput. Sci.*, 276(1-2):445–447, 2002. doi:10.1016/S0304-3975(01)00307-3.

22 W. Rudin. *Principles of Mathematical Analysis*. International series in pure and applied mathematics. McGraw-Hill, 1976. URL: https://books.google.pl/books?id=kwqzPAAACAAJ.

23 Joel I. Seiferas and Robert McNaughton. Regularity-preserving relations. *Theor. Comput. Sci.*, 2(2):147–154, 1976. doi:10.1016/0304-3975(76)90030-X.

24 Richard Edwin Stearns and Juris Hartmanis. Regularity preserving modifications of regular expressions. *Inf. Control.*, 6(1):55–69, 1963. doi:10.1016/S0019-9958(63)90110-4.

25 Klaus Weihrauch. *Computable Analysis – An Introduction*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1st edition, 2000. doi:10.1007/978-3-642-56999-9.

Residual Nominal Automata

Joshua Moerman
RTWH Aachen University, Germany

Matteo Sammartino
Royal Holloway University of London, UK
University College London, UK

Abstract

We are motivated by the following question: which nominal languages admit an active learning algorithm? This question was left open in previous work, and is particularly challenging for languages recognised by nondeterministic automata. To answer it, we develop the theory of *residual nominal automata*, a subclass of nondeterministic nominal automata. We prove that this class has canonical representatives, which can always be constructed via a finite number of observations. This property enables active learning algorithms, and makes up for the fact that residuality – a semantic property – is undecidable for nominal automata. Our construction for canonical residual automata is based on a machine-independent characterisation of residual languages, for which we develop new results in nominal lattice theory. Studying residuality in the context of nominal languages is a step towards a better understanding of learnability of automata with some sort of nondeterminism.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Automated reasoning

Keywords and phrases nominal automata, residual automata, derivative language, decidability, closure, exact learning, lattice theory

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.44

Related Version Full version at `https://arxiv.org/abs/1910.11666`.

Funding ERC AdG project 787914 FRAPPANT, EPSRC Standard Grant CLeVer (EP/S028641/1).

Acknowledgements We would like to thank Gerco van Heerdt for providing examples similar to that of $\mathcal{L}_r$ in the context of probabilistic automata. We thank Borja Balle for references on residual probabilistic languages, and Henning Urbat for discussions on nominal lattice theory. Lastly, we thank the reviewers of a previous version of this paper for their interesting questions and suggestions.

1 Introduction

Formal languages over infinite alphabets have received considerable attention recently. They include data languages for reasoning about XML databases [32], trace languages for analysis of programs with resource allocation [18], and behaviour of programs with data flows [19]. Typically, these languages are accepted by *register automata*, first introduced in the seminal paper [20]. Another appealing model is that of *nominal automata* [6]. While nominal automata are as expressive as register automata, they enjoy convenient properties. For example, the deterministic ones admit canonical minimal models, and the theory of formal languages and many textbook algorithms generalise smoothly.

In this paper, we investigate the properties of so-called *residual* nominal automata. An automaton accepting a language $\mathcal{L}$ is residual whenever the language of each state is a *derivative* of $\mathcal{L}$. In the context of regular languages over finite alphabets, residual finite state automata (RFSAs) are a subclass of nondeterministic finite automata (NFAs) introduced by Denis et al. [14] as a solution to the well-known problem of NFAs *not having* unique minimal representatives. They show that every regular language $\mathcal{L}$ admits a unique canonical RFSA.

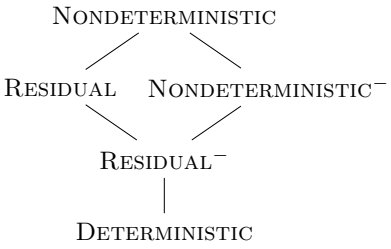


Figure 1 Relationship between classes of nominal languages. Edges are strict inclusions. With $\cdot^-$ we denote classes where automata are not allowed to *guess* values, i.e., to store symbols in registers without explicitly reading them.

Residual automata play a key role in the context of *exact learning*[1], in which one computes an automaton representation of an unknown language via a finite number of observations. The defining property of residual automata allows one to (eventually) observe the semantics of each state independently. In the finite-alphabet setting, residuality underlies the seminal algorithm $\mathsf{L}^\star$ for learning deterministic automata [1] (deterministic automata are always residual), and enables efficient algorithms for learning nondeterministic [8] and alternating automata [2, 3]. Residuality has also been studied for learning probabilistic automata [13]. Existence of canonical residual automata is crucial for the convergence of these algorithms.

Our investigation of residuality in the nominal setting is motivated by the following question: which nominal languages admit an exact learning algorithm? In previous work [28], we have shown that the $\mathsf{L}^\star$ algorithm generalises smoothly to nominal languages, meaning that deterministic nominal automata can be learned. However, the general non-deterministic case proved to be significantly more challenging. In fact, in stark contrast with the finite-alphabet case, nondeterministic nominal automata are *strictly more expressive* than deterministic ones, thus residual automata are not just succinct representations of deterministic languages. As a consequence, our attempt to generalise the $\mathsf{NL}^\star$ algorithm for nondeterministic finite automata to the nominal setting did not fully succeed: we could only prove that it works for deterministic languages, leaving the nondeterministic case open. By investigating residual languages, and how they relate to deterministic and nondeterministic ones, we are finally able to settle this case.

In summary, our contributions are as follows:

- Section 3: We refine nominal languages as depicted in Figure 1, by giving separating languages for each class.
- Section 4: We develop new results of nominal lattice theory, and we provide the main characterisation theorem (Theorem 4.10), showing that the class of residual languages allow for canonical automata which: a) are minimal in their respective class and unique (up to isomorphism); b) can be constructed via a finite number of observations of the language. Both properties are crucial for learning. We prove this important result by a machine-independent characterisation of those classes of languages. We also give an analogous result for non-guessing languages (Theorem 4.16).
- Section 5: We study decidability and closure properties. Many decision problems, such as equivalence and universality, are known to be undecidable for nondeterministic nominal automata. For residual automata, we show that universality becomes decidable. However, the problem of whether an automaton is residual is undecidable.

[1] Exact learning is also known as query learning or active (automata) learning [1].

- Section 6: We settle important open questions about exact learning of nominal languages. We show that residuality does not imply convergence of existing algorithms, and we give a (modified) $\mathsf{NL}^\star$-style algorithm that works precisely for residual languages.

This research mirrors that of *residual probabilistic automata* [13]. There, too, one has distinct classes of which the deterministic and residual ones admit canonical automata and have an algebraic characterisation. We believe that our results contribute to a better understanding of learnability of automata with some sort of nondeterminism.

2 Preliminaries

We recall the notions of nominal sets [33] and nominal automata [6]. Let $\mathbb{A}$ be a countably infinite set of *atoms*[2] and let $\mathsf{Perm}(\mathbb{A})$ be the set of *permutations on* $\mathbb{A}$, i.e., the bijective functions $\pi\colon \mathbb{A} \to \mathbb{A}$. Permutations form a group where the unit is given by the identity function, the inverse by functional inverse, and multiplication by function composition.

A *nominal set* is a set X equipped with a function $\cdot\colon \mathsf{Perm}(\mathbb{A}) \times X \to X$, interpreting permutations over X. This function must be a *group action* of $\mathsf{Perm}(\mathbb{A})$, i.e., it must satisfy $\mathsf{id} \cdot x = x$ and $\pi \cdot (\pi' \cdot x) = (\pi \circ \pi') \cdot x$. We say that a set $A \subset \mathbb{A}$ *supports* $x \in X$ whenever $\pi \cdot x = x$ for all π fixing A, i.e., such that $\pi_{|A} = \mathsf{id}_A$. We require for nominal sets that each element x has a *finite* support. We denote by $\mathsf{supp}(x)$ the smallest finite set supporting x.

The *orbit* $\mathsf{orb}(x)$ of $x \in X$ is the set of elements in X reachable from x via permutations: $\mathsf{orb}(x) \coloneqq \{\pi \cdot x \mid \pi \in \mathsf{Perm}(\mathbb{A})\}$. X is *orbit-finite* whenever it is a finite union of orbits. Orbit-finite sets are finitely-representable, hence algorithmically tractable [5].

Given a nominal set X, a subset $Y \subseteq X$ is *equivariant* if it is preserved by permutations, i.e., $\pi \cdot Y = Y$, for all $\pi \in \mathsf{Perm}(\mathbb{A})$, where π acts element-wise. This definition extends to relations and functions. For instance, a function $f\colon X \to Y$ between nominal sets is equivariant whenever $\pi \cdot f(x) = f(\pi \cdot x)$. Given a nominal set X, the *nominal power set* is defined as $\mathcal{P}_{\mathsf{fs}}(X) \coloneqq \{U \subseteq X \mid U \text{ is finitely supported}\}$.

We recall the notion of nominal automaton from [6]. The theory of nominal automata seamlessly extends classical automata theory by having orbit-finite nominal sets and equivariant functions in place of finite sets and functions.

▶ **Definition 2.1.** *A (nondeterministic)* nominal automaton $\mathcal{A}$ *consists of: an orbit-finite nominal set* Σ*, the alphabet; an orbit-finite nominal set of states* Q*; equivariant subsets* $I, F \subseteq Q$ *of initial and final states; and an equivariant subset* $\delta \subseteq Q \times \Sigma \times Q$ *of transitions.*

The usual notions of acceptance and language apply. We denote the language of $\mathcal{A}$ by $\mathcal{L}(\mathcal{A})$, and the language accepted by a state $q \in Q$ by $\mathcal{L}(q)$. Note that the language $\mathcal{L}(\mathcal{A}) \in \mathcal{P}_{\mathsf{fs}}(\Sigma^*)$ is equivariant, and that $\mathcal{L}(q) \in \mathcal{P}_{\mathsf{fs}}(\Sigma^*)$ need not be equivariant, but it is supported by $\mathsf{supp}(q)$.

We recall the notion of *derivative language* [14].[3]

▶ **Definition 2.2.** *Given a language* $\mathcal{L}$ *and a word* $u \in \Sigma^*$*, we define the* derivative *of* $\mathcal{L}$ *w.r.t.* u *as* $u^{-1}\mathcal{L} \coloneqq \{w \mid uw \in \mathcal{L}\}$ *and the set of all derivatives as* $\mathsf{Der}(\mathcal{L}) \coloneqq \left\{u^{-1}\mathcal{L} \mid u \in \Sigma^*\right\}$.

These definitions seamlessly extend to the nominal setting. Note that $w^{-1}\mathcal{L}$ is finitely supported whenever $\mathcal{L}$ is.

[2] Sometimes these are called *data values*.

[3] This is sometimes called a *residual language* or *left quotient*. We do not use the term residual language here, because residual language will mean a language accepted by a residual automaton.

Of special interest are the deterministic, residual, and non-guessing nominal automata, which we introduce next.

▶ **Definition 2.3.** *A nominal automaton $\mathcal{A}$ is:*

- Deterministic *if $I = \{q_0\}$, and for each $q \in Q$ and $a \in \Sigma$ there is a unique q' such that $(q, a, q') \in \delta$. In this case, the relation is in fact functional $\delta \colon Q \times \Sigma \to Q$.*
- Residual *if each state $q \in Q$ accepts a derivative of $\mathcal{L}(\mathcal{A})$, formally: $\mathcal{L}(q) = w^{-1}\mathcal{L}(\mathcal{A})$ for some word $w \in \Sigma^*$. The words w such that $\mathcal{L}(q) = w^{-1}\mathcal{L}(\mathcal{A})$ are called* characterising words *for the state q.*
- Non-guessing *if $\mathsf{supp}(q_0) = \emptyset$, for each $q_0 \in I$, and $\mathsf{supp}(q') \subseteq \mathsf{supp}(q) \cup \mathsf{supp}(a)$, for each $(q, a, q') \in \delta$.*

Observe that the transition function of a deterministic automaton preserves supports (i.e., if C supports (q, a) then C also supports $\delta(q, a)$). Consequently, all deterministic automata are non-guessing. For the sake of succinctness, in the following we drop the qualifier "nominal" when referring to these classes of nominal automata.

For many examples, it is useful to define the notion of an anchor. Given a state q, a word w is an *anchor* if $\delta(I, w) = \{q\}$, that is, the word w leads to q and no other state. Every anchor for q is also a characterising word for q (but not vice versa).

Finally, we recall the Myhill-Nerode theorem for nominal automata.

▶ **Theorem 2.4** ([6, Theorem 5.2])**.** *Let $\mathcal{L}$ be a language. Then $\mathcal{L}$ is accepted by a deterministic automaton if and only if $\mathsf{Der}(\mathcal{L})$ is orbit-finite.*

3 Separating languages

Deterministic, nondeterministic and residual automata have the same expressive power when dealing with finite alphabets. The situation is more nuanced in the nominal setting. We now give one language for each class in Figure 1. For the sake of simplicity, we will use the one-orbit nominal set of atoms $\mathbb{A}$ as alphabet. These languages separate the different classes, meaning that they belong to the respective class, but not to the classes below or beside it.

For each example language $\mathcal{L}$, we depict: a nominal automaton recognising $\mathcal{L}$ (on the left); the set of derivatives $\mathsf{Der}(\mathcal{L})$ (on the right). We make explicit the poset structure of $\mathsf{Der}(\mathcal{L})$: grey rectangles represent orbits of derivatives, and lines stand for set inclusions (we grey out irrelevant ones). This poset may not be orbit-finite, in which case we depict a small, indicative part. Observing the poset structure of $\mathsf{Der}(\mathcal{L})$ explicitly is important for later, where we show that the existence of residual automata depends on it. We write $aa^{-1}\mathcal{L}$ to mean $(aa)^{-1}\mathcal{L}$. Variables $a, b, \ldots$ are always atoms and $u, w, \ldots$ are always words.

Deterministic: First symbol equals last symbol

Consider the language $\mathcal{L}_{\mathrm{d}} \coloneqq \{awa \mid a \in \mathbb{A}, w \in \mathbb{A}^*\}$. This is accepted by the following deterministic nominal automaton. The automaton is actually infinite-state, but we represent it symbolically using a register-like notation, where we annotate each state with the current

Figure 2 A deterministic automaton accepting $\mathcal{L}_{\mathrm{d}}$, and the poset $\mathsf{Der}(\mathcal{L}_{\mathrm{d}})$.

value of a register. Note that the derivatives $a^{-1}\mathcal{L}_{\mathrm{d}}, b^{-1}\mathcal{L}_{\mathrm{d}}, \ldots$ are in the same orbit. In total $\mathsf{Der}(\mathcal{L}_{\mathrm{d}})$ has three orbits, which correspond to the three orbits of states in the deterministic automaton. The derivative $awa^{-1}\mathcal{L}_{\mathrm{d}}$, for example, equals $aa^{-1}\mathcal{L}_{\mathrm{d}}$.

Non-guessing residual: Some atom occurs twice

The language is $\mathcal{L}_{\mathrm{ng,r}} \coloneqq \{uavaw \mid u, v, w \in \mathbb{A}^*, a \in \mathbb{A}\}$. The poset $\mathsf{Der}(\mathcal{L}_{\mathrm{ng,r}})$ is not orbit-finite, so by the nominal Myhill-Nerode theorem there is no deterministic automaton accepting $\mathcal{L}_{\mathrm{ng,r}}$. However, derivatives of the form $ab^{-1}\mathcal{L}_{\mathrm{ng,r}}$ can be written as a union $ab^{-1}\mathcal{L}_{\mathrm{ng,r}} = a^{-1}\mathcal{L}_{\mathrm{ng,r}} \cup b^{-1}\mathcal{L}_{\mathrm{ng,r}}$. In fact, we only need an orbit-finite set of derivatives to recover $\mathsf{Der}(\mathcal{L}_{\mathrm{ng,r}})$. These orbits are highlighted in the diagram on the right. Selecting the "right" derivatives is the key idea behind constructing residual automata in Theorem 4.10.

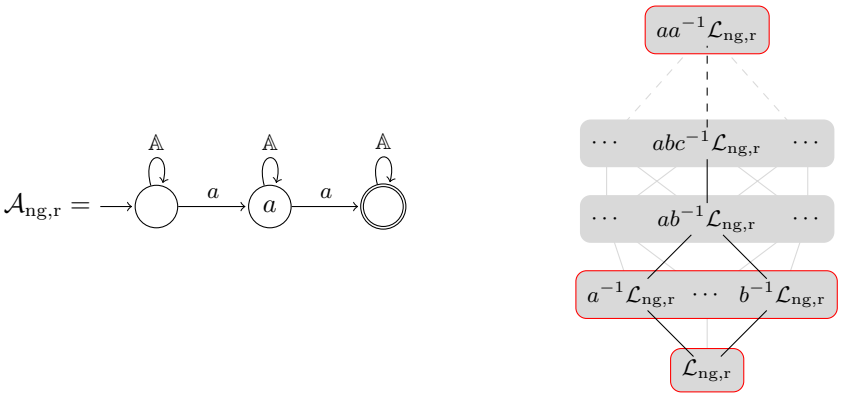


Figure 3 A (nonresidual) nondeterministic automaton accepting $\mathcal{L}_{\mathrm{ng,r}}$, and the poset $\mathsf{Der}(\mathcal{L}_{\mathrm{ng,r}})$.

Nondeterministic: Last letter is unique

The language is $\mathcal{L}_{\mathrm{n}} \coloneqq \{wa \mid a \text{ not in } w\} \cup \{\epsilon\}$. Derivatives $a^{-1}\mathcal{L}_{\mathrm{n}}$ are again unions of smaller languages: $a^{-1}\mathcal{L}_{\mathrm{n}} = \bigcup_{b \neq a} ab^{-1}\mathcal{L}_{\mathrm{n}}$. (We have omitted languages like $aa^{-1}\mathcal{L}_{\mathrm{n}}$, as they only differ from $a^{-1}\mathcal{L}_{\mathrm{n}}$ on the empty word.) However, the poset $\mathsf{Der}(\mathcal{L})$ has an infinite descending chain of languages (with an increasing support), namely $a^{-1}\mathcal{L} \supset ab^{-1}\mathcal{L} \supset abc^{-1}\mathcal{L} \supset \ldots$ The existence of a such a chain implies that $\mathcal{L}_{\mathrm{n}}$ cannot be accepted by a residual automaton. This is a consequence of Theorem 4.10, as we shall see later.

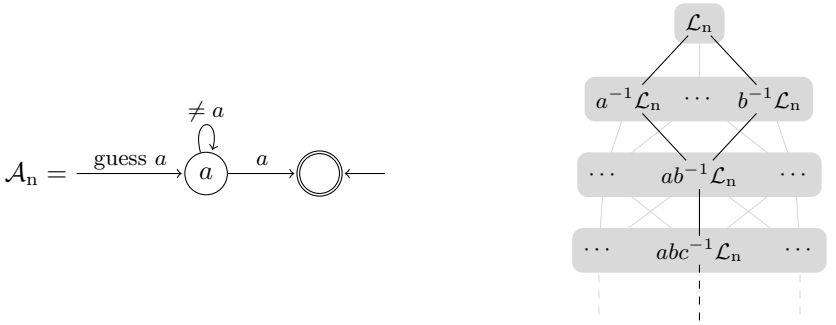


Figure 4 A nondeterministic automaton accepting $\mathcal{L}_{\mathrm{n}}$, and the poset $\mathsf{Der}(\mathcal{L}_{\mathrm{n}})$.

Residual: Last letter is unique but anchored

Consider the alphabet $\Sigma = \mathbb{A} \cup \{\mathsf{Anc}(a) \mid a \in \mathbb{A}\}$, where Anc is nothing more than a label. We add the transitions $(a, \mathsf{Anc}(a), a)$ to the automaton in the previous example. We obtain the language $\mathcal{L}_\mathrm{r} = \mathcal{L}(\mathcal{A}_\mathrm{r})$. Here, we have forced the automaton to be residual, by adding an anchor to the first state. Nevertheless, guessing is still necessary. In the poset, we note that all elements in the descending chain can now be obtained as unions of $\mathsf{Anc}(a)^{-1}\mathcal{L}_\mathrm{r}$. For instance, $a^{-1}\mathcal{L}_\mathrm{r} = \bigcup_{b \neq a} \mathsf{Anc}(b)^{-1}\mathcal{L}_\mathrm{r}$. Note that $\mathsf{Anc}(a)\mathsf{Anc}(b)^{-1}\mathcal{L}_\mathrm{r} = \emptyset$ and $\mathsf{Anc}(a)a^{-1}\mathcal{L}_\mathrm{r} = \{\epsilon\}$.

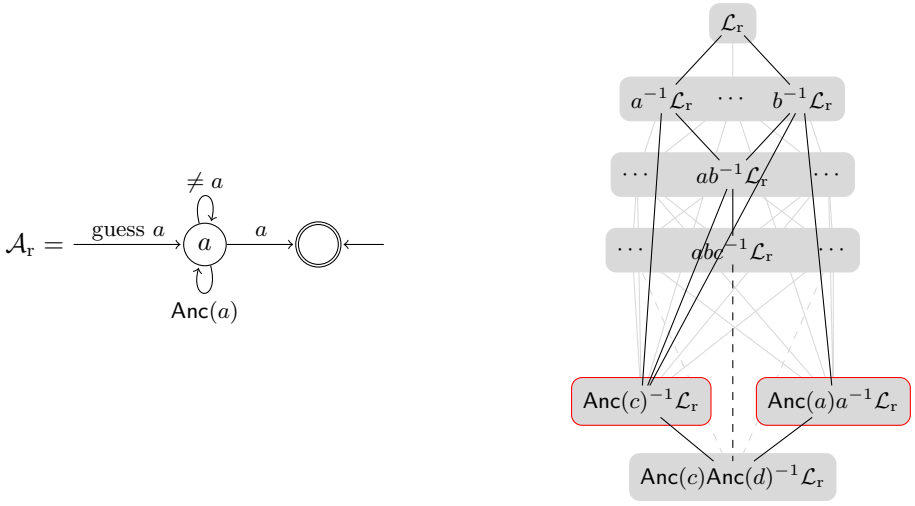


Figure 5 A residual automaton accepting $\mathcal{L}_\mathrm{r}$, and the poset $\mathsf{Der}(\mathcal{L}_\mathrm{r})$.

Non-guessing nondeterministic: Repeated atom with different successor

The language is $\mathcal{L}_\mathrm{ng} \coloneqq \{uabvac \mid u, v \in \mathbb{A}^*, a, b, c \in \mathbb{A}, b \neq a\}$. (We allow $a = b$ or $a = c$.) This is a language which can be accepted by a non-guessing automaton. However, there is no residual automaton for this language. The poset structure of $\mathsf{Der}(\mathcal{L}_\mathrm{ng})$ is very complicated. We will return to this example after Theorem 4.10.

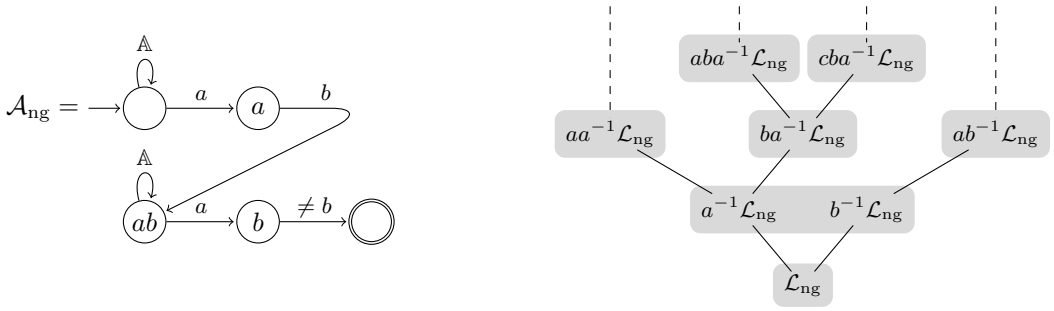


Figure 6 A deterministic automaton accepting $\mathcal{L}_\mathrm{ng}$, and the poset $\mathsf{Der}(\mathcal{L}_\mathrm{ng})$.

4 Canonical Residual Nominal Automata

In this section we will give a characterisation of *canonical* residual automata. We will first introduce notions of nominal lattice theory, then we will state our main result (Theorem 4.10). We conclude the section by providing similar results for non-guessing automata.

4.1 Nominal lattice theory

We abstract away from words and languages and consider the set $\mathcal{P}_{\mathsf{fs}}(Z)$ for an arbitrary nominal set Z. This is a Boolean algebra of which the operations $\wedge, \vee, \neg$ are all equivariant maps [17]. Moreover, the finitely supported union

$$\bigvee : \mathcal{P}_{\mathsf{fs}}(\mathcal{P}_{\mathsf{fs}}(Z)) \to \mathcal{P}_{\mathsf{fs}}(Z)$$

is also equivariant. We note that this is more general than a binary union, but it is not a complete join semi-lattice. Hereafter, we shall denote set inclusion by $\leq$ ($<$ when strict).

▶ **Definition 4.1.** *Given a nominal set Z and $X \subseteq \mathcal{P}_{\mathsf{fs}}(Z)$ equivariant*[4]*, we define the set generated by X as*

$$\langle X \rangle \coloneqq \left\{ \bigvee \mathfrak{x} \mid \mathfrak{x} \subseteq X \text{ finitely supported} \right\} \subseteq \mathcal{P}_{\mathsf{fs}}(Z).$$

▶ Remark 4.2. The set $\langle X \rangle$ is closed under the operation $\bigvee$, and moreover is the smallest equivariant set closed under $\bigvee$ containing X. In other words, $\langle - \rangle$ defines a closure operator. We will often say "X generates Y", by which we mean $Y \subseteq \langle X \rangle$.

▶ **Definition 4.3.** *Let $X \subseteq \mathcal{P}_{\mathsf{fs}}(Z)$ equivariant and $x \in X$, we say that x is* join-irreducible *in X if it is non-empty and $x = \bigvee \mathfrak{x} \implies x \in \mathfrak{x}$, for every finitely supported $\mathfrak{x} \subseteq X$. The set of all join-irreducible elements is denoted by*

$$\mathsf{JI}(X) \coloneqq \{x \in X \mid x \text{ join-irreducible in } X\}.$$

This is again an equivariant set.

▶ Remark 4.4. In lattice and order theory, join-irreducible elements are usually defined only for a lattice (see, e.g., [11]). However, we define them for arbitrary subsets of a lattice. (Note that a subset of a lattice is merely a poset.) This generalisation will be needed later, when we consider the poset $\mathsf{Der}(\mathcal{L})$ which is not a lattice, but is contained in the lattice $\mathcal{P}_{\mathsf{fs}}(\Sigma^*)$.

▶ Remark 4.5. The notion of join-irreducible, as we have defined here, corresponds to the notion of *prime* in [8, 14, 28]. Unfortunately, the word *prime* has a slightly different meaning in lattice theory. We stick to the terminology of lattice theory.

If a set Y is well-behaved, then its join-irreducible elements will actually generate the set Y. This is normally proven with a descending chain condition. We first restrict our attention to orbit-finite sets. The following Lemma extends [11, Lemma 2.45] to the nominal setting.

▶ **Lemma 4.6.** *Let $X \subseteq \mathcal{P}_{\mathsf{fs}}(Z)$ be an orbit-finite and equivariant set.*

1. *Let $a \in X, b \in \mathcal{P}_{\mathsf{fs}}(Z)$ and $a \not\leq b$. Then there is a join-irreducible $x \in X$ such that $x \leq a$ and $x \not\leq b$.*
2. *Let $a \in X$, then $a = \bigvee \{x \in X \mid x \text{ join-irreducible in } X \text{ and } x \leq a\}$.*

[4] A similar definition could be given for finitely supported X. In fact, all results in this section generalise to finitely supported. But we use equivariance for convenience.

▶ **Corollary 4.7.** *Let $X \subseteq \mathcal{P}_{\mathsf{fs}}(Z)$ be an orbit-finite equivariant subset. The join-irreducibles of X generate X, i.e., $X \subseteq \langle \mathsf{JI}(X) \rangle$.*

So far, we have defined join-irreducible elements relative to some fixed set. We will now show that these elements remain join-irreducible when considering them in a bigger set, as long as the bigger set is generated by the smaller one. This will later allow us to talk about *the* join-irreducible elements.

▶ **Lemma 4.8.** *Let $Y \subseteq X \subseteq \mathcal{P}_{\mathsf{fs}}(Z)$ equivariant and suppose that $X \subseteq \langle \mathsf{JI}(Y) \rangle$. Then $\mathsf{JI}(Y) = \mathsf{JI}(X)$.*

In other words, the join-irreducibles of X are the smallest set generating X.

▶ **Corollary 4.9.** *If an orbit-finite set Y generates X, then $\mathsf{JI}(X) \subseteq Y$.*

4.2 Characterising Residual Languages

We are now ready to state and prove the main theorem of this paper. We fix the alphabet Σ. Recall that the nominal Myhill-Nerode theorem tells us that a language is accepted by a deterministic automaton if and only if $\mathsf{Der}(\mathcal{L})$ is orbit-finite. Here, we give a similar characterisation for languages accepted by residual automata. Moreover, the following result gives a canonical construction.

▶ **Theorem 4.10.** *Given a language $\mathcal{L} \in \mathcal{P}_{\mathsf{fs}}(\Sigma^*)$, the following are equivalent:*

1. *$\mathcal{L}$ is accepted by a residual automaton.*
2. *There is some orbit-finite set $J \subseteq \mathsf{Der}(\mathcal{L})$ which generates $\mathsf{Der}(\mathcal{L})$.*
3. *The set $\mathsf{JI}(\mathsf{Der}(\mathcal{L}))$ is orbit-finite and generates $\mathsf{Der}(\mathcal{L})$.*

Proof. We prove three implications:

(1 ⇒ 2) Take the set of languages accepted by the states: $J \coloneqq \{\mathcal{L}(q) \mid q \in \mathcal{A}\}$. This is clearly orbit-finite, since Q is. Moreover, each derivative is generated as follows: $w^{-1}\mathcal{L} = \bigvee \{\mathcal{L}(q) \mid q \in \delta(I, w)\}$.

(2 ⇒ 3) We can apply Lemma 4.8 with $Y = J$ and $X = \mathsf{Der}(\mathcal{L})$. Now it follows that $\mathsf{JI}(\mathsf{Der}(\mathcal{L}))$ is orbit-finite (since it is a subset of J) and generates $\mathsf{Der}(\mathcal{L})$.

(3 ⇒ 1) We can construct the following residual automaton, whose language is exactly $\mathcal{L}$:

$$\begin{aligned} Q &\coloneqq \mathsf{JI}(\mathsf{Der}(\mathcal{L})) \\ I &\coloneqq \left\{w^{-1}\mathcal{L} \in Q \mid w^{-1}\mathcal{L} \leq \mathcal{L}\right\} \\ F &\coloneqq \left\{w^{-1}\mathcal{L} \in Q \mid \epsilon \in w^{-1}\mathcal{L}\right\} \\ \delta(w^{-1}\mathcal{L}, a) &\coloneqq \left\{v^{-1}\mathcal{L} \in Q \mid v^{-1}\mathcal{L} \leq wa^{-1}\mathcal{L}\right\} \end{aligned}$$

First, note that $\mathcal{A} \coloneqq (\Sigma, Q, I, F, \delta)$ is a well-defined nominal automaton. In fact, all the components are orbit-finite, and equivariance of $\leq$ implies equivariance of δ. Second, we show by induction on words that each state $q = w^{-1}\mathcal{L}$ accepts its corresponding language, namely $\mathcal{L}(q) = w^{-1}\mathcal{L}$.

$$\begin{aligned} \epsilon \in \mathcal{L}(w^{-1}\mathcal{L}) &\iff w^{-1}\mathcal{L} \in F \iff \epsilon \in w^{-1}\mathcal{L} \\ au \in \mathcal{L}(w^{-1}\mathcal{L}) &\iff u \in \mathcal{L}\left(\delta(w^{-1}\mathcal{L}, a)\right) \\ &\iff u \in \mathcal{L}\left(\left\{v^{-1}\mathcal{L} \in Q \mid v^{-1}\mathcal{L} \leq wa^{-1}\mathcal{L}\right\}\right) \\ &\overset{\text{(i)}}{\iff} u \in \bigvee \left\{v^{-1}\mathcal{L} \in Q \mid v^{-1}\mathcal{L} \leq wa^{-1}\mathcal{L}\right\} \\ &\iff \exists v^{-1}\mathcal{L} \in Q \text{ with } v^{-1}\mathcal{L} \leq wa^{-1}\mathcal{L} \text{ and } u \in v^{-1}\mathcal{L} \\ &\overset{\text{(ii)}}{\iff} u \in wa^{-1}\mathcal{L} \iff au \in w^{-1}\mathcal{L} \end{aligned}$$

At step (i) we have used the induction hypothesis (u is a shorter word than au) and the fact that $\mathcal{L}(-)$ preserves unions. At step (ii, right-to-left) we have used that $v^{-1}\mathcal{L}$ is join-irreducible. The other steps are unfolding definitions.
Finally, note that $\mathcal{L} = \bigvee \{w^{-1}\mathcal{L} \mid w^{-1}\mathcal{L} \leq \mathcal{L}\}$, since the join-irreducible languages generate all languages. In particular, the initial states (together) accept $\mathcal{L}$. ◀

▶ **Corollary 4.11.** *The construction above defines a* canonical *residual automaton with the following uniqueness property: it has the minimal number of orbits of states and the maximal number of orbits of transitions.*

For finite alphabets, the classes of languages accepted by DFAs and NFAs are the same (by determinising an NFA). This means that $\mathsf{Der}(\mathcal{L})$ is always finite if $\mathcal{L}$ is accepted by an NFA, and we can always construct the canonical RFSA. Here, this is not the case, that is why we need to stipulate (in Theorem 4.10) that the set $\mathsf{JI}(\mathsf{Der}(\mathcal{L}))$ is orbit-finite *and* actually generates $\mathsf{Der}(\mathcal{L})$. Either condition may fail, as we will see in Example 4.13.

▶ **Example 4.12.** In this example we show that residual automata can also be used to compress deterministic automata. The language $\mathcal{L} \coloneqq \{abb\dots b \mid a \neq b\}$ can be accepted by a deterministic automaton of 4 orbits, and this is minimal. (A zero amount of bs is also accepted in $\mathcal{L}$.) The minimal residual automaton, however, has only 2 orbits, given by the join-irreducible languages:

$$\epsilon^{-1}\mathcal{L} = \{abb\dots b \mid a \neq b\}$$
$$ab^{-1}\mathcal{L} = \{bb\dots b\} \qquad (a, b \in \mathbb{A} \text{ distinct})$$

The trick in defining the automaton is that the a-transition from $\epsilon^{-1}\mathcal{L}$ to $ab^{-1}\mathcal{L}$ *guesses* the value b. In the next section (Section 4.3), we will define the canonical *non-guessing* residual automaton, which has 3 orbits.

▶ **Example 4.13.** We return to the examples $\mathcal{L}_{\mathrm{n}}$ and $\mathcal{L}_{\mathrm{ng}}$ from Section 3. We claim that neither language can be accepted by a residual automaton.

For $\mathcal{L}_{\mathrm{n}}$ we note that there is an infinite descending chain of derivatives

$$\mathcal{L}_{\mathrm{n}} > a^{-1}\mathcal{L}_{\mathrm{n}} > ab^{-1}\mathcal{L}_{\mathrm{n}} > abc^{-1}\mathcal{L}_{\mathrm{n}} > \cdots$$

Each of these languages can be written as a union of smaller derivatives. For instance, $a^{-1}\mathcal{L}_{\mathrm{n}} = \bigcup_{b\neq a} ab^{-1}\mathcal{L}_{\mathrm{n}}$. This means that $\mathsf{JI}(\mathsf{Der}(\mathcal{L}_{\mathrm{n}})) = \emptyset$, hence it does not generate $\mathsf{Der}(\mathcal{L}_{\mathrm{n}})$ and by Theorem 4.10 there is no residual automaton.

In the case of $\mathcal{L}_{\mathrm{ng}}$, we have an infinite ascending chain

$$\mathcal{L}_{\mathrm{ng}} < a^{-1}\mathcal{L}_{\mathrm{ng}} < ba^{-1}\mathcal{L}_{\mathrm{ng}} < cba^{-1}\mathcal{L}_{\mathrm{ng}} < \cdots$$

This in itself is not a problem: the language $\mathcal{L}_{\mathrm{ng,r}}$ also has an infinite ascending chain. However, for $\mathcal{L}_{\mathrm{ng}}$, none of the languages in this chain are a union of smaller derivatives. Put differently: all the languages in this chain are join-irreducible (see appendix for the details). So the set $\mathsf{JI}(\mathsf{Der}(\mathcal{L}_{\mathrm{ng}}))$ is *not orbit-finite*. By Theorem 4.10, we conclude that there is no residual automaton accepting $\mathcal{L}_{\mathrm{ng}}$.

▶ Remark 4.14. For arbitrary (nondeterministic) languages there is also a characterisation in the style of Theorem 4.10. Namely, $\mathcal{L}$ is accepted by an automaton iff there is an orbit-finite set $Y \subseteq \mathcal{P}_{\mathsf{fs}}(\Sigma^*)$ which generates the derivatives. However, note that the set Y need not be a subset of the set of derivatives. In these cases, we do not have a canonical construction for the automaton. Different choices for Y define different automata and there is no way to pick Y naturally.

4.3 Automata without guessing

We reconsider the above results for non-guessing automata. Nondeterminism in nominal automata allows naturally for guessing, meaning that the automaton may store symbols in registers without explicitly reading them. However, the original definition of register automata in [20] does not allow for guessing, and non-guessing automata remain actively researched [29]. Register automata with guessing were introduced in [21], because it was realised that non-guessing automata are not closed under reversal.

To adapt to non-guessing automata, we redefine join-irreducible elements. As we would like to remove states which can be written as a "non-guessing" union of other states, we only consider joins of sets of elements where all elements are supported by the same support.

▶ **Definition 4.15.** *Let $X \subseteq \mathcal{P}_{\mathsf{fs}}(Z)$ be equivariant and $x \in X$, we say that x is* join-irreducible$^-$ in X *if $x = \bigvee \mathfrak{x} \implies x \in \mathfrak{x}$, for every finitely supported $\mathfrak{x} \subseteq X$ such that $\mathsf{supp}(x_0) \subseteq \mathsf{supp}(x)$, for each $x_0 \in \mathfrak{x}$. The set of all join-irreducible$^-$ elements is denoted by*

$$\mathsf{JI}^-(X) \coloneqq \left\{x \in X \mid x \text{ join-irreducible}^- \text{ in } X\right\}.$$

The only change required is an additional condition on the elements and supports in $\mathfrak{x}$. In particular, the sets $\mathfrak{x}$ are *uniformly supported* sets. Unions of such sets are called *uniformly supported unions*.

All the lemmas from the previous section are proven similarly. We state the main result for non-guessing automata.

▶ **Theorem 4.16.** *Given a language $\mathcal{L} \in \mathcal{P}_{\mathsf{fs}}(\Sigma^*)$, the following are equivalent:*

1. *$\mathcal{L}$ is accepted by a non-guessing residual automaton.*
2. *There is some orbit-finite set $J \subseteq \mathsf{Der}(\mathcal{L})$ which generates $\mathsf{Der}(\mathcal{L})$ by uniformly supported unions.*
3. *The set $\mathsf{JI}^-(\mathsf{Der}(\mathcal{L}))$ is orbit-finite and generates $\mathsf{Der}(\mathcal{L})$ by uniformly supported unions.*

Proof. The proof is similar to that of Theorem 4.10. However, we need a slightly different definition of the canonical automaton. It is defined as follows.

$$\begin{aligned} Q &\coloneqq \mathsf{JI}^-(\mathsf{Der}(\mathcal{L})) \\ I &\coloneqq \left\{w^{-1}\mathcal{L} \in Q \mid w^{-1}\mathcal{L} \leq \mathcal{L}, \mathsf{supp}(w^{-1}\mathcal{L}) \subseteq \mathsf{supp}(\mathcal{L})\right\} \\ F &\coloneqq \left\{w^{-1}\mathcal{L} \in Q \mid \epsilon \in w^{-1}\mathcal{L}\right\} \\ \delta(w^{-1}\mathcal{L}, a) &\coloneqq \left\{v^{-1}\mathcal{L} \in Q \mid v^{-1}\mathcal{L} \leq wa^{-1}\mathcal{L}, \mathsf{supp}(v^{-1}\mathcal{L}) \subseteq \mathsf{supp}(wa^{-1}\mathcal{L})\right\} \end{aligned}$$

Note that, in particular, the initial states have empty support since $\mathcal{L}$ is equivariant. This means that the automaton cannot guess any values at the start. Similarly, the transition relation does not allow for guessing. ◀

To better understand the structure of the canonical non-guessing residual automaton, we recall the following fact (see [33] for details) and its consequence on non-guessing automata.

▶ **Lemma 4.17.** *Let X be an orbit-finite nominal set and $A \subset \mathbb{A}$ be a finite set of atoms. The set $\{x \in X \mid A \text{ supports } x\}$ is finite.*

▶ **Corollary 4.18.** *The transition relation δ of non-guessing automata can be equivalently be described as a function $\delta \colon Q \times \Sigma \to \mathcal{P}_{\mathrm{fin}}(Q)$, where $\mathcal{P}_{\mathrm{fin}}(Q)$ is the set of* finite *subsets of Q.*

In particular, this shows that the canonical non-guessing residual automaton has finite nondeterminism. It also shows that it is sufficient to consider *finite unions* in Theorem 4.16, instead of uniformly supported unions.

5 Decidability and Closure Results

In this section we investigate decidability and closure properties. First, a positive result: universality is decidable for residual automata. This is in contrast to the nondeterministic case, where universality is undecidable, even for non-guessing automata [4].

▶ **Proposition 5.1.** *Universality for residual nominal automata is decidable. Formally: given a residual automaton $\mathcal{A}$, it is decidable whether $\mathcal{L}(\mathcal{A}) = \Sigma^*$.*

Second, a negative result: determining whether an automaton is residual is undecidable. In other words, residuality cannot be characterised as a syntactic property. This adds value to learning techniques, as they are able to provide automata that are residual by construction, thus "getting around" this undecidability issue.

▶ **Proposition 5.2.** *The problem of determining whether a given nondeterministic nominal automaton is residual is undecidable.*

The above result is obtained by reducing the universality problem for general nondeterministic nominal automata to the residuality problem. Given an automaton $\mathcal{A}$, we construct another automaton $\mathcal{A}'$ which is residual if and only if $\mathcal{A}$ is universal (see appendix for details). This result also holds for the subclass of non-guessing automata, as the construction of $\mathcal{A}'$ does not introduce any guessing and universality for non-guessing nondeterministic nominal automata is undecidable.

▶ Remark 5.3. Equivalence between residual nominal automata is still an open problem. The usual proof of undecidability of equivalence is via a reduction from universality. This proof does not work anymore, because universality for residual automata is decidable (Proposition 5.1). We conjecture that equivalence remains undecidable for residual automata.

Closure properties

We will now show that several closure properties fail for residual languages. Interestingly, this parallels the situation for probabilistic languages: residual ones are not even closed under convex sums. We emphasise that residual automata were devised for learning purposes, where closure properties play no significant role. In fact, one typically exploits closure properties of the wider class of nondeterministic models, e.g., for automata-based verification. The following results show that in our setting this is indeed unavoidable.

Consider the alphabet $\Sigma = \mathbb{A} \cup \{\mathsf{Anc}(a) \mid a \in \mathbb{A}\}$ and the residual language $\mathcal{L}_\mathrm{r}$ from Section 3. We consider a second language $\mathcal{L}_2 = \mathbb{A}^*$ which can be accepted by a deterministic (hence residual) automaton. We have the following non-closure results:

Union: The language $\mathcal{L} = \mathcal{L}_\mathrm{r} \cup \mathcal{L}_2$ cannot be accepted by a residual automaton. In fact, although derivatives of the form $\mathsf{Anc}(a)^{-1}\mathcal{L}$ are still join-irreducible (see Section 3, residual case), they have no summand $\mathbb{A}^*$, which means that they cannot generate $a^{-1}\mathcal{L} = \mathbb{A}^* \cup \bigcup_{b \neq a} \mathsf{Anc}(b)^{-1}\mathcal{L}$. By Theorem 4.10(3) it follows that $\mathcal{L}$ is not residual.

Intersection: The language $\mathcal{L} = \mathcal{L}_\mathrm{r} \cap \mathcal{L}_2 = \mathcal{L}_\mathrm{n}$ cannot be accepted by a residual automaton, as we have seen in Section 3.

Concatenation: The language $\mathcal{L} = \mathcal{L}_2 \cdot \mathcal{L}_\mathrm{r}$ cannot be accepted by a residual automaton, for similar reasons as the union.

Reversal: The language $\{aw \mid a \text{ not in } w\}$ is residual (even deterministic), but its reverse language is $\mathcal{L}_\mathrm{n}$ and cannot be accepted by a residual automaton.

Complement: Consider the language $\mathcal{L}_{\mathrm{ng,r}}$ of words where some atom occurs twice. Its complement $\overline{\mathcal{L}_{\mathrm{ng,r}}}$ is the language of all fresh atoms, which cannot even be recognised by a nondeterministic nominal automaton [6].

Closure under Kleene star is yet to be settled.

6 Exact learning

In our previous paper on learning nominal automata [28], we provided an exact learning algorithm for nominal deterministic languages. Moreover, we observed by experimentations that the algorithm was also able to learn specific nondeterministic languages. However, several questions on nominal languages remained open, most importantly:

- Which nominal languages can be characterised via a finite set of observations?
- Which nominal languages admit an Angluin-style learning algorithm?

In this section we will answer these questions using the theory developed in the previous sections.

6.1 Angluin-style learning

We briefly review the classical automata learning algorithms $\mathsf{L}^\star$ by Angluin [1] for deterministic automata, and $\mathsf{NL}^\star$ by Bollig et al. [8] for residual automata. Then we discuss convergence in the nominal setting.

Both algorithms can be seen as a game between two players: *the learner* and *the teacher*. The learner aims to construct the minimal automaton for an unknown language $\mathcal{L}$ over a finite alphabet Σ. In order to do this, it may ask the teacher, who knows about the language, two types of queries:

Membership query: Is a given word w in the target language, i.e., $w \in \mathcal{L}$?

Equivalence query: Does a given *hypothesis* automaton $\mathcal{H}$ recognise the target language, i.e., $\mathcal{L} = \mathcal{L}(\mathcal{H})$?

If the teacher replies *yes* to an equivalence query, then the algorithm terminates, as the hypothesis $\mathcal{H}$ is correct. Otherwise, the teacher must supply a *counterexample*, that is a word in the symmetric difference of $\mathcal{L}$ and $\mathcal{L}(\mathcal{H})$. Availability of equivalence queries may seem like a strong assumption, and in fact it is often weakened by allowing only random sampling (see [22] or [35] for details).

Observations about the language made by the learner via queries are stored in an *observation table* $\mathcal{T}$. This is a table where rows and columns range over two finite sets of words $S, E \subseteq \Sigma^\star$ respectively, and $\mathcal{T}(u, v) = 1$ if and only if $uv \in \mathcal{L}$. Intuitively, each row of $\mathcal{T}$ approximates a derivative of $\mathcal{L}$, in fact we have $\mathcal{T}(u) \subseteq u^{-1}\mathcal{L}$. However, the information contained in $\mathcal{T}$ may be incomplete: some derivatives $w^{-1}\mathcal{L}$ are not reached yet because no membership queries for w have been posed, and some pairs of rows $\mathcal{T}(u)$, $\mathcal{T}(v)$ may seem equal to the learner, because no word has been seen yet which distinguishes them. The learning algorithm will add new words to S when new derivatives are discovered, and to E when words distinguishing two previously identical derivatives are discovered.

The table $\mathcal{T}$ is *closed* whenever one-letter extensions of derivatives are already in the table, i.e., $\mathcal{T}$ has a row for $ua^{-1}\mathcal{L}$, for all $u \in S, a \in \Sigma$. If the table is closed,[5] $\mathsf{L}^\star$ is able to construct an automaton from $\mathcal{T}$, where states are distinct rows (i.e., derivatives). The construction follows the classical one for the canonical automaton of a language from its derivatives [31]. The $\mathsf{NL}^\star$ algorithm uses a modified notion of closedness, where one is allowed to take unions (i.e., a one-letter extension can be written as unions of rows in $\mathcal{T}$), and hence is able to learn a RFSA accepting the target language.

[5] $\mathsf{L}^\star$ also needs the table to be *consistent*. We do not need that in our discussion here.

When the table is not closed, then a derivative is missing, and a corresponding row needs to be added. Once an automaton is constructed, it is submitted in an equivalence query. If a counterexample is returned, then again the table is extended[6], after which the process is repeated iteratively.

6.2 The nominal case

In [28] we have given nominal versions of $\mathsf{L}^\star$ and $\mathsf{NL}^\star$, called $\nu\mathsf{L}^\star$ and $\nu\mathsf{NL}^\star$ respectively. They seamlessly extend the original algorithms by operating on orbit-finite sets. The algorithm $\nu\mathsf{L}^\star$ always terminates for deterministic languages, because distinct derivatives, and hence distinct rows in the observation table, are orbit-finitely many (see Theorem 2.4).

However, it will *never* terminate for languages not accepted by deterministic automata (such as residual or nondeterministic languages).

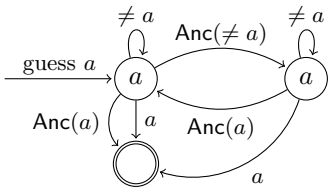
▶ **Theorem 6.1** ([27]). *$\nu\mathsf{L}^\star$ converges if and only if $\mathsf{Der}(\mathcal{L})$ is orbit-finite, in which case it outputs the canonical deterministic automaton accepting $\mathcal{L}$. Moreover, at most $\mathcal{O}(nk)$ equivalence queries are needed, where n is the number of orbits of the minimal deterministic automaton, and k is the maximum support size of its states.*

The nondeterministic case is more interesting. Using Theorem 4.10, we can finally establish which nondeterministic languages can be characterised via orbit-finitely-many observations.

▶ **Corollary 6.2** (of Theorem 4.10). *Let $\mathcal{L}$ be a nondeterministic nominal language. Then $\mathcal{L}$ can be represented via an observation table with orbit-finitely-many rows and columns if and only if $\mathcal{L}$ is residual. Rows of this table correspond to join-irreducible derivatives.*

This explains why in [28] $\nu\mathsf{NL}^\star$ was able to learn some residual nondeterministic automata: an orbit-finite observation table exists, which allows $\nu\mathsf{NL}^\star$ to construct the canonical residual automaton. Unfortunately, the current $\nu\mathsf{NL}^\star$ algorithm does not guarantee that it finds this orbit-finite observation table. We only have that guarantee for deterministic languages. The following example shows that $\nu\mathsf{NL}^\star$ may indeed diverge when trying to close the table.

▶ **Example 6.3.** Suppose $\nu\mathsf{NL}^\star$ tries to learn the residual language $\mathcal{L}$ accepted by the automaton below over the alphabet $\Sigma = \mathbb{A} \cup \{\mathsf{Anc}(a) \mid a \in \mathbb{A}\}$. This is a slight modification of the residual language of Section 3.



The algorithm starts by considering the row for the empty word ϵ, and its one-letter extensions $\epsilon \cdot a = a$ and $\epsilon \cdot \mathsf{Anc}(a) = \mathsf{Anc}(a)$. These rows correspond to the derivatives $\epsilon^{-1}\mathcal{L} = \mathcal{L}$, $a^{-1}\mathcal{L}$ and $\mathsf{Anc}(a)^{-1}\mathcal{L}$. Column labels are initialised to the empty word ϵ. At this point $a^{-1}\mathcal{L}$ and $\mathsf{Anc}(a)^{-1}\mathcal{L}$ appear identical, as the only column ϵ does not distinguish them. However, they appear different from $\epsilon^{-1}\mathcal{L}$, so the algorithm will add the row for either a or $\mathsf{Anc}(a)$ in order

[6] $\mathsf{L}^\star$ and $\mathsf{NL}^\star$ adopt different counterexample-handling strategies: the former adds a new row, the latter a new column. Both result in a new derivative being detected.

to close the table. Suppose the algorithm decides to add a. Then it will consider one-letter extensions ab, abc, $abcd$, etc... Since these correspond to different derivatives – each strictly smaller than the previous one – the algorithm will get stuck in an attempt to close the table. At no point it will try to close the table with the word $\mathsf{Anc}(a)$, since it stays equivalent to a. So in this case $\nu\mathsf{NL}^\star$ will not terminate. However, if the algorithm instead adds $\mathsf{Anc}(a)$ to the row labels, it will then also add $\mathsf{Anc}(a)\mathsf{Anc}(b)$, which is a characterising word for the initial state. In that case, $\nu\mathsf{NL}^\star$ will terminate.

While there is no hope of convergence in the non-residual case, as no orbit-finite observation table exists characterising derivatives, we now propose a modification of $\nu\mathsf{NL}^\star$ which guarantees termination for residual languages.

▶ **Theorem 6.4.** *There is an algorithm which query learns residual nominal languages.*

Proof (Sketch). When the algorithm adds a word w to the set of rows, then it also adds all other words of length $|w|$.[7] Since all words of bounded length are added, the algorithm will eventually find all words that are characterising for states of the canonical residual automaton, and it will therefore be able to reconstruct this automaton. See appendix for details. ◀

Unfortunately, considering all words bounded by a certain length requires many membership queries. In fact, characterising words can be exponential in length [14], meaning that this algorithm may need doubly exponentially many membership queries.

▶ Remark 6.5. We note that nondeterministic automata can be enumerated, and hence can be learned via equivalence queries only. This would result in a highly inefficient algorithm. This parallels the current understanding of learning probabilistic languages. Although efficient (learning in the limit) learning algorithms for deterministic and residual languages exist [12], the general case is still open.

7 Conclusions, related and future work

In this paper we have investigated a subclass of nondeterministic automata over infinite alphabets. This class naturally arises in the context of query learning, where automata have to be constructed from finitely many observations. Although there are many classes of data languages, we have shown that our class of residual languages admit canonical automata. The states of these automata correspond to join-irreducible elements.

In the context of learning, we show that convergence of standard Angluin-style algorithms is not guaranteed, even for residual languages. We propose a modified algorithm which guarantees convergence at the expense of an increase in the number of observations.

We emphasise that, unlike other algorithms based on residuality such as $\mathsf{NL}^\star$ [8] and $\mathsf{AL}^\star$ [2], our algorithm does not depend on the size, or even the existence, of the minimal deterministic automaton for the target language. This is a crucial difference, since dependence on the minimal deterministic automaton hinders generalisation to nondeterministic nominal automata, which are strictly more expressive. Ideally, in the residual case, one would like an algorithm for which the complexity depends only on the length of characterising words, which is an intrinsic feature of residual automata. To the best of our knowledge, no such algorithm exists in the finite setting.

[7] The set $\{w \in \Sigma^* \mid |w| = k\}$ is orbit-finite, for any fixed $k \in \mathbb{N}$.

We also show that universality is decidable for residual automata, in contrast to undecidability in the general nondeterministic case. As future work, we plan to attack the language inclusion/equivalence problem for residual automata. This is a well-known and challenging problem for data languages, which has been answered for specific subclasses [9, 10, 29, 34].

Of special interest is the subclass of *unambiguous automata* [10, 29]. We note that residual languages are orthogonal to unambiguous languages. For instance, the language $\mathcal{L}_{\mathrm{n}}$ is unambiguous but not residual, whereas $\mathcal{L}_{\mathrm{ng,r}}$ is residual but ambiguous. Moreover, their intersection has neither property, and every deterministic language has both properties. One interesting fact is that if a canonical residual automaton is unambiguous, then the join-irreducibles form an anti-chain.

Other related work are nominal languages/expressions with an explicit notion of binding [15, 25, 26, 34]. Although these are sub-classes of nominal languages, binding is an important construct, e.g., to represent resource-allocation. Availability of a notion of derivatives [25] suggests that residuality may prove beneficial for learning these languages.

Residual automata over finite alphabets also have a categorical characterisation [30]. We see no obstructions in generalising those results to nominal sets. This would amount to finding the right notion of nominal (complete) join-semilattice, with either finitely or uniformly supported joins.

Finally, in [16, 17] aspects of nominal lattices and Boolean algebras are investigated. To the best of our knowledge, our results of nominal lattice theory, especially those on join-irreducibles, are new.

References

1 Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

2 Dana Angluin, Sarah Eisenstat, and Dana Fisman. Learning regular languages via alternating automata. In *IJCAI*, pages 3308–3314. AAAI Press, 2015.

3 Sebastian Berndt, Maciej Liśkiewicz, Matthias Lutter, and Rüdiger Reischuk. Learning residual alternating automata. In *AAAI*, pages 1749–1755, 2017. URL: `http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14748`.

4 Mikołaj Bojańczyk. *Slightly Infinite Sets*. Draft September 6, 2019, 2019. URL: `https://www.mimuw.edu.pl/~bojan/paper/atom-book`.

5 Mikołaj Bojańczyk, Laurent Braud, Bartek Klin, and Slawomir Lasota. Towards nominal computation. In *POPL*, pages 401–412, 2012. `doi:10.1145/2103656.2103704`.

6 Mikołaj Bojańczyk, Bartek Klin, and Slawomir Lasota. Automata theory in nominal sets. *Logical Methods in Computer Science*, 10(3), 2014.

7 Mikołaj Bojańczyk and Szymon Toruńczyk. On computability and tractability for infinite sets. In *LICS*, pages 145–154. ACM, 2018.

8 Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-style learning of NFA. In *IJCAI*, pages 1004–1009, 2009.

9 Benedikt Bollig, Peter Habermehl, Martin Leucker, and Benjamin Monmege. A robust class of data languages and an application to learning. *Logical Methods in Computer Science*, 10(4), 2014. `doi:10.2168/LMCS-10(4:19)2014`.

10 Thomas Colcombet. Unambiguity in automata theory. In *Descriptional Complexity of Formal Systems - 17th International Workshop, DCFS 2015, Waterloo, ON, Canada, June 25-27, 2015. Proceedings*, pages 3–18, 2015. `doi:10.1007/978-3-319-19225-3_1`.

11 Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order, Second Edition*. Cambridge University Press, 2002. `doi:10.1017/CBO9780511809088`.

12 François Denis and Yann Esposito. Learning classes of probabilistic automata. In *COLT*, volume 3120 of *Lecture Notes in Computer Science*, pages 124–139. Springer, 2004.

13 François Denis and Yann Esposito. On rational stochastic languages. *Fundam. Inform.*, 86(1-2):41–77, 2008.

14 François Denis, Aurélien Lemay, and Alain Terlutte. Residual finite state automata. *Fundam. Inform.*, 51(4):339–368, 2002.

15 Murdoch James Gabbay and Vincenzo Ciancia. Freshness and name-restriction in sets of traces with names. In *FOSSACSs*, pages 365–380, 2011. doi:10.1007/978-3-642-19805-2_25.

16 Murdoch James Gabbay and Michael Gabbay. Representation and duality of the untyped λ-calculus in nominal lattice and topological semantics, with a proof of topological completeness. *Ann. Pure Appl. Logic*, 168(3):501–621, 2017. doi:10.1016/j.apal.2016.10.001.

17 Murdoch James Gabbay, Tadeusz Litak, and Daniela Petrişan. Stone duality for nominal boolean algebras with N. In *CALCO*, volume 6859 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2011.

18 Radu Grigore, Dino Distefano, Rasmus Lerchedahl Petersen, and Nikos Tzevelekos. Runtime verification based on register automata. In *TACAS*, volume 7795 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2013.

19 Falk Howar, Bengt Jonsson, and Frits W. Vaandrager. Combining black-box and white-box techniques for learning register automata. In *Computing and Software Science*, volume 10000 of *Lecture Notes in Computer Science*, pages 563–588. Springer, 2019.

20 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

21 Michael Kaminski and Daniel Zeitlin. Finite-memory automata with non-deterministic reassignment. *Int. J. Found. Comput. Sci.*, 21(5):741–760, 2010. doi:10.1142/S0129054110007532.

22 Michael J. Kearns and Umesh V. Vazirani. *An Introduction to Computational Learning Theory*. MIT Press, 1994. URL: https://mitpress.mit.edu/books/introduction-computational-learning-theory.

23 Bartek Klin and Michał Szynwelski. SMT solving for functional programming over infinite structures. In *MSFP*, volume 207 of *EPTCS*, pages 57–75, 2016.

24 Eryk Kopczynski and Szymon Toruńczyk. LOIS: syntax and semantics. In *POPL*, pages 586–598. ACM, 2017.

25 Dexter Kozen, Konstantinos Mamouras, Daniela Petrişan, and Alexandra Silva. Nominal kleene coalgebra. In *ICAL*, pages 286–298, 2015. doi:10.1007/978-3-662-47666-6_23.

26 Alexander Kurz, Tomoyuki Suzuki, and Emilio Tuosto. On nominal regular languages with binders. In *FOSSACS*, pages 255–269, 2012. doi:10.1007/978-3-642-28729-9_17.

27 Joshua Moerman. *Nominal Techniques and Black Box Testing for Automata Learning*. PhD thesis, Radboud University, Nijmegen, The Netherlands, 2019. URL: http://hdl.handle.net/2066/204194.

28 Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michał Szynwelski. Learning nominal automata. In *POPL*, pages 613–625. ACM, 2017.

29 Antoine Mottet and Karin Quaas. The containment problem for unambiguous register automata. In *STACS*, pages 53:1–53:15, 2019. doi:10.4230/LIPIcs.STACS.2019.53.

30 Robert S. R. Myers, Jirí Adámek, Stefan Milius, and Henning Urbat. Coalgebraic constructions of canonical nondeterministic automata. *Theor. Comput. Sci.*, 604:81–101, 2015.

31 Anil Nerode. Linear automaton transformations. *Proceedings of the AMS*, 9:541– 544, 1958.

32 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

33 Andrew M. Pitts. *Nominal sets: Names and symmetry in computer science*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2013.

34 Lutz Schröder, Dexter Kozen, Stefan Milius, and Thorsten Wißmann. Nominal automata with name binding. In *FOSSACS*, pages 124–142, 2017. doi:10.1007/978-3-662-54458-7_8.

35 Frits W. Vaandrager. Model learning. *Commun. ACM*, 60(2):86–95, 2017.

A Omitted proofs

▶ Remark 4.2. The set $\langle X \rangle$ is closed under the operation $\bigvee$, and moreover is the smallest equivariant set closed under $\bigvee$ containing X. In other words, $\langle - \rangle$ defines a closure operator. We will often say "X generates Y", by which we mean $Y \subseteq \langle X \rangle$.

Proof. Take any $\mathfrak{x} \subseteq \langle X \rangle$ finitely supported. All $x \in \mathfrak{x}$ are of the form $\bigvee \mathfrak{y}_x$, for some $\mathfrak{y}_x \subseteq X$ finitely supported. Consider the finitely supported set $T = \{y \mid y \in \mathfrak{y}_x, x \in \mathfrak{x}\} \subseteq X$. Then we see that $\bigvee \mathfrak{x} = \bigvee T \in \langle X \rangle$, meaning that $\langle X \rangle$ is closed under $\bigvee$. The second part of the claim is easy: any set closed under $\bigvee$ and containing X must also contain $\langle X \rangle$. ◀

▶ **Lemma 4.6.** *Let $X \subseteq \mathcal{P}_{\mathsf{fs}}(Z)$ be an orbit-finite and equivariant set.*

1. *Let $a \in X, b \in \mathcal{P}_{\mathsf{fs}}(Z)$ and $a \not\leq b$. Then there is a join-irreducible $x \in X$ such that $x \leq a$ and $x \not\leq b$.*
2. *Let $a \in X$, then $a = \bigvee \{x \in X \mid x \text{ join-irreducible in } X \text{ and } x \leq a\}$.*

Proof. In this proof we need a technicality. Let P be a finitely supported, non-empty poset (i.e., both P and $\leq$ are supported by a finite $A \subset \mathbb{A}$). If P is A-orbit-finite then P has a minimal element, as we can consider the finite poset of A-orbits and find a minimal A-orbit. Here we use the notion of an A-orbit, i.e., an orbit defined over permutations that fix A. (See [33, Chapter 5] for details.)

Ad 1. Consider the set $S = \{x \in X \mid x \leq a, x \not\leq b\}$. This is a finitely supported and $\mathsf{supp}(S)$-orbit-finite set, hence it has some minimal element $m \in S$. We shall prove that m is join-irreducible in X. Let $\mathfrak{x} \subseteq X$ finitely supported and assume that $x_0 < m$ for each $x_0 \in \mathfrak{x}$. Note that $x_0 < m \leq a$ and so that $x_0 \notin S$ (otherwise m was not minimal). Hence $x_0 \leq b$ (by definition of S). So $\bigvee \mathfrak{x} \leq b$ and so $\bigvee \mathfrak{x} \notin S$, which concludes that $\bigvee \mathfrak{x} \neq m$, and so $\bigvee \mathfrak{x} < m$ as required.

Ad 2. Consider the set $T = \{x \in \mathsf{JI}(X) \mid x \leq a\}$. This set is finitely supported, so we may define the element $b = \bigvee T \in \mathcal{P}_{\mathsf{fs}}(Z)$. It is clear that $b \leq a$, we shall prove equality by contradiction. Suppose $a \not\leq b$, then by *(1.)*, there is a join-irreducible x such that $x \leq a$ and $y \not\leq b$. By the first property of x we have $x \in T$, so that $x \not\leq b = \bigvee T$ is a contradiction. We conclude that $a = b$, i.e. $a = \bigvee T$ as required. ◀

▶ **Lemma 4.8.** *Let $Y \subseteq X \subseteq \mathcal{P}_{\mathsf{fs}}(Z)$ equivariant and suppose that $X \subseteq \langle \mathsf{JI}(Y) \rangle$. Then $\mathsf{JI}(Y) = \mathsf{JI}(X)$.*

Proof. ($\supseteq$) Let $x \in X$ be join-irreducible in X. Suppose that $x = \bigvee \mathfrak{y}$ for some finitely supported $\mathfrak{y} \subseteq Y$. Note that also $\mathfrak{y} \subseteq X$ Then $x = y_0$ for some $y_0 \in \mathfrak{y}$, and so x is join-irreducible in Y.

($\subseteq$) Let $y \in Y$ be join-irreducible in Y. Suppose that $y = \bigvee \mathfrak{x}$ for some finitely supported $\mathfrak{x} \subseteq X$. Note that every element $x \in \mathfrak{x}$ is a union of elements in $\mathsf{JI}(Y)$ (by the assumption $X \subseteq \langle \mathsf{JI}(Y_0) \rangle$). Take $\mathfrak{y}_x = \{y \in \mathsf{JI}(Y) \mid y \leq x\}$, then we have $x = \bigvee \mathfrak{y}_x$ and

$$y = \bigvee \mathfrak{x} = \bigvee \left\{ \bigvee \mathfrak{y}_x \mid x \in \mathfrak{x} \right\} = \bigvee \{y_0 \mid y_0 \in \mathfrak{y}_x, x \in \mathfrak{x}\}.$$

The last set is a finitely supported subset of Y, and so there is a y_0 in it such that $y = y_0$. Moreover, this y_0 is below some $x_0 \in \mathfrak{x}$, which gives $y_0 \leq x_0 \leq y$. We conclude that $y = x_0$ for some $x_0 \in \mathfrak{x}$. ◀

▶ **Corollary 4.11.** *The construction above defines a* canonical *residual automaton with the following uniqueness property: it has the minimal number of orbits of states and the maximal number of orbits of transitions.*

Proof. State minimality follows from Corollary 4.9, where we note that the states of any residual automata accepting $\mathcal{L}$ form a generating subset of $\mathsf{Der}(\mathcal{L})$. Maximality of transitions follows from the fact that it is *saturated*, meaning that no transitions can be added without changing the language. ◀

▶ **Example 4.13.** All the languages in the following ascending chain are join-irreducible.

$$\mathcal{L}_{\mathrm{ng}} < a^{-1}\mathcal{L}_{\mathrm{ng}} < ba^{-1}\mathcal{L}_{\mathrm{ng}} < cba^{-1}\mathcal{L}_{\mathrm{ng}} < \cdots$$

Proof. Consider the word $w = a_k \dots a_1 a_0$ with $k \geq 1$ and all a_i distinct atoms. We will prove that $w^{-1}\mathcal{L}_{\mathrm{ng}}$ is join-irreducible in $\mathsf{Der}(\mathcal{L}_{\mathrm{ng}})$, by considering all $u^{-1}\mathcal{L}_{\mathrm{ng}} \subseteq w^{-1}\mathcal{L}_{\mathrm{ng}}$.

Observe that if u is a suffix of w, then $u^{-1}\mathcal{L}_{\mathrm{ng}} \subseteq w^{-1}\mathcal{L}_{\mathrm{ng}}$. This is easily seen from the given automaton, since it may skip any prefix. We now show that u being a suffix of w is also a necessary condition.

First, suppose that u contains an atom a different from all a_i. If it is the last symbol of u, then $aaa_0 \in u^{-1}\mathcal{L}_{\mathrm{ng}}$, but $aaa_0 \notin w^{-1}\mathcal{L}_{\mathrm{ng}}$. If a is succeeded by b (not necessarily distinct), then either aa or aa_0 is in $u^{-1}\mathcal{L}_{\mathrm{ng}}$. But neither aa nor aa_0 is in $w^{-1}\mathcal{L}_{\mathrm{ng}}$. This shows that for $u^{-1}\mathcal{L}_{\mathrm{ng}} \subseteq w^{-1}\mathcal{L}_{\mathrm{ng}}$, we necessarily have that $u \in \{a_0, \dots, a_k\}^*$. (This also means that automatically $\mathsf{supp}(u^{-1}\mathcal{L}_{\mathrm{ng}}) \subseteq \mathsf{supp}(w^{-1}\mathcal{L}_{\mathrm{ng}})$.)

Second, when $u = \epsilon$, we have $u^{-1}\mathcal{L}_{\mathrm{ng}} \subseteq w^{-1}\mathcal{L}_{\mathrm{ng}}$. And for $|u| = 1$, if $u = a_0$, then $u^{-1}\mathcal{L}_{\mathrm{ng}} \subseteq w^{-1}\mathcal{L}_{\mathrm{ng}}$. If $u = a_i$ with $i > 0$, then $a_i a_i a_{i-1} \in u^{-1}\mathcal{L}_{\mathrm{ng}}$, but that word is not in $w^{-1}\mathcal{L}_{\mathrm{ng}}$. This shows that for $u^{-1}\mathcal{L}_{\mathrm{ng}} \subseteq w^{-1}\mathcal{L}_{\mathrm{ng}}$ with $|u| \leq 1$, we necessarily have that u is a suffix of w.

Third, we prove the same for $|u| \leq 2$. We first consider which bigrams may occur in u. Suppose that u contains a bigram $a_i a_j$ with $i > 0$ and $j \neq i - 1$. Then $a_i a_{i-1}$ is in $u^{-1}\mathcal{L}_{\mathrm{ng}}$, but not in $w^{-1}\mathcal{L}_{\mathrm{ng}}$. Suppose that u contains $a_0 a_i$ $(i > 0)$ or $a_0 a_0$, then $u^{-1}\mathcal{L}_{\mathrm{ng}}$ contains either $a_0 a_0$ or $a_0 a_1$ respectively. Neither of these words are in $w^{-1}\mathcal{L}_{\mathrm{ng}}$. This shows that $u^{-1}\mathcal{L}_{\mathrm{ng}} \subseteq w^{-1}\mathcal{L}_{\mathrm{ng}}$ implies that u may only contain the bigrams $a_i a_{i-1}$. In particular, these bigrams compose in a unique way. So u is a (contiguous) subword of w, whenever $u^{-1}\mathcal{L}_{\mathrm{ng}} \subseteq w^{-1}\mathcal{L}_{\mathrm{ng}}$.

Continuing, suppose that u ends in the bigram $a_{i+1}a_i$ with $i > 0$. Then we have $a_i a_i a_{i-1}$ in $u^{-1}\mathcal{L}_{\mathrm{ng}}$, but not in $w^{-1}\mathcal{L}_{\mathrm{ng}}$. This shows that u has to end in $a_1 a_0$. That is, for $u^{-1}\mathcal{L}_{\mathrm{ng}} \subseteq w^{-1}\mathcal{L}_{\mathrm{ng}}$ with $|u| \geq 2$, we necessarily have that u is a suffix of w.

So far, we have shown that

$$\{u \mid u^{-1}\mathcal{L}_{\mathrm{ng}} \subseteq w^{-1}\mathcal{L}_{\mathrm{ng}}\} = \{u \mid u \text{ is a suffix of } w\}.$$

To see that $w^{-1}\mathcal{L}_{\mathrm{ng}}$ is indeed join-irreducible, we consider the join $X = \bigvee\{u^{-1}\mathcal{L}_{\mathrm{ng}} \mid u \text{ is a strict suffix of } w\}$. Note that $a_k a_k \notin X$, but $a_k a_k \in w^{-1}\mathcal{L}_{\mathrm{ng}}$. We conclude that $w^{-1}\mathcal{L}_{\mathrm{ng}} \neq \bigvee\{u^{-1}\mathcal{L}_{\mathrm{ng}} \mid u^{-1}\mathcal{L}_{\mathrm{ng}} \subsetneq w^{-1}\mathcal{L}_{\mathrm{ng}}\}$ as required. ◀

▶ **Proposition 5.1.** *Universality for residual nominal automata is decidable. Formally: given a residual automaton $\mathcal{A}$, it is decidable whether $\mathcal{L}(\mathcal{A}) = \Sigma^*$.*

Proof. In the constructions below, we use *computation with atoms*. This is a computation paradigm which allow algorithmic manipulation of infinite – but orbit-finite – nominal sets. For instance, it allows looping over such a set in finite time. Important here is that this paradigm is equivalent to regular computability (see [7]) and implementations exist to compute with atoms [23, 24].

We will sketch an algorithm that, given a residual automaton $\mathcal{A}$, answers whether $\mathcal{L}(\mathcal{A}) = \Sigma^*$. The algorithm decides *negatively* in the following cases:

- $I = \emptyset$. In this case the language accepted by $\mathcal{A}$ is empty.
- Suppose there is a $q \in Q$ with $q \notin F$. By residuality we have $\mathcal{L}(q) = w^{-1}\mathcal{L}(\mathcal{A})$ for some w. Note that q is not accepting, so that $\epsilon \notin w^{-1}\mathcal{L}(\mathcal{A})$. Put differently: $w \notin \mathcal{L}(\mathcal{A})$. (We note that w is not used by the algorithm. It is only needed for the correctness.)
- Suppose there is a $q \in Q$ and $a \in \Sigma$ such that $\delta(q, a) = \emptyset$. Again $\mathcal{L}(q) = w^{-1}\mathcal{L}(\mathcal{A})$ for some w. Note that a is not in $\mathcal{L}(q)$. This means that wa is not in the language.

When none of these three cases hold, the algorithm decides *positively*. We shall prove that this is indeed the correct decision. If none of the above conditions hold, then $I \neq \emptyset$, $Q = F$, and for all $q \in Q, a \in \Sigma$ we have $\delta(q, a) \neq \emptyset$. Here we can prove that the language of each state is $\mathcal{L}(q) = \Sigma^*$. Given that there is an initial state, the automaton accepts Σ^*.

Note that the operations on sets performed in the above cases all terminate, because all involve orbit-finite sets. ◀

▶ **Proposition 5.2.** *The problem of determining whether a given nondeterministic nominal automaton is residual is undecidable.*

Proof. The construction is inspired by [14, Proposition 8.4].[8] We show undecidability by reducing the universality problem for nominal automata to the residuality problem.

Let $\mathcal{A} = (\Sigma, Q, I, F, \delta)$ be a nominal (nondeterministic) automaton on the alphabet Σ. We first extend the alphabet:

$$\Sigma' = \Sigma \cup \{\underline{q} \mid q \in Q\} \cup \{\overline{q} \mid q \in Q\} \cup \{\$, \#\},$$

where we assume the new symbols to be disjoint from Σ. We define $\mathcal{A}' = (\Sigma', Q', I', F', \delta')$ by

$$\begin{aligned}
Q' &= \{\overline{q} \mid q \in Q\} \cup \{\underline{q} \mid q \in Q\} \cup \{\top, x, y\} \\
I' &= \{\underline{q} \mid q \in Q\} \cup \{x, y\} \\
F' &= \{\overline{q} \mid q \in F\} \cup \{\top\} \\
\delta' &= \{(\overline{q}, a, \overline{q}' \mid (q, a, q') \in \delta\} \cup \{(\underline{q}, \underline{q}, \underline{q}) \mid q \in Q\} \cup \{(\underline{q}, \overline{q}, \overline{q}) \mid q \in Q\} \\
&\quad \cup \{(x, \$, \top), (x, \#, x), (y, \#, y)\} \cup \{(\top, a, \top) \mid a \in \Sigma\} \cup \{(y, \$, \overline{i}) \mid i \in I\}
\end{aligned}$$

See Figure 7 for a sketch of the automaton $\mathcal{A}'$. The blue part is a copy of the original automaton. The red part forces the original states to be residual, by providing anchors to each state. Finally the orange part is the interesting part. The key players are states x and y with their languages $\mathcal{L}(y) \subseteq \mathcal{L}(x)$. Note that their languages are equal if and only if $\mathcal{A}$ is universal.

Before we assume anything about $\mathcal{A}$, let us analyse $\mathcal{A}'$. In particular, let us consider whether the residuality property holds for each state. For the original states of $\mathcal{A}$ the property holds, as we can provide anchors: All the states $\overline{q}$ and $\underline{q}$ are anchored by the words $\overline{q}$ and $\underline{q}$ respectively. Then we consider the states x and $\top$, their languages are $\mathcal{L}(\top) = \Sigma^* = \$^{-1}\mathcal{L}(\mathcal{A}')$ and $\mathcal{L}(x) = \#^{-1}\mathcal{L}(\mathcal{A}')$ (see Figure 7). The only remaining state for which we do not yet know whether the residuality property holds is state y.

If $\mathcal{L}(\mathcal{A}) = \Sigma^*$ (i.e. the original automaton is universal), then we note that $\mathcal{L}(y) = \mathcal{L}(x)$. In this case, $\mathcal{L}(y) = \#^{-1}\mathcal{L}(\mathcal{A}')$. So, in this case, $\mathcal{A}'$ is residual.

[8] They prove that checking residuality for NFAs is PSPACE-complete via a reduction from universality. Instead of using NFAs, they use a union of n DFAs. This would not work in the nominal setting.

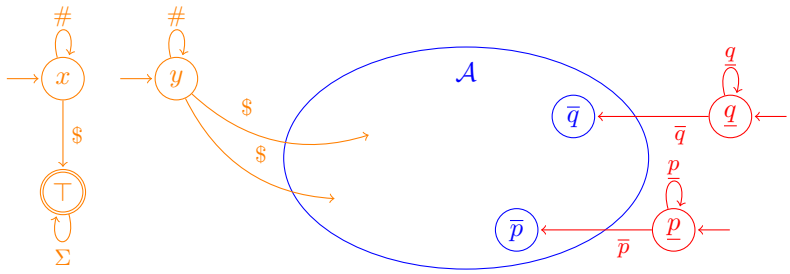


Figure 7 Sketch of the automaton $\mathcal{A}'$ constructed in the proof of Proposition 5.2.

Suppose that $\mathcal{A}'$ is residual. Then $\mathcal{L}(y) = w^{-1}\mathcal{L}'$ for some word w. Provided that $\mathcal{L}(\mathcal{A})$ is not empty, there is some $u \in \mathcal{L}(\mathcal{A})$. So we know that $\$u \in \mathcal{L}(y)$. This means that word w cannot start with $a \in \Sigma$, $\overline{q}$, $\underline{q}$ for $q \in Q$, or \$ as their derivatives do not contain $\$u$. The only possibility is that $w = \#^k$ for some $k > 0$. This implies $\mathcal{L}(y) = \mathcal{L}(x)$, meaning that the language of $\mathcal{A}$ is universal.

This proves that $\mathcal{A}$ is universal iff $\mathcal{A}'$ is residual. Moreover, the construction $\mathcal{A} \mapsto \mathcal{A}'$ is effective, as it performs computations with orbit-finite sets. ◀

▶ **Theorem 6.4.** *There is an algorithm which query learns residual nominal languages.*

Proof. As explained in the text, we modify the $\nu\mathsf{NL}^\star$ algorithm from [28]: When the table is not closed, we not only add the missing words, but all the words of the same length. This guarantees that the algorithm finds rows for all join-irreducible derivatives, i.e., all states of the canonical residual automaton.

The pseudocode is given in Algorithm 1, where the modifications to $\nu\mathsf{NL}^\star$ are highlighted in red. We briefly explain the notation. An observation table $\mathcal{T}$ is defined by a set of row (resp. column) indices S (resp. E). The value $\mathcal{T}(s, e)$ is given by $\mathcal{L}(se)$ (we may do this via membership queries). We denote the set of rows by $\mathrm{Rows}(S, E) \coloneqq \{\mathrm{row}(s) \mid s \in S\Sigma \cup S\}$,

Algorithm 1 Modified nominal $\mathsf{NL}^\star$ algorithm for Theorem 6.4.

MODIFIED $\nu\mathsf{NL}^\star$ LEARNER

$S, E = \{\epsilon\}$
repeat
  while (S, E) is not residually-closed or not residually-consistent
  if (S, E) is not residually-closed
    find $s \in S, a \in A$ such that $row(sa) \in \mathsf{JI}(\mathrm{Rows}(S, E)) \setminus \mathrm{Rows}^\top(S, E)$
    $k =$ length of the word sa
    $S = S \cup \Sigma^{\leq k}$
  if (S, E) is not residually-consistent
    find $s_1, s_2 \in S$, $a \in A$, and $e \in E$ such that $\mathrm{row}(s_1) \sqsubseteq \mathrm{row}(s_2)$ and $\mathcal{L}(s_1ae) = 1$, $\mathcal{L}(s_2ae) = 0$
    $E = E \cup \mathsf{orb}(ae)$
  Make the conjecture $N(S, E)$
  if the Teacher replies **no**, with a counter-example t
    $E = E \cup \{\mathsf{orb}(t_0) \mid t_0$ is a suffix of $t\}$
until the Teacher replies **yes** to the conjecture $N(S, E)$.
return $N(S, E)$

where $\mathrm{row}(s)(e) = \mathcal{T}(s, e)$. Note that $\mathrm{Rows}(S, E)$ also includes rows for one-letter extensions. The set of rows labelled by S is denoted by $\mathrm{Rows}^{\top}(S, E) \coloneqq \{\mathrm{row}(s) \mid s \in S\}$. The set $\mathrm{Rows}(S, E)$ is a poset, ordered by $r_1 \sqsubseteq r_2$ iff $r_1(e) \leq r_2(e)$ for all $e \in E$. To construct a hypothesis $N(S, E)$, we use the construction from Theorem 4.10, where $\mathrm{Rows}(S, E)$ plays the role of $\mathsf{Der}(\mathcal{L})$.

We can give a bound to the number of equivalence queries. Given an orbit-finite nominal set X, let $|X|$ be the number of its orbit. Then equivalence queries are bounded by $\mathcal{O}(m + |\Sigma^{\leq m+1}| \times k)$, where m is the length of the longest characterising word and k is the maximum support size of the canonical residual automaton. Intuitively, each of the rows in the table could be a separate state, and for each state there is some work to be done, concerning learning the right support and local symmetries (see [27] for details on this). ◀

Flatness and Complexity of Immediate Observation Petri Nets

Mikhail Raskin
Technical University of Munich, Germany
raskin@in.tum.de

Chana Weil-Kennedy
Technical University of Munich, Germany
chana.weilkennedy@in.tum.de

Javier Esparza
Technical University of Munich, Germany
esparza@in.tum.de

Abstract

In a previous paper we introduced immediate observation (IO) Petri nets, a class of interest in the study of population protocols and enzymatic chemical networks. In the first part of this paper we show that IO nets are globally flat, and so their safety properties can be checked by efficient symbolic model checking tools using acceleration techniques, like FAST. In the second part we study Branching IO nets (BIO nets), whose transitions can create tokens. BIO nets extend both IO nets and communication-free nets, also called BPP nets, a widely studied class. We show that, while BIO nets are no longer globally flat, and their sets of reachable markings may be non-semilinear, they are still locally flat. As a consequence, the coverability and reachability problem for BIO nets, and even a certain set-parameterized version of them, are in PSPACE. This makes BIO nets the first natural net class with non-semilinear reachability relation for which the reachability problem is provably simpler than for general Petri nets.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Concurrency

Keywords and phrases Petri Nets, Reachability Analysis, Parameterized Verification, Flattability

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.45

Funding This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 787367 (PaVeS).

Acknowledgements We thank Jérôme Leroux and Rupak Majumdar for interesting conversations that put us on the path of flatness and BIO nets. We also thank the reviewers whose comments allowed us to improve this paper, and fix a small mistake in Lemma 18.

1 Introduction

Immediate observation Petri nets (IO nets) model immediate observation population protocols, as introduced by Angluin *et al.* in their seminal paper on the expressive power of population protocols [2]. In an IO net each transition is defined by three places: the source place p_s, the destination place p_d, and the observed place p_o. The transition can move one token from p_s to p_d, provided that p_o is not empty (if $p_s = p_o$, then p_o should contain at least two tokens). In the population protocol interpretation, p_s, p_d, and p_o are three possible states of each of the identical agents executing the protocol, and a transition models an agent in the state p_s observing another agent in the state p_o and switching to the state p_d.

In a previous paper [10] we investigated "many-to-many" versions of the reachability and coverability problems for IO nets, in which we have a set of initial markings and a set of final markings instead of the standard "one-to-one" versions with a single initial marking and a single final marking. The sets we consider are *cubes*, i.e., sets of markings obtained by attaching to each place a lower bound and an upper bound (possibly infinite) for the number of tokens. We showed that while the standard one-to-one problems are PSPACE-hard, they remain in PSPACE in the many-to-many case. This is in strong contrast with general conservative Petri nets (nets in which transitions neither create nor destroy tokens), for which the many-to-many versions of the problems become EXPSPACE-hard or even non-elementary.

In this paper we continue our study of IO nets, and initiate the study of Branching IO nets (BIO nets for short), in which transitions can create or destroy agents. BIO nets deserve study for at least three reasons:

- They are a natural generalization of both IO nets and communication-free nets (aka BPP nets), another very well studied subclass (see e.g. [8, 9, 17, 11, 14, 13, 16]).
- The reachability sets of BIO nets are not necessarily semilinear. In particular, Hopcroft and Pansiot's well-known example of a Petri net with a non-semilinear reachability set (see [12]) is a BIO net. The classes of unbounded Petri nets for which the reachability problem is demonstrably simpler than for arbitrary Petri nets, like BPP-nets, reversible nets, and IO nets, have semilinear reachability sets. This makes BIO nets ideal to investigate the existence of efficient verification techniques that do not depend on semilinearity.
- BIO nets are a natural model for enzymatic catalytic reactions of the form $A + C \to C + B_1 + \cdots + B_n$ with more than one product. For example, catalase degrades hydrogen peroxide into water and oxygen, a reaction of the form $A + C \to C + B_1 + B_2$ [7]. Since IO nets have been used to model and analyze enzymatic reactions $A + C \to C + B$ (see [1, 4, 15]), we expect our results to find a similar application.

In this paper we prove that IO nets are globally flat, in the sense of Leroux and Sutre [14]. In particular, this shows that their reachability relation is semilinear. Since the reachability relation of BIO nets is not semilinear, this result cannot extend to BIO nets. However, we prove that they are locally pre^*-flat, also in the sense of [14] [1]. Both global and local flatness allow us to analyze nets applying existing symbolic model checking tools like FAST [5], LASH [6] and TREX [3]. Further, we prove that the many-to-many versions of the reachability and coverability problems for BIO nets are still PSPACE-complete, as for IO nets. To the best of our knowledge, this makes BIO nets the first natural class of nets whose reachability relation is non-semilinear for which these problems have elementary complexity.

Our flatness and complexity results are consequences of two theorems, called the Shortening Theorems for IO and BIO nets. They state that if M is reachable from M', then M can be reached by a sequence of bounded *accelerated* length, defined as the length of the sequence after exhaustively replacing any subsequence of the form tt by t. In the case of IO nets the accelerated length is independent of the initial and final markings, while for BIO nets it only depends on the final marking. We consider that the Shortening Theorems are also interesting in their own right.

The paper is organized as follows. Section 2 contains preliminaries, and Section 3 defines IO and BIO nets. Section 4 states the Shortening Theorems, and derives our flatness and complexity results for the one-to-one reachability and coverability problems as corollaries.

[1] Actually, the locally flat of [14] are what we call locally $post^*$-flat. A net is locally pre^*-flat iff its reverse net is locally $post^*$-flat, and so with respect to reachability questions the difference is immaterial.

The proof of the Shortening Theorem for IO nets is given in Section 5 and our main result, the Shortening Theorem for BIO nets, is proved in Section 6. Finally, we prove in Section 7 that the many-to-many reachability and coverability problems remain in PSPACE.

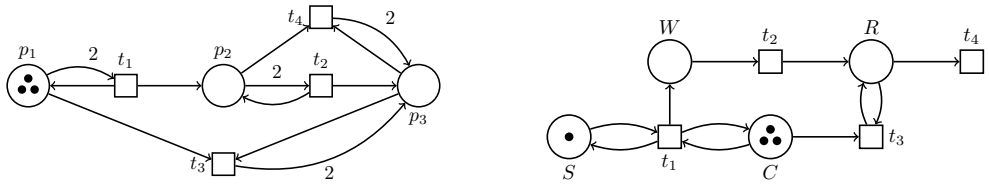
2 Preliminaries

Multisets. A *multiset* on a finite set E is a mapping $C\colon E \to \mathbb{N}$, i.e. for any $e \in E$, $C(e)$ denotes the number of occurrences of element e in C. Let $\wr e_1, \ldots, e_n \wr$ denote the multiset C such that $C(e) = |\{j \mid e_j = e\}|$. Operations on $\mathbb{N}$ like addition or comparison are extended to multisets by defining them component wise on each element of E. Subtraction is allowed in the following way: if C, D are multisets on set E then for all $e \in E$, $(C - D)(e) = \max(C(e) - D(e), 0)$. We call $|C| \stackrel{\text{def}}{=} \sum_{e \in E} C(e)$ the *size* of C, and $\|C\| \stackrel{\text{def}}{=} \{e \mid C(e) > 0\}$ the *support* of C. Given a total order $e_1 \prec e_2 \prec \cdots \prec e_n$ on E, a multiset C can be equivalently represented by the vector $(C(e_1), \ldots, C(e_n)) \in \mathbb{N}^n$. A set $V \subseteq \mathbb{N}^n$ is *linear* if there is a root $\boldsymbol{r} \in \mathbb{N}^n$ and a set $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n\}$ of periods such that $V = \{v + \sum_{i=1}^{n} \lambda_i \boldsymbol{p}_i \mid \lambda_1, \ldots, \lambda_n \in \mathbb{N}\}$, and *semilinear* if it is the union of a finite set of linear sets. A relation on $\mathbb{N}^n$ is semilinear if it is semilinear as a set of $\mathbb{N}^{2n}$. All these notions extend to sets of multisets.

Place/transition Petri nets with weighted arcs. A *Petri net* N is a triple (P, T, F) consisting of a finite set of *places* P, a finite set of *transitions* T and a *flow function* $F\colon (P \times T) \cup (T \times P) \to \mathbb{N}$. A *marking* M is a multiset on P, and we say that a marking M puts $M(p)$ *tokens* in place p of P. The *size* of M, denoted by $|M|$, is the total number of tokens in M. The *preset* ${}^\bullet t$ and *postset* $t^\bullet$ of a transition t are the multisets on P given by ${}^\bullet t(p) = F(p, t)$ and $t^\bullet(p) = F(t, p)$. A transition t is *enabled* at a marking M if ${}^\bullet t \leq M$, i.e. ${}^\bullet t$ is component-wise smaller or equal to M. If t is enabled then it can be *fired*, leading to a new marking $M' = M - {}^\bullet t + t^\bullet$. We let $M \xrightarrow{t} M'$ denote this.

Reachability and coverability. Given $\sigma = t_1 \ldots t_n$ we write $M \xrightarrow{\sigma} M_n$ when $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \ldots \xrightarrow{t_n} M_n$, and call σ a *firing sequence*. We write $M' \xrightarrow{*} M''$ if $M' \xrightarrow{\sigma} M''$ for some $\sigma \in T^*$, and say that M'' is *reachable* from M'. A marking M *covers* another marking M', written $M \geq M'$ if $M(p) \geq M'(p)$ for all places p. A marking M is *coverable* from M' if there exists a marking M'' such that $M' \xrightarrow{*} M'' \geq M$. The *reachability relation* is the set of pairs of markings (M, M') such that $M \xrightarrow{*} M'$, and we denote it $\xrightarrow{*}$. The sets of predecessors and successors of a set $\mathcal{M}$ of markings of N are $pre^*(\mathcal{M}) \stackrel{\text{def}}{=} \{M' | \exists M \in \mathcal{M} \,.\, M' \xrightarrow{*} M\}$ and $post^*(\mathcal{M}) \stackrel{\text{def}}{=} \{M | \exists M' \in \mathcal{M} \,.\, M' \xrightarrow{*} M\}$, respectively.

Global and local flatness. A net $N = (P, T, F)$ is *globally flat* if there exist transition words $w_1, w_2, \ldots, w_k \in T^*$ such that for every two markings M', M, if $M' \xrightarrow{*} M$, then there exist $j_1, \ldots, j_k \geq 0$ satisfying $M' \xrightarrow{w_1^{j_1} \ldots w_k^{j_k}} M$. Observe that the words $w_1, w_2, \ldots, w_k$ are independent of both M and M'. A net $N = (P, T, F)$ N is *locally pre*-flat* (resp. *locally post*-flat*) if for every M (resp. M') there exist transition words $w_1, w_2, \ldots, w_k \in T^*$ such that for every M' (resp. M) satisfying $M' \xrightarrow{*} M$ there exist $j_1, \ldots, j_k \geq 0$ such that $M' \xrightarrow{w_1^{j_1} \ldots w_k^{j_k}} M$. The locally flat Petri nets of [14] correspond to our *post**-flat nets.



(a) An IO net. **(b)** A BIO net.

Figure 1 Examples of IO and BIO nets.

3 Immediate Observation and Branching Immediate Observation Nets

We recall the definition of immediate observation nets (IO nets), as introduced in [10], and extend it to branching immediate observation nets (BIO nets).

▶ **Definition 1.** *A transition t of a Petri net is an* immediate observation transition *(IO transition) if there are places p_s, p_d, p_o, not necessarily distinct, such that ${}^\bullet t = \{\!|p_s, p_o|\!\}$ and $t^\bullet = \{\!|p_d, p_o|\!\}$. We call p_s, p_d, p_o the* source, destination, *and* observed *places of t, respectively. A Petri net is an* immediate observation net *(IO net) if all its transitions are IO transitions.*

A transition t of a Petri net is a branching IO transition *(BIO transition) if there is $k \geq 0$ and places $p_s, p_{d_1}, \ldots, p_{d_k}, p_o$, not necessarily distinct, such that ${}^\bullet t = \{\!|p_s, p_o|\!\}$ and $t^\bullet = \{\!|p_{d_1}, \ldots, p_{d_k}, p_o|\!\}$. A Petri net is a* branching IO net *(BIO net) if all its transitions are BIO transitions.*

In the following examples, we allow ourselves to consider IO and BIO nets containing transitions with no observed place. To make the net a formally correct IO or BIO net, it suffices to add an extra marked place which acts as observed place for these transitions.

▶ **Example 2.** Figure 1a shows an IO net taken from the literature on population protocols [2]. Intuitively, it models a protocol allowing a crowd of undistinguishable agents that can only interact in pairs to decide whether they are at least 3. Initially all agents are in state p_1, modelled by tokens in place p_1. If two agents in state p_1 interact, one of them moves to state p_2 (transition t_1). If two agents in state p_2 interact, one of them moves to p_3 (transition t_2). Finally, an agent in state p_3 can "attract" all other agents to state p_3 (transitions t_3 and t_4). Given a marking M_0 with tokens only in p_1, if $M_0(p_1) \geq 3$ and the pairs of tokens that interact next are chosen uniformly at random, then eventually all tokens reach p_3.

Figure 1b shows a BIO net representing a client server interaction. If the server S observes a client C, it creates a worker W, which creates a response R and terminates. The client C "leaves" after observing a response. Responses may expire.

IO nets are *conservative*, i.e. there is no creation or destruction of tokens, while BIO nets are not. The next example, taken from [12], shows that BIO nets may have non-semilinear sets of reachable markings.

▶ **Example 3** ([12])**.** Consider the BIO net N of Figure 2, with states p, q, c_1, c_2, c_3 and initial marking $M_0 = (1, 0, 0, 0, 1)$. The set of markings reachable from M_0 in N is characterized by the condition $(\boldsymbol{p} = 1 \wedge \boldsymbol{q} = 0 \wedge 0 < \boldsymbol{c_2} + \boldsymbol{c_3} \leq 2^{\boldsymbol{c_1}}) \vee (\boldsymbol{p} = 0 \wedge \boldsymbol{q} = 1 \wedge 0 < 2\boldsymbol{c_2} + \boldsymbol{c_3} \leq 2^{\boldsymbol{c_1}+\boldsymbol{1}})$, where $\boldsymbol{c}$ denotes the number of tokens in some place c. Informally, one token cycles between p and q, putting a new token in c_1 at every new cycle. When p is marked, tokens in c_3 can move to c_2, and when q is marked, tokens in c_2 can move to c_3 while doubling their number (see Lemma 2.8 of [12]). Clearly the reachability relation of this BIO net is not semilinear.

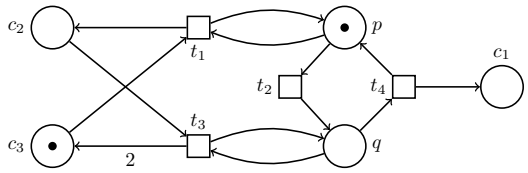


Figure 2 A non-flat BIO net.

4 Shortening Theorems

We introduce the main results of our paper, called the Shortening Theorems. We use them to prove flatness results for IO and BIO nets, and to extend complexity results of [10] for the reachability and coverability problems of IO nets to the (much harder) case of BIO nets. The Shortening Theorems themselves are proved in Sections 5 and 6, respectively.

First, we introduce a measure of the length of firing sequences that abstracts from the number of times a transition is consecutively executed.

▶ **Definition 4.** *Let N be a Petri net, and let σ be a firing sequence. Let $k_1, \ldots, k_m$ be the unique positive natural numbers such that $\sigma = t_1^{k_1} t_2^{k_2} \ldots t_m^{k_m}$ and $t_i \neq t_{i+1}$ for every $i = 1, \ldots, m-1$. We say that σ has* accelerated length *m, and let $|\sigma|_a$ denote the accelerated length of σ.*

The Shortening Theorems for IO and BIO show that a firing sequence leading from M' to M can be shortened to a sequence of bounded accelerated length. For IO nets the bound only depends on the net, not on the markings M or M':

▶ **Theorem 5** (IO Shortening). *Let N be an IO net with n places, and let M', M be two markings of N. If $M' \xrightarrow{*} M$, then $M' \xrightarrow{\sigma} M$ for some σ of accelerated length $|\sigma|_a \leq (n^3+1)^n$.*

Example 3 shows that for BIO nets the bound cannot be independent of both M and M':

▶ **Example 6.** Recall the BIO net of Example 3 with states p, q, c_1, c_2, c_3. It is easy to see that for $j \geq 1$ the marking $M_j \stackrel{\text{def}}{=} (1, 0, j, 0, 2^j)$ is reachable only via the firing sequence

$$(t_1 t_2 t_3 t_4)(t_1^2 t_2 t_3^2 t_4) \ldots (t_1^i t_2 t_3^i t_4) \ldots (t_1^j t_2 t_3^j t_4).$$

This sequence has accelerated length $4j$, which depends on the target marking M_j.

However, we can still obtain a bound independent of M':

▶ **Theorem 7** (BIO Shortening). *Let N be a BIO net with n places, let M', M be two markings of N, and let $|M'| = m'$, $|M| = m$. Let $m_d := \max_{t \in T} |t^\bullet - {}^\bullet t|$ denote the maximum number of tokens created by a transition of N. If $M' \xrightarrow{*} M$, then $M' \xrightarrow{\sigma} M$ for some σ of accelerated length $|\sigma|_a \leq 2^n (m+1)^n (n+1)^n$. Further, the intermediate markings along σ have size at most $(m' + 2^n (m+1)^n (n+1)^n (m+n) m_d) m_d^n$.*

4.1 Flatness and complexity results

The Shortening Theorems lead easily to our flatness and complexity results:

▶ **Theorem 8.** *IO nets are globally flat. BIO nets are locally pre*-flat, but neither globally flat nor locally post*-flat.*

Proof.

(a) We show that IO nets are globally flat. Let $N = (P, T, F)$ be an IO net with n places and $T = \{t_1, \ldots, t_m\}$, and let $K = (n^3 + 1)^n$. By Theorem 5, for every two markings M' and M of N there is a firing sequence $t_{i_1}^{j_1} \cdots t_{i_K}^{j_K}$ leading from M' to M. Since every such sequence belongs to the regular language $(t_1^* t_2^* \cdots t_m^*)^K$, the words $w_1, w_2, \ldots, w_{m \cdot K}$ given by $w_i = t_{((i-1) \bmod m)+1}$ for every $1 \leq i \leq m \cdot K$ witness that N is globally flat.

(b) We show that BIO nets are locally pre^*-flat. Let $N = (P, T, F)$ be a BIO net with n places and $T = \{t_1, \ldots, t_m\}$, let M be a marking of N with $|M| = m$, and let $K = 2^n(m+1)^n(n+1)^n$. By Theorem 7, for every marking M' of N there is a firing sequence $t_{i_1}^{j_1} \cdots t_{i_K}^{j_K}$ leading from M' to M. Proceed now as for (a).

(c) We show that BIO nets are not locally $post^*$-flat, and so also not globally flat. Consider the BIO net of Figure 2 with states p, q, c_1, c_2, c_3. Recall that for all $j \geq 1$, M_0 only reaches the marking $M_j \stackrel{\text{def}}{=} (1, 0, j, 0, 2^j)$ via $(t_1 t_2 t_3 t_4)(t_1^2 t_2 t_3^2 t_4) \ldots (t_1^i t_2 t_3^i t_4) \ldots (t_1^j t_2 t_3^j t_4)$. So in order to reach M_j it is necessary to fire j times a sequence of the form $t_1^k t_2^{k_2} t_3^k t_4^{k_4}$, which proves the result. ◀

▶ **Theorem 9.** *The reachability and coverability problems for BIO nets are* PSPACE*-complete.*

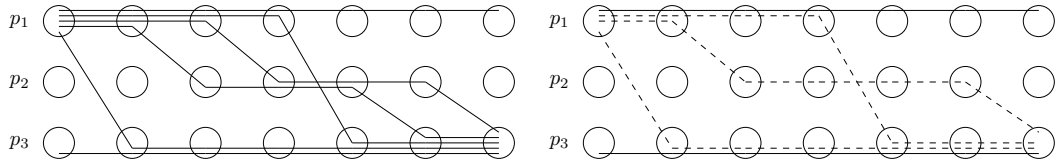
Proof. Reachability and coverability are PSPACE-complete for IO nets [10], and IO nets are a subclass of BIO nets, so the problems stay PSPACE-hard for BIO nets. By Savitch's theorem it suffices to show that the problems are in NPSPACE. Consider first the reachability problem. By the Shortening Theorem, given a BIO net with n places and two markings M and M' we can guess a firing sequence leading from M to M', if one exists, using space $\log(f(n, m, m', m_d))$, where $f(n, m, m', m_d)$ is the exponential bound of the Shortening Theorem. So the reachability problem is in NPSPACE. For coverability, we reduce it to reachability in the usual way. Let M be the marking we want to cover. For each place p, we add a "destroying transition" τ_p with preset ${}^\bullet t = \{p\}$ and postset $t^\bullet = \emptyset$. It is easy to see that for every marking M', the modified net N' has a firing sequence from M' to M iff N has a firing sequence from M' to some marking covering M. ◀

5 Shortening Theorem for IO nets

The proof of Theorem 5 is based on a result of [10] called the Pruning Lemma. We briefly introduce some notions required to state the lemma, and then the lemma itself. More details can be found in [10].

Trajectories and histories. Since the transitions of IO nets do not create or destroy tokens, we can give tokens identities. Given a firing sequence, each token of the initial marking follows a *trajectory* through the places of the net until it reaches the final marking of the sequence. The trajectories of the tokens between given source and target markings constitute a *history*.

Fix an IO net N. A *trajectory* of an IO net N is a sequence $\tau = p_1 \ldots p_k$ of places. We let $\tau(i)$ denote the i-th place of τ. The *i-th step* of τ is the pair $\tau(i)\tau(i+1)$. A *history* H of length h is a multiset of trajectories of length h. Given an index $1 \leq i \leq h$, *the i-th marking of H*, denoted M_H^i, is defined as follows: for every place p, $M_H^i(p)$ is the number of trajectories $\tau \in H$ such that $\tau(i) = p$. The markings M_H^1 and M_H^h are the *initial* and *final* markings of H, and we write $M_H^1 \xrightarrow{H} M_H^h$. A history H of length $h \geq 1$ is *realizable* if there exist transitions $t_1, \ldots, t_{h-1}$ and numbers $k_1, \ldots, k_{h-1} \geq 0$ such that



(a) A realizable history. **(b)** Pruning of the history on the left.

Figure 3 A realizable history of the IO net of Figure 1a before and after pruning.

- $M_H^1 \xrightarrow{t_1^{k_1}} M_H^2 \cdots M_H^{h-1} \xrightarrow{t_{h-1}^{k_{h-1}}} M_H^h$, where for every t we define $M' \xrightarrow{t^0} M$ iff $M' = M$.
- For every $1 \leq i \leq h-1$, there are exactly k_i trajectories $\tau \in H$ such that $\tau(i)\tau(i+1) = p_s p_d$, where p_s, p_d are the source and target places of t_i, and all other trajectories $\tau \in H$ satisfy $\tau(i) = \tau(i+1)$. Moreover, there is at least one trajectory τ in H such that $\tau(i)\tau(i+1) = p_o p_o$, where p_o is the observed place of t_i.

We say that $t_1^{k_1} \cdots t_{h-1}^{k_{h-1}}$ realizes H. Intuitively, at a step of a realizable history only one transition occurs, although perhaps multiple times, for different tokens. From the definition of realizable history we immediately obtain:

- $M' \xrightarrow{*} M$ iff there exists a realizable history with M' and M as initial and final markings.
- Every firing sequence that realizes a history of length h has accelerated length at most h.

▶ **Example 10.** Figure 3a shows a realizable history of the IO net of Figure 1a. It consists of six trajectories. The initial and final markings are $(5, 0, 1)$ and $(1, 0, 5)$. The history is realized by the firing sequence $t_3 t_1 t_1 t_3 t_2 t_4$.

Bunches and Pruning Lemma. A *bunch* is a multiset of trajectories with the same length and the same initial and final place. The Pruning Lemma states that every realizable history containing a bunch of trajectories from p to p' of size larger than the number of places n can be "pruned", meaning that the bunch can be replaced by a smaller one, also leading from p to p', while keeping the history realizable. (Notice, however, that the smaller bunch cannot always be chosen as a sub-multiset of the original one.)

▶ **Lemma 11** (Pruning Lemma). *Let N be an IO net with n places. Let H be a realizable history of N containing a bunch $B \subseteq H$ of size larger than n. There exists a bunch B' of size at most n with the same initial and final places as B, such that the history $H' \stackrel{def}{=} H - B + B'$ (where $+$ and $-$ denote multiset addition and subtraction) is also realizable in N.*

▶ **Example 12.** The realizable history H of Figure 3a, leading from $(5, 0, 1)$ to $(1, 0, 5)$, has a bunch B of size $4 \geq n$ from p_1 to p_3. Figure 3b shows a history H', leading from $(4, 0, 1)$ to $(1, 0, 4)$, resulting from the application of the Pruning Lemma to H and B. The new bunch B' from p_1 to p_3 given by the Pruning Lemma is drawn in dashed trajectories. Notice that the trajectory of B' that passes through p_2 does not appear in B. The firing sequence $t_3 t_1 t_3 t_4$ realizes H'.

Proof of the Shortening Theorem. We need a Boosting Lemma, which states that duplicating a trajectory of a history of an IO net preserves realizability. Intuitively, duplicating a trajectory corresponds to adding a "shadow" to a token, that follows the token wherever it goes. Since an enabled IO transition can move arbitrarily many tokens from its source place to its destination place, the shadow token can always follow the primary token. A formal proof of the lemma is given in the Appendix.

▶ **Lemma 13** (Boosting Lemma). *Let H be a realizable history of an IO net containing a trajectory τ. The history $H + \wr\tau\int$ is also realizable.*

▶ **Theorem 5** (IO Shortening). *Let N be an IO net with n places, and let M', M be two markings of N. If $M' \xrightarrow{*} M$, then $M' \xrightarrow{\sigma} M$ for some σ of accelerated length $|\sigma|_a \leq (n^3+1)^n$.*

Proof sketch. We explain our proof strategy for the IO Shortening Theorem. Given $M' \xrightarrow{*} M$, we take a history H such that $M' \xrightarrow{H} M$. Repeatedly applying the Pruning Lemma, we construct another realizable history $\widetilde{H}$ such that $\widetilde{T}_{p,q} = \min\{n, T_{p,q}\}$ for every two places p and q, where $T_{p,q}$ and $\widetilde{T}_{p,q}$ denote the number of trajectories of H and $\widetilde{H}$ leading from p to q. Using the fact that H has at most n^3 trajectories, we show that $\widetilde{H}$ can be chosen so that its length is bounded by $(n^3+1)^n$. We are not done yet, because in general $\widetilde{H}$ does not lead from M' to M, we only have $\widetilde{M'} \xrightarrow{\widetilde{H}} \widetilde{M}$ for markings $\widetilde{M'}, \widetilde{M}$ such that $\widetilde{M'} \leq M'$ and $\widetilde{M} \leq M$. In the last step we use the Boosting Lemma to add trajectories to $\widetilde{H}$ without increasing its length, yielding a realizable history $\overline{H}$ of the same length as $\widetilde{H}$, but satisfying $M' \xrightarrow{\overline{H}} M$. Finally, we extract from $\overline{H}$ a sequence $M' \xrightarrow{\sigma} M$ of accelerated length at most $(n^3+1)^n$. The full proof can be found in the Appendix. ◀

6 Shortening Theorem for BIO nets

The proof of the BIO Shortening Theorem (Theorem 7) is very involved. It follows the proof outline of Theorem 5: Given a firing sequence, consider a history H realized by it, construct an equivalent "small" history H', and extract from H' a sequence of short accelerated length. However, since BIO nets can create and destroy tokens, trajectories must be generalized to branching trajectories, which are trees of places; intuitively, the tree captures the cascade of tokens created by a token of the initial marking.

We fix a BIO net $N = (P, T, F)$ with n places, and let $m_d := \max_{t \in T} |t^\bullet - {}^\bullet t|$ denote the maximum number of tokens created by a transition.

Branching trajectories. A *branching trajectory* of N is a nonempty, directed tree β whose nodes are labeled with places of P. A node labeled by p is called a *p-node*. The i-th level of β, denoted by $\beta(i)$, is the (possibly empty) set of nodes of β at distance $(i-1)$ from the root. We let $M_\beta(i)$ denote the multiset of places labeling the nodes of $\beta(i)$. Observe that $M_\beta(i)$ is a marking. We say that β has *length* l if $\beta(l) \neq \emptyset$ and $\beta(l+1) = \emptyset$.

Histories and realizable histories. A *history* H of length l is a forest of branching trajectories of length at most l. We use histories to describe a behaviour from an initial marking; the history contains a branching trajectory for each token of the initial marking.

Given a history H of length h and an index $1 \leq i \leq h$, the i-th level of H is the set $H(i) = \bigcup_{\beta \in H} \beta(i)$, and the *the i-th marking of* H, denoted M_H^i, is the multiset $M_H^i = \sum_{\beta \in H} M_\beta(i)$. The markings M_H^1 and M_H^h are called the *initial* and *final* markings of H, and we write $M_H^1 \xrightarrow{H} M_H^h$. If the length of H is longer than the length of its branching trajectories, the final marking of H is the zero marking. Two histories are *equivalent* if they have the same initial and final markings.

A history H of length $h \geq 1$ is *realizable* if there exist transitions $t_1, \ldots, t_{h-1} \in T$ and numbers $k_1, \ldots, k_{h-1} \geq 0$ such that for every $1 \leq i \leq h-1$ the set $H(i)$ can be partitioned into two sets:

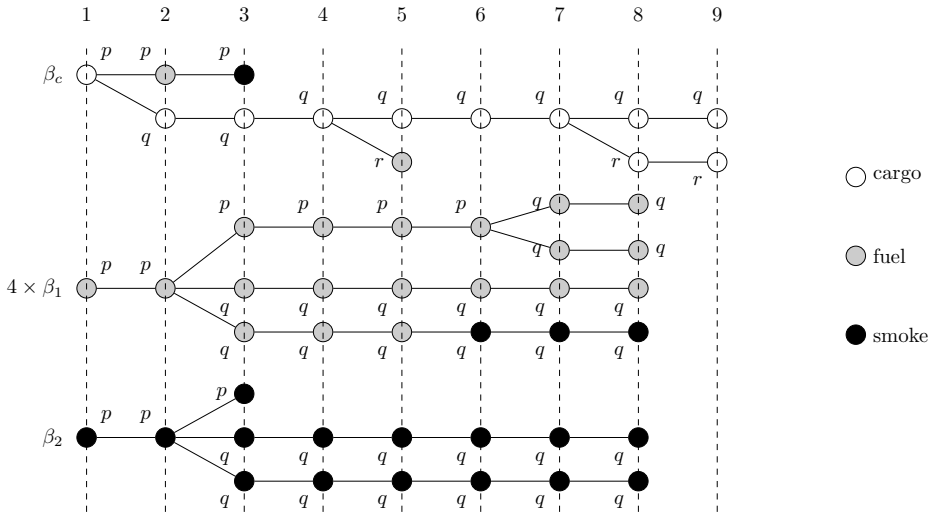


Figure 4 A decorated realizable history of a BIO net.

- A set $H_a(i)$ of exactly k_i nodes labeled by the source place of t_i. We call these nodes *active* nodes. Given a particular active node, say v, the multiset of labels of its children is the (possibly empty) multiset $\{\!|p_{d_1}, \ldots, p_{d_k}|\!\}$ of destination places of t_i.
- A set $H_p(i)$ of nodes, each of them with exactly one child, carrying the same label as their parents. We call these nodes *passive* nodes. This set must contain at least one node labeled by the place p_o observed by t_i.

We say that the sequence $t_1^{k_1} \cdots t_{h-1}^{k_{h-1}}$ *realizes* H. It follows easily from the definitions that $M_H^1 \xrightarrow{t_1^{k_1}} M_H^2 \cdots M_H^{h-1} \xrightarrow{t_{h-1}^{k_{h-1}}} M_H^h$ holds (where $M \xrightarrow{t^0} M'$ iff $M = M'$).

From this definition we easily obtain:

- $M \xrightarrow{*} M'$ iff there exists a realizable history with M and M' as initial and final markings.
- Every firing sequence that realizes a history of length h has accelerated length at most h.

▶ **Example 14.** - Figure 4 shows a realizable history H of a BIO net with places $\{p, q, r\}$. H consists of six branching trajectories: β_c, four copies of β_1, and β_2. The initial and final markings are $(6, 0, 0)$ and $(0, 1, 1)$. The transition t_i executed at step i is

$$t_1 = p \xrightarrow{p} \{\!|q, p|\!\} \quad t_2 = p \xrightarrow{p} \{\!|2q, p|\!\} \quad t_3 = p \xrightarrow{q} \emptyset \quad t_4 = q \xrightarrow{q} \{\!|r, q|\!\}$$
$$t_5 = r \xrightarrow{p} \emptyset \quad t_6 = p \xrightarrow{q} \{\!|2q|\!\} \quad t_7 = t_4 \quad t_8 = q \xrightarrow{r} \emptyset$$

where $t = x \xrightarrow{y} m$ denotes that x is the source place, y the observed place, and m the multiset of destination places of t. The firing sequence that realizes H is $t_1\, t_2^5\, t_3^2\, t_4\, t_5\, t_6^4\, t_7\, t_8^{18}$. While the final marking of H is produced by β_c only, β_c is not realizable on its own. For example, the r-node of β_c at level 5 is destroyed in the next step by the firing of t_5, but t_5 can only occur if there is at least one token in place p; this token is supplied by β_1 or β_2. We can think of β_1 and β_2 as branching trajectories that eventually become extinct, but before extinction provide tokens that need to be observable to fire some transitions.

Cargo, fuel, and smoke of a history. A *decoration* $\widehat{H}$ of a history H consists of the history H itself and a partition of the nodes of H into *cargo*, *fuel*, and *smoke* nodes. Figure 4 shows not only a history H but also a decoration $\widehat{H}$. Cargo nodes are white, grey nodes are fuel,

and black nodes are smoke. Before giving the formal definition of a decoration, let us provide some intuition. Think of the sequence of markings of a history as the sequence of states of a ship. All nodes of the final marking are cargo, they are what the ship "delivers" in the end. At any other marking, the cargo nodes are the "causal predecessors" of the final cargo nodes. Every decoration has the same cargo nodes, they only differ in the partition of the other nodes into fuel and smoke. Intuitively, a decoration reserves the right to use fuel nodes to fire transitions (a p-node can be "used" to fire a transition that observes p), and commits to never using a smoke node or its descendants. The most conservative decoration (which always exists) is the one that declares all non-cargo nodes as fuel. Our first goal will be to show that every history has an equivalent *fuel-efficient* history that delivers the same cargo but admits a low-fuel decoration.

Formally, a *decoration* of H is a partition of the nodes of H into *cargo*, *fuel*, and *smoke* nodes satisfying the following conditions:

- A node of H is a *cargo node* iff it has at least one descendant in $H(l)$.
- All descendants of smoke nodes are smoke nodes.
- For every place p and level i, if $H(i)$ contains smoke p-nodes, then it also contains fuel p-nodes. ("No smoke without fuel". Intuitively, the smoke p-nodes are not needed because the fuel p-nodes can be used instead.)

A *decorated history* is a pair consisting of H and a decoration of H. Observe that along all paths cargo comes before fuel, and fuel before smoke. Graphically, white nodes (if any) come before grey nodes (if any), and grey nodes before black nodes (if any).

Every history is equivalent to a fuel-efficient history. We prove that every realizable history has an equivalent realizable history with a *fuel-efficient* decoration, defined as follows:

▶ **Definition 15.** *Let $\widehat{H}$ be a decorated history. A place p is* wasteful at level i *if $\widehat{H}(i)$ contains more than n fuel p-nodes. A place p is* wasteful *in $\widehat{H}$ if it is wasteful at some level; otherwise p is* fuel-efficient *in $\widehat{H}$. Finally, $\widehat{H}$ is* fuel-efficient *if all places are fuel-efficient.*

▶ **Example 16.** Since $n = 3$, in the decorated history of Figure 4 place p is wasteful at levels 1 to 6, and q is wasteful at levels 3 to 8. The history is not fuel-efficient.

The proof is based on a Replacement Lemma, which plays the same role as the combination of the Pruning and Boosting Lemmas for IO nets. We start by introducing a definition.

▶ **Definition 17.** *The (p, i)-*bunch *of H, denoted $B_p(i)$, is the set of subtrees of H rooted at the p-nodes of $H(i)$.*

Loosely speaking, the Replacement Lemma shows that if i is the earliest level at which p is wasteful, then the bunch $B_p(i)$ of trajectories can be replaced so that the new history has a decoration where p is not wasteful anymore. The lemma shows how to do this while ensuring that the histories before and after the replacement are equivalent. Repeated applications of the Replacement Lemma yield a fuel-efficient history.

Formally, given a history B'_p with p-nodes as roots and with the same number of trees as $B_p(i)$, we let $H[B'_p/B_p(i)]$ denote the result of replacing each tree of $B_p(i)$ by a different tree of B'_p. For this we assume that $B_p(i)$ and B'_p have been enumerated in some way, and the j-th tree of $B_p(i)$ is replaced by the j-th tree of B'_p. We state the Replacement Lemma:

▶ **Lemma 18** (Replacement Lemma)**.** *Let $\widehat{H}$ be a decoration of a realizable history H such that p is wasteful, and i is the earliest level at which p is wasteful. There exists a history B'_p such that $H' \stackrel{def}{=} H[B'_p/B_p(i)]$ is realizable, equivalent to H, and has a decoration whose fuel-efficient places contain all fuel-efficient places of $\widehat{H}$ and p.*

Proof sketch. We describe the history B'_p, illustrating the construction on the decorated history of Figure 4. In this example p is already wasteful at level $i = 1$, and $B_p(1) = H$. So all of H is replaced by the bunch B'_p, shown in Figure 5.

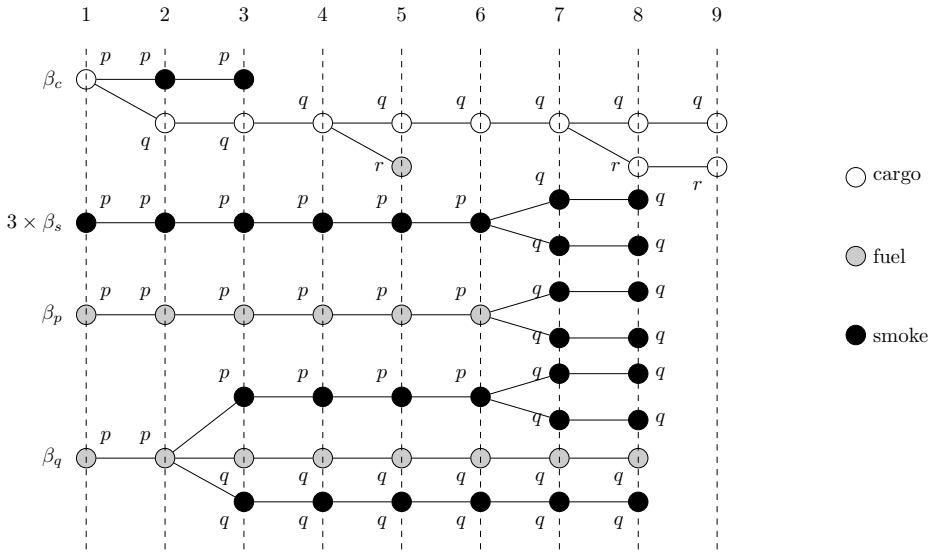


Figure 5 Result of replacing $B_p(1)$ in the history of Figure 4.

In order to describe B'_p we need some notions. We call smoke and fuel nodes *transportation* nodes. Given a decorated history $\widehat{H}$, let $last(p)$ denote the last level i such that $\widehat{H}(i)$ contains a transportation p-node. A *place-level* is a pair (q, j), where q is a place and j is a level of H. A *path* of place-levels is a concatenation of "steps" of two types: "doing nothing steps" from (r, l) to $(r, l+1)$ such that $l < last(r)$, and "transportation steps" from (r, l) to $(s, l+1)$ such that some transportation r-node of $\widehat{H}(l)$ has an s-child in $\widehat{H}(l+1)$. We say that (q, j) is *reachable* from (p, i) if there is a path from (p, i) to (q, j), and let $\mathcal{R}_{p,i}$ be the set of all place-levels (q, j) reachable from (p, i). In our example we have $\mathcal{R}_{p,1} = \{(p, 1), \ldots, (p, 6), (q, 3), \ldots, (q, 8)\}$. (Observe that $(r, 5)$ does not belong to $\mathcal{R}_{p,1}$, because its parent is a cargo node.)

B'_p is the union of three sets of branching trajectories, B_c, B_f, and B_s (where c, f, s stand for cargo, fuel, and smoke):

- B_c contains all branching trajectories of $B_p(i)$ rooted at a cargo node. (In Figure 5, B_c is the singleton set $\{\beta_c\}$.) The decoration of B_c is chosen so that it conserves the cargo nodes of $\widehat{H}$. Intuitively, B_c ensures that H' delivers the same cargo as H.
- B_f contains a branching trajectory β_q for every q such that $(q, j) \in \mathcal{R}_{p,i}$ for some j. (In Figure 5, B_f contains the two trees β_p and β_q.) Intuitively, these trajectories guarantee that the new set $\mathcal{R}_{p,i}$ of $\widehat{H'}$ is a superset of the old one, and so that any transition firing that relies on observing some place q at level j can still occur, because (q, j) is still reachable from (p, i).

  Let us now define β_q. (Figure 6 shows β_q for the history of Figure 5.) Let $first(q)$ be the smallest j such that $(q, j) \in \mathcal{R}_{p,i}$. There is a shortest path from (p, i) to $(q, first(q))$, and each step of the path corresponds to doing nothing or to executing a transition once. (In Figure 6 we have $(p, i) = (p, 1)$, $(q, first(q)) = (q, 3)$, and the path corresponds to doing nothing in the first step, and then firing t_2.) Let δ_q be the corresponding branching trajectory. (In Figure 6, δ_q is the tree contained in the blue area.) First we append a path to each leaf of δ_q: If the leaf is, say, an r-node at level j, then we append to it a

path of r-nodes from level j to level $last(r)$. (Red area of Figure 6.) Then, we append to the end of each path a *destroyer*, i.e., a tree that makes the token disappear. We choose for this any subtree of $\widehat{H}$ rooted in a transportation node of $(r, last(r))$. (Green area of Figure 6; in order to destroy a p-node we first transform it into two q-nodes by firing t_6, wait while t_7 is fired in another part of the history, and then destroy the q-nodes by firing t_8 twice. The two q-nodes are destroyed by firing t_8 twice.) The decoration of β_q is chosen so that there is a fuel path rooted in (p, i) containing q-nodes from levels $first(q)$ to $last(q)$, and the rest is smoke.

- B_s contains $|B_p(i)| - |B_c| - |B_f|$ copies of a tree of smoke nodes β_s, consisting of a path of p-nodes, leading from level i to level $last(p)$, appended with a destroyer. Intuitively, this is smoke added to ensure that $H(i) = H'(i)$.

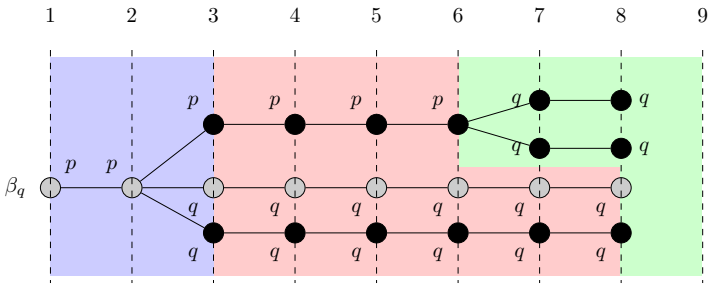


Figure 6 Illustration of the construction of the set B_f of trees.

This concludes the description of B'_p. There are at most $|B_f| \leq n$ fuel nodes per level in B'_p, so p is fuel-efficient. The proof that H' is realizable, equivalent to H, and has a decoration in which there are no new wasteful places can be found in the appendix. ◀

Repeated applications of the Replacement Lemma yield the existence of a fuel-efficient decoration $\widehat{H'}$ of a history H' equivalent to H.

▶ **Example 19.** Applying the Replacement Lemma to p and $i := 1$ and the decorated history $\widehat{H}$ of Figure 4 yields the decorated history $\widehat{H'}$ of Figure 5. Like H, it leads from $(6, 0, 0)$ to $(0, 1, 1)$. It is realized by $t_1\, t_2\, t_3\, t_4\, t_5\, t_6^5\, t_7\, t_8^{12}$. Place p is no longer wasteful in $\widehat{H'}$, and in fact all places are fuel-efficient.

The next step of the proof is the Unique Footprint Lemma. Loosely speaking, it shows that for every history there exists an equivalent history in which any two levels differ in the cargo, the fuel, or the *support* of the smoke. This allows us to bound the length of the history. We need a preliminary lemma. Let $\widehat{H}_c(i)$, $\widehat{H}_f(i)$, $\widehat{H}_s(i)$ denote the multisets of cargo, fuel, and smoke nodes of $\widehat{H}(i)$. Intuitively, the Smoke Irrelevance lemma shows that we can always deliver the same cargo using the same fuel *independently* of the initial amount of smoke.

▶ **Lemma 20** (Smoke Irrelevance Lemma)**.** *Let $\widehat{H}$ be a realizable decorated history of length h, and let μ be any multiset of places such that $\|\mu\| \subseteq \|\widehat{H}_s(1)\|$. There exists a realizable decorated history $\widehat{H'}$ of length h such that $\widehat{H'}_s(1) = \mu$, and $\widehat{H'}_c(i) = \widehat{H}_c(i)$ and $\widehat{H'}_f(i) = \widehat{H}_f(i)$ for every level $1 \leq i \leq h$.*

Proof sketch. Rename $\nu \stackrel{\text{def}}{=} \widehat{H}_s(1)$ for clarity. To construct $\widehat{H'}$, start with $\widehat{H}$, and do the following for every place $p \in P$. If $\mu(p) \leq \nu(p)$, then delete $\nu(p) - \mu(p)$ smoke p-nodes from $\widehat{H}(1)$ as well as all their descendants (which are all smoke nodes by definition). If $\mu(p) > \nu(p)$,

then add to $\widehat{H}$ $(\mu(p) - \nu(p))$ copies of an arbitrary tree β of smoke nodes of $\widehat{H}$ rooted in $(p, 1)$. This tree exists because $p \in \|\mu\|$, and so $p \in \|\nu\|$. The addition of the copies of β maintains the "no smoke without fuel" property, because it was already fulfilled in $\widehat{H}$ by the nodes of β. The smoke nodes of $\widehat{H}'(1)$ thus constructed are labelled by μ, and fuel and cargo nodes are neither added nor removed. The proof that $\widehat{H}'$ is realizable can be found in the Appendix. ◀

▶ **Definition 21.** *Given a level $\widehat{H}(i)$ of a decorated history, define its* footprint *as the triple $(\widehat{H}_c(i), \widehat{H}_f(i), \|\widehat{H}_s(i)\|)$ (that is, we only take the support of $\widehat{H}_s(i)$, not $\widehat{H}_s(i)$ itself).*

▶ **Lemma 22** (Unique Footprint Lemma)**.** *Every realizable history has an equivalent fuel-efficient decorated history in which every level has a different fooprint.*

Proof. Let $\widehat{H}$ be a realizable decorated history. By the Replacement Lemma, we can assume w.l.o.g. that $\widehat{H}$ is fuel-efficient. Assume further that $\widehat{H}$ has minimal length h, i.e., every equivalent decorated history that is also fuel-efficient has length at least h. We claim that every level of $\widehat{H}$ has a different footprint. Assume this is not the case. Then there exist two indices $1 \leq i < j \leq h$ such that $(\widehat{H}_c(i), \widehat{H}_f(i), \|\widehat{H}_s(i)\|) = (\widehat{H}_c(j), \widehat{H}_f(j), \|\widehat{H}_s(j)\|)$. The truncated history $\widehat{H}(j)\widehat{H}(j+1)\dots\widehat{H}(h)$ is clearly realizable. Since $\|\widehat{H}_s(i)\| = \|\widehat{H}_s(j)\|$, we can apply the Smoke Irrelevance Lemma with $\mu := \widehat{H}_s(i)$ and obtain a decorated history $\widehat{H'}$ of length $h - j + 1$ such that $(\widehat{H}_c(i), \widehat{H}_f(i), \widehat{H}_s(i)) = (\widehat{H'}_c(1), \widehat{H'}_f(1), \widehat{H'}_s(1))$ (notice: now $\widehat{H}_s(i) = \widehat{H'}_s(1)$, instead of only $\|\widehat{H}_s(i)\| = \|\widehat{H'}_s(1)\|$). But this implies $\widehat{H}(i) = \widehat{H'}(1)$, and so the concatenation $H(1)\cdots H(i-1)H'(1)\cdots H'(h-j+1)$ is also a realizable history. By the Smoke Irrelevance Lemma we have $\widehat{H'}_c(h-j+1) = \widehat{H}_c(h)$. Since the last levels of a decorated history only contain cargo nodes, this implies $\widehat{H'}(h-j+1) = \widehat{H}(h)$, and so the concatenation is equivalent to H. Further, since $\widehat{H'}$ has the same cargo and fuel nodes as $\widehat{H}(j)\widehat{H}(j+1)\dots\widehat{H}(h)$, the concatenation is also fuel-efficient, contradicting that $\widehat{H}$ has minimal length. ◀

We are equipped to prove the Shortening Theorem.

▶ **Theorem 7** (BIO Shortening)**.** *Let N be a BIO net with n places, let M', M be two markings of N, and let $|M'| = m'$, $|M| = m$. Let $m_d := \max_{t \in T} |t^\bullet - {}^\bullet t|$ denote the maximum number of tokens created by a transition of N. If $M' \xrightarrow{*} M$, then $M' \xrightarrow{\sigma} M$ for some σ of accelerated length $|\sigma|_a \leq 2^n(m+1)^n(n+1)^n$. Further, the intermediate markings along σ have size at most $(m' + 2^n(m+1)^n(n+1)^n(m+n)m_d)m_d^n$.*

Proof. We first prove the bound on the accelerated length. By the Unique Footprint Lemma, there is a history H such that $M' \xrightarrow{H} M$ and H has a decoration $\widehat{H}$ where every level has a different footprint. So the length of $\widehat{H}$ is bounded by the number of possible footprints of the histories leading from M' to M. Since, by definition, the number of cargo nodes cannot decrease from a level to the next, and the last level consists of only cargo, every level has between 0 and m cargo nodes per place. Since $\widehat{H}$ is fuel-efficient, every level has between 0 and n fuel nodes per place. Finally, there are at most 2^n possible supports in a net with n places. So the number of footprints, and so the length of $\widehat{H}$, and the accelerated length of any firing sequence realizing $\widehat{H}$, is at most $2^n(m+1)^n(n+1)^n$.

Let us now prove the token bound. To bound the number of smoke nodes in each level, we apply the following operation. Replace every largest tree of smoke nodes (since the children of smoke nodes are smoke, this means trees rooted at smoke nodes whose parents are cargo or fuel) by the tree β_s defined as in the Replacement Lemma: β_s is a path of smoke p-nodes ending at level $last(p)$, appended by a p-destroyer tree. This maintains realizability, because

(by the "no smoke without fuel" property in $\widehat{H}$), it does not decrease the support of the multiset of places of any level. We call $\widehat{H'}$ the resulting realizable history with decorated nodes. Note that the "no smoke without fuel" property may not hold in $\widehat{H'}$, so it is not formally a decorated history, but it is sufficient to conclude the proof. $\widehat{H'}$ has the following property: smoke p-nodes can only create other nodes (which, by definition, are also smoke) at the level $last(p)$, and it can create at most m_d of them.

At all other levels j of $\widehat{H'}$, only cargo and fuel nodes can create nodes. There are at most $h' \leq 2^n(m+1)^n(n+1)^n$ levels, and at most $(m+n)$ cargo and fuel nodes per place. Each transition has a unique source place, and all the nodes are added to the initial m' nodes corresponding to the tokens of M'. Thus there are at most $m' + h'(m+n)m_d$ nodes at the first level $last(p)$ in which a smoke node creates nodes. At most all of the nodes are smoke, so at most $(m' + h'(m+n)m_d)m_d$ nodes are created. There are at most n levels $last(p)$, which each create at most the total amount of nodes times m_d nodes. Thus at every level of the history there are at most $(m' + h'(m+n)m_d)m_d^n$ nodes, concluding the proof. ◀

7 Many-to-many reachability and coverability

In [10] we prove that many-to-many versions of the reachability and coverability problems for IO nets are PSPACE-complete. We extend this result to BIO nets, which requires to use not only the Shortening Theorem itself, but also the lemmas conducting to its proof.

We recall some definitions of [10]. A set $\mathcal{C}$ of markings of a net $N = (P, T, F)$ is a *cube* if there exist mappings $L\colon P \to \mathbb{N}$ and $U\colon P \to \mathbb{N} \cup \infty$ such that $M \in \mathcal{C}$ if and only if $L \leq M \leq U$. Abusing language, we identify $\mathcal{C}$ with the pair (L, U). Observe that cubes can be infinite sets of markings. The *cube-reachability (coverability)* consists of deciding, given a net N and cubes $\mathcal{C}, \mathcal{C}'$ of $\mathbb{N}$, whether there exist markings $M \in \mathcal{C}$ and $M' \in \mathcal{C}'$ such that M is reachable (coverable) from M'.

▶ **Theorem 23.** *The cube-reachability and cube-coverability problems for BIO nets are* PSPACE*-complete.*

Proof. PSPACE-hardness follows from PSPACE-hardness for IO nets. We show that the problems are in NPSPACE and apply Savitch's theorem. Cube-coverability from $\mathcal{C}'$ to $\mathcal{C} = (L, U)$ reduces to cube-reachability from $\mathcal{C}'$ to the cube (L, U'') such that $U''(p) = \infty$ for all p, so it suffices to consider cube-reachability from $\mathcal{C}' = (L', U')$ to $\mathcal{C} = (L, U)$. For each place p with upper bound $U(p) = \infty$ in $\mathcal{C}$, add a "destroying transition" τ_p to N with preset ${}^\bullet\tau_p = \{p\}$ and postset $\tau_p{}^\bullet = \emptyset$. We guess a marking M of size m satisfying $M(p) = L(p)$ if $U(p) = \infty$, and $L(p) \leq M(p) \leq U(p)$ if $U(p) < \infty$. This reduces the problem to checking if M is reachable in the modified net from some marking of $\mathcal{C}'$. By Lemma 20, only the footprint of a marking matters for knowing whether it can reach marking M. We pick M' in $\mathcal{C}'$ of size $m' \leq m + n^2 + \max(|L'|, n)$. The summands correspond to the cargo, fuel and smoke nodes of the initial marking of a fuel-efficient decorated history given by the Replacement Lemma if $M' \xrightarrow{*} M$ holds, where $\max(|L'|, n)$ is enough smoke nodes so that $M' \in \mathcal{C}'$ and any set of places is covered. By Theorem 9, $M' \xrightarrow{*} M$ can be checked in PSPACE. ◀

8 Conclusion

We have shown that immediate observation Petri nets are globally flat, allowing the use of existing efficient verification tools. We have also studied branching immediate observation nets, which are simultaneously a generalisation of IO nets, and of the Basic Parallel Processes model. The class of BIO nets significantly extends the expressive power of both IO nets and BPP nets, bringing together process creation and (restricted) cross-process interaction via

a simple and natural definition. While such an extension does not preserve global flatness, we have proven that local flatness is still preserved, and many-to-many reachability and coverability problems are still in PSPACE.

As BIO nets combine PSPACE-verifiable reachability and non-semilinear reachability relation, the further study of the structure of this reachability relation seems of interest. For instance, we plan to obtain the bounds on the size of the pre- and post- image of a marking, provided that these images are finite. It is also worth noting that the results of this paper still hold (up to a slight alteration of the Shortening Theorem bounds) if we define BIO transitions via the constraint $|{}^\bullet t - t^\bullet| \leq 1$. This is equivalent to extending BIO transitions with the possibility of multiple observations and the absence of a source place.

References

1 David Angeli, Patrick De Leenheer, and Eduardo D Sontag. A petri net approach to the study of persistence in chemical reaction networks. *Mathematical biosciences*, 210(2):598–618, 2007.

2 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

3 Aurore Annichini, Ahmed Bouajjani, and Mihaela Sighireanu. TReX: A tool for reachability analysis of complex systems. In *Lecture Notes in Computer Science*, volume 2102, pages 368–372, 2001.

4 Paolo Baldan, Nicoletta Cocco, Andrea Marin, and Marta Simeoni. Petri nets for modelling metabolic pathways: a survey. *Natural Computing*, 9(4):955–989, 2010.

5 Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Laure Petrucci. FAST: Fast acceleration of symbolic transition systems. In *Lecture Notes in Computer Science*, volume 2725, pages 118–121, 2003.

6 Bernard Boigelot. The LASH toolset homepage, 2014. URL: `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/index.html`.

7 P. Chelikani, I. Fita, and P.C. Loewen. Diversity of structures and properties among catalases. *Cell. Mol. Life Sci.*, 61, 2004.

8 Søren Christensen, Yoram Hirshfeld, and Faron Moller. Decomposability, decidability and axiomatisability for bisimulation equivalence on basic parallel processes. In *LICS*, pages 386–396, 1993. URL: `https://www.wikidata.org/entity/Q59557027`.

9 Javier Esparza. Petri nets, commutative context-free grammars, and basic parallel processes. *Fundam. Inform.*, 31(1):13–25, 1997.

10 Javier Esparza, Mikhail Raskin, and Chana Weil-Kennedy. Parameterized analysis of immediate observation petri nets. In *Lecture Notes in Computer Science*, volume 11522, pages 365–385, 2019.

11 Laurent Fribourg. Petri nets, flat languages and linear arithmetic. In *WFLP*, pages 344–365, 2000.

12 John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8:135–159, 1979.

13 Slawomir Lasota. EXPSPACE lower bounds for the simulation preorder between a communication-free petri net and a finite-state system. *Inf. Process. Lett.*, 109(15):850–855, 2009.

14 Jérôme Leroux and Grégoire Sutre. Flat counter automata almost everywhere! In *ATVA*, volume 3707 of *Lecture Notes in Computer Science*, pages 489–503. Springer, 2005.

15 Wolfgang Marwan, Annegret Wagler, and Robert Weismantel. Petri nets as a framework for the reconstruction and analysis of signal transduction pathways and regulatory networks. *Natural Computing*, 10(2):639–654, 2011.

16 Ernst W. Mayr and Jeremias Weihmann. Complexity results for problems of communication-free petri nets and related formalisms. *Fundam. Inform.*, 137(1):61–86, 2015.

17 Hsu-Chun Yen. On reachability equivalence for BPP-nets. *Theor. Comput. Sci.*, 179(1-2):301–317, 1997.

A Shortening Theorem for IO nets

▶ **Lemma 13** (Boosting Lemma). *Let H be a realizable history of an IO net containing a trajectory τ. The history $H + \wr\tau\wr$ is also realizable.*

Proof sketch. Let h be the length of H, and let $t_1^{k_1} \cdots t_{h-1}^{k_{h-1}}$ be a realization of H. For every $1 \leq i \leq h-1$ define k'_i as follows: if $\tau(i) = \tau(i+1)$, then $k'_i \stackrel{\text{def}}{=} k_i$; if $\tau(i) \neq \tau(i+1)$, then $k'_i \stackrel{\text{def}}{=} k_i + 1$. We claim that $t_1^{k'_1} \cdots t_{h-1}^{k'_{h-1}}$ is a realization of $H + \tau$. The proof is by induction on h.

Assume $h = 1$. Then H is realizable by t^0 for any transition t, and so is $H + \tau$.

Assume that the induction property holds for some $h \geq 1$, and let H be of length $h+1$, realizable by $t_1^{k_1} \cdots t_h^{k_h}$. By induction, the history $H + \tau$ truncated of its last step is realizable by $t_1^{k'_1} \cdots t_{h-1}^{k'_{h-1}}$. If $\tau(h) \neq \tau(h+1)$ in H, then since $\tau \in H$ and H is realizable, $\tau(h)\tau(h+1) = p_s p_d$ for p_s and p_d the source and destination places of t_h. Additionally, there are $k_h - 1$ other trajectories τ' such that $\tau'(h)\tau'(h+1) = p_s p_d$, and there is at least one trajectory τ' such that $\tau'(h)\tau'(h+1) = p_o p_o$. Thus $t_1^{k'_1} \cdots t_{h-1}^{k'_{h-1}} t_h^{k_h+1}$ realizes $H + \tau$. If $\tau(h) = \tau(h+1)$ in H, then $H + \tau$ is realized by $t_1^{k'_1} \cdots t_{h-1}^{k'_{h-1}} t_h^{k_h}$. ◀

▶ **Theorem 5** (IO Shortening). *Let N be an IO net with n places, and let M', M be two markings of N. If $M' \xrightarrow{*} M$, then $M' \xrightarrow{\sigma} M$ for some σ of accelerated length $|\sigma|_a \leq (n^3+1)^n$.*

Proof. Let H be a realizable history such that $M' \xrightarrow{H} M$, and let h be the length of H. For every two places p, q, let $B_{p,q}$ denote the bunch of all trajectories of H leading from p to q, and let $T_{p,q} = \text{size}(B_{p,q})$. Applying the Pruning Lemma to all bunches $B_{p,q}$ such that $T_{p,q} \geq n$, we obtain a new realizable history $\widetilde{H}$ satisfying

$$\widetilde{T}_{p,q} = \min\{n, T_{p,q}\} \quad \text{for every } p, q \in P. \tag{1}$$

So $\widetilde{H}$ has $\sum_{p,q \in P} \widetilde{T}_{p,q} \leq n^3$ trajectories. Let $M^1_{\widetilde{H}} \xrightarrow{t_1^{k_1}} M^2_{\widetilde{H}} \cdots M^{h-1}_{\widetilde{H}} \xrightarrow{t_{h-1}^{k_{h-1}}} M^h_{\widetilde{H}}$ be a realization of $\widetilde{H}$. Since $\widetilde{H}$ hast at most n^3 trajectories, we have $M^i_{\widetilde{H}}(p) \leq n^3$ for every $p \in P$ and $1 \leq i \leq n$. If $h \geq (n^3+1)^n$, then there are $1 \leq i \neq j \leq h$ such that $M^i_{\widetilde{H}} = M^j_{\widetilde{H}}$, and the history $\widetilde{H'}$ obtained by "cutting out" the fragment of $\widetilde{H}$ between $M^i_{\widetilde{H}}$ and $M^j_{\widetilde{H}}$ is also realizable. (Formally, $\widetilde{H'}$ is the result of replacing every trajectory $\tau \in \widetilde{H}$ by $\tau(1) \cdots \tau(i)\tau(j+1) \cdots \tau(h)$.) So w.l.o.g. we can assume $\widetilde{h} < (n^3+1)^n$.

Since $\widetilde{H}$ is realizable, we have $\widetilde{M'} \xrightarrow{\widetilde{H}} \widetilde{M}$ for some markings $\widetilde{M'}, \widetilde{M}$. We examine the relation between M' and $\widetilde{M'}$, and between M and $\widetilde{M}$. For every place p, the initial (final) number of tokens of p in H is equal to the number of trajectories of H of starting in p (ending in p), and similarly for $\widetilde{H}$. So we have

$$\begin{array}{llll} M'(p) = \sum_{q \in P} T_{p,q} & \text{and} & M(p) = \sum_{q \in P} T_{q,p} \\ \widetilde{M'}(p) = \sum_{q \in P} \widetilde{T}_{p,q} & \text{and} & \widetilde{M}(p) = \sum_{q \in P} \widetilde{T}_{q,p} \ . \end{array}$$

Further, for every place $p \in P$:

(a) $\widetilde{M'}(p) \leq M'(p)$, and $\widetilde{M}(p) \leq M(p)$.
Follows immediately from $\widetilde{T}_{p,q} \leq T_{p,q}$ for every $q \in P$ (Equation 1).

(b) If $\widetilde{M'}(p) = 0$ then $M'(p) = 0$, and if $\widetilde{M}(p) = 0$ then $M(p) = 0$.
If $\widetilde{M'}(p) = 0$ then $\widetilde{T}_{p,q} = 0$ for every $q \in P$. So, by Equation 1, $\widetilde{T}_{p,q} = T_{p,q}$ for every $q \in P$, and so $M'(p) = \sum_{q \in P} T_{p,q} = \sum_{q \in P} \widetilde{T}_{p,q} = \widetilde{M'}(p) = 0$. The proof for the target markings is analogous.

Let $\overline{H}$ be the history obtained from $\widetilde{H}$ as follows: For every $p, q \in P$, if $\widetilde{T}_{p,q} > 0$ then pick a trajectory $\tau \in B_{p,q}$, and set $\overline{B}_{p,q} = \widetilde{B}_{p,q} + (\widetilde{T}_{p,q} - T_{p,q} - 1) \cdot \tau$ By the Boosting Lemma, $\overline{H}$ is realizable, and so there are markings $\overline{M'}, \overline{M}$ such that $\overline{M'} \xrightarrow{\overline{H}} \overline{M}$. Further, by (a) and (b) above we have $\overline{T}_{p,q} = T_{p,q}$ for every $p, q \in P$, and so for every $p \in P$:

$$\overline{M'}(p) = \sum_{q \in P} \overline{T}_{p,q} = \sum_{q \in P} T_{p,q} = M'(p)$$

So we get $M' \xrightarrow{\overline{H}} M$. Since $\widetilde{H}$ and $\overline{H}$ have the same length, we get $\overline{h} < (n^3 + 1)^n$. So every firing sequence realizing $\overline{H}$ has accelerated length at most $(n^3 + 1)^n$, and we are done. ◀

B Shortening Theorem for BIO nets

We give ourselves a few more definitions to help in the proofs. We call smoke and fuel nodes *transportation* nodes. Given a decorated history $\widehat{H}$, let $last(p)$ denote the last level i such that $\widehat{H}(i)$ contains a transportation p-node. A *place-level* is a pair (q, j), where q is a place and j is a level of H. A *path* of place-levels is a concatenation of "steps" of two types: "doing nothing steps" from (r, l) to $(r, l+1)$, and "transportation steps" from (r, l) to $(s, l+1)$ such that some transportation r-node of $\widehat{H}(l)$ that has an s-child in $\widehat{H}(l+1)$. We say that (q, j) is *reachable* from (p, i) if there is a path from (p, i) to (q, j), and let $\mathcal{R}_{p,i}$ be the set of all place-levels (q, j) reachable from (p, i).

▶ **Lemma 18** (Replacement Lemma). *Let $\widehat{H}$ be a decoration of a realizable history H such that p is wasteful, and i is the earliest level at which p is wasteful. There exists a history B'_p such that $H' \stackrel{def}{=} H[B'_p/B_p(i)]$ is realizable, equivalent to H, and has a decoration whose fuel-efficient places contain all fuel-efficient places of $\widehat{H}$ and p.*

Proof. We first construct $H' \stackrel{\text{def}}{=} H[B'_p/B_p(i)]$ and show that it is realizable and equivalent to H. Then, we define a decoration $\widehat{H'}$ of H', and show that it realizes the condition of the lemma.

Construction of H'. We define B'_p as the union of three sets of branching trajectories, B_c, B_f, and B_s (where c, f, s stand for cargo, fuel, and smoke):

- B_c contains all branching trajectories of $B_p(i)$ rooted at a cargo node.
- B_f contains a branching trajectory β_q for every $q \in \mathcal{R}_{p,i}$.
  We define β_q. Let $first(q)$ be the smallest j such that $(q, j) \in \mathcal{R}_{p,i}$. Notice that $first(q) \leq last(q)$ for all $q \in P$, since by definition of reachability there exists a transportation q-node in level $first(q)$. There is a shortest path from (p, i) to $(q, first(q))$, and each step of the path corresponds to doing nothing or to executing a transition once. Let δ_q be the corresponding branching trajectory. First we append a path to each leaf of δ_q: If the leaf is, say, an r-node at level j, then we append to it a path of r-nodes from level j to level $last(r)$. Then, we append to the end of each path a *destroyer*, i.e., a tree that makes the token disappear. We choose for this any subtree γ_r of $\widehat{H}$ rooted in a transportation node of $(r, last(r))$.
- B_s contains $|B_p(i)| - |B_c| - |B_f|$ copies of a tree β_s, consisting of a path of p-nodes, leading from level i to level $last(p)$, appended with a destroyer γ_p.

We define the replacement $H' = H[B'_p/B_p(i)]$: we replace the trees of $B_p(i)$ with a cargo root in $\widehat{H}$ by the same tree in B_c, we replace some trees of $B_p(i)$ with a fuel root in $\widehat{H}$ by the trees of B_f (in any order), and the rest of the trees of $B_p(i)$ by the trees of B_s. This is

well-defined because the the trees of B'_p all have p-nodes as root, there are no more than n trees in B_f and more than n trees with fuel roots in $B_p(i)$ since p is wasteful at i, and there are as many trees overall in B'_p as in $B_p(i)$.

History H' is realizable and equivalent. History H' is equivalent to history H: the trees added in $B'_p \setminus B_c$ all end in destroyers, and the other trees of H' were already in H, so H' has the same final marking. In case $i = 1$, the number of p-nodes in $H(i)$ and $H'(i)$ is the same so H' has the same initial marking.

History H is realizable, and we note $t_1^{k_1} \cdots t_{h-1}^{k_{h-1}}$ a sequence that realizes it, for some transitions $t_1, \ldots, t_{h-1} \in T$ and numbers $k_1, \ldots, k_{h-1} \geq 0$. We show that H' is realizable using the same transitions but different numbers $l_1, \ldots, l_{h-1} \geq 0$. Let $1 \leq j \leq h-1$. Let $H'_p(i)$ be the set of nodes of $H'(j)$ which have exactly one child with the same label, and let $H'_a(j)$ be the rest. We claim that for every node v' in $H'_a(j)$ with label r and multiset of children labels c, there exists a node v in $H_a(j)$ with label r and multiset of children labels c. By realizability of H this entails that v' is labeled with the source place p_s of t_j, and the multiset of labels of its children is the multiset $\wr p_{d_1}, \ldots, p_{d_k} \int$ of destinations of t_j.

Now to show our claim. Let v' a node of $H'_a(j)$. If v' is not a node of the subtree B'_p, or if v' is a node of B_c, then we are done. Let us assume this is not the case, i.e. $v' \in B_f \cup B_s$.

- If v' is in a tree β_s, then it is in a a destroyer (since v' is not in $H'_p(j)$) and so it is in a copy of a subtree of $\widehat{H}$.
- Assume v' is in a tree β_q for some $q \in \mathcal{R}_{p,i}$. If v' is in a destroyer then it is in a copy of a subtree of $\widehat{H}$, we are done. Otherwise, v' is in the tree δ_q induced by the shortest path ρ_q from (p,i) to $(q, \mathit{first}(q))$ in H. Since v' is not passive, i.e. $v' \notin H'_p(j)$, and by definition of how a path induces a tree, there is an r-node v of $H(j)$ with the same children as v'.

We now show that the set $H'_p(j)$ contains a node labeled by the place p_o observed by t_j. If there is a node labeled p_o in $H_p(j)$ that is not in $B_p(i)$, then it is also in $H'_p(j)$ and we are done. Let us assume that the only nodes of $H_p(j)$ labeled p_o are in $B_p(i)$. If there is a cargo node labeled p_o in $\widehat{H}_p(j)$ then it is also in $H'_p(j)$ so we are done. Otherwise there exists a transportation node v labeled p_o in $\widehat{H}_p(j)$, and $j \leq \mathit{last}(p_o)$ by definition. Since v is in $B_p(i)$, either v is in a tree with a cargo root, or place-level (p_o, j) is reachable from (p,i). If v is in a tree of $B_p(i)$ with a cargo root, it is also in $B_c \subseteq B'_p$. Otherwise $(p_o, j) \in \mathcal{R}_{p,i}$, and therefore by construction there is a node in B'_p labeled p_o at every level between $\mathit{first}(p_o)$ and $\mathit{last}(p_o)$, in particular at j.

Decoration of H'. Let $\widehat{H'}$ be the following decoration of H'.

We start with the nodes of B_f and B_s. In each tree β_q in B_f, constructed around the tree induced by a shortest path ρ_q from (p,i) to $(q, \mathit{first}(q))$, we let the nodes along the path ρ_q be fuel nodes, along with the nodes along one branch from $(q, \mathit{first}(q))$ to $(q, \mathit{last}(q))$. All the other nodes of β_q are defined as smoke nodes. We let all the nodes of the trees β_s be smoke nodes.

The rest of the nodes of H' are decorated in two steps. First, we set $\widehat{H'}$ to be equal to $\widehat{H}$ on the nodes of $H' \setminus (B_f \cup B_s)$, which is possible because $H' \setminus B'_p = H \setminus B_p(i)$ and the trajectories of B_c are trajectories of $B_p(i)$. Then, we do the following "re-decoration". Let (q,j) be a place level reachable from (p,i) in H'. If there are any fuel nodes labeled q in $(H' \setminus B_f)(j)$, redecorate them and all their descendants as smoke nodes in $\widehat{H'}$. Do this for every (q,j) reachable from (p,i).

The order of "cargo then fuel then smoke" is respected along the branching trajectories of $\widehat{H'}$ because they are respected in B'_p, and there are no more than n trees with a fuel root in B'_p while there are more than n in $B_p(i)$. The cargo nodes in $\widehat{H'}$ are well defined, as the cargo nodes of $\widehat{H'}$ are the cargo nodes of $\widehat{H}$.

The smoke/fuel partition of $\widehat{H'}$ is well defined: First, remark that the last level index $last(p)$ at which there is a transportation p-node in $\widehat{H'}$ is equal to $last(p)$ in $\widehat{H}$, for any place p by construction. Let v be a smoke q-node at level $\widehat{H'}(j)$, for some q and j. We check that the "no smoke without fuel" condition is fulfilled. If (q, j) is reachable from (p, i) in H then there exists a fuel q-node in $\widehat{H'}(j)$ provided by β_q, since $j \leq last(q)$ by virtue of v being smoke. If (q, j) is not reachable from (p, i) in H, then there is no subtree of $B_p(i)$ rooted in a transportation p-node with a descendant labeled q. Therefore in $\widehat{H'}$, node v is not in B_f. Since it is also not in B_s, whose trees are only p nodes until $last(p)$, v is in either a tree of B_c or in no tree of B'_p, and therefore v exists also in $\widehat{H}$ as a smoke node. Since the smoke/fuel partition of $\widehat{H}$ is well defined, there exists a fuel q-node v' in $\widehat{H}(j)$. Since (q, j) is not reachable from (p, i) in H, v' is either in B_c or not part of $B_p(i)$ and so v' is also in $\widehat{H'}(j)$.

For every place-level (q, j) in H' reachable from (p, i), there are at most n fuel q-nodes in $\widehat{H'}(j)$. Indeed, by definition, the only fuel nodes labeled q in $\widehat{H'}(j)$ are in $B_f(j)$. By definition of $B_f(j)$, the only fuel nodes labeled q in $B_f(j)$ are in the trees β_r for some $r \in \mathcal{R}_{p,i}$. There are a most n such trees, and in each tree there is at most one fuel node per level.

Therefore there are no wasteful places q at some level j such that (q, j) is reachable from (p, i) in H'. In particular, p is fuel-efficient since i is the earliest level at which p is wasteful in H. If there is a wasteful place-level in $\widehat{H'}$, then it is unreachable from (p, i) in H'. By definition of H', this means that it is also a wasteful place-level in $H \setminus B'_p$ and thus in H. Thus the fuel-efficient places of $\widehat{H'}$ contain all the fuel-efficient places of $\widehat{H}$, as well as the place p. ◀

We remind the reader that $\widehat{H}_c(i)$, $\widehat{H}_f(i)$, $\widehat{H}_s(i)$ denote the multiset of cargo, fuel, and smoke nodes of $\widehat{H}(i)$.

▶ **Lemma 20** (Smoke Irrelevance Lemma). *Let $\widehat{H}$ be a realizable decorated history of length h, and let μ be any multiset of places such that $\|\mu\| \subseteq \|\widehat{H}_s(1)\|$. There exists a realizable decorated history $\widehat{H'}$ of length h such that $\widehat{H'}_s(1) = \mu$, and $\widehat{H'}_c(i) = \widehat{H}_c(i)$ and $\widehat{H'}_f(i) = \widehat{H}_f(i)$ for every level $1 \leq i \leq h$.*

Proof. Rename $\nu := \widehat{H}_s(1)$ for clarity. To construct $\widehat{H'}$, start with $\widehat{H}$, and do the following for every place $p \in P$. If $\mu(p) \leq \nu(p)$, then delete $\nu(p) - \mu(p)$ smoke p-nodes from $\widehat{H}(1)$ as well as all their descendants (which are all smoke nodes by definition). If $\mu(p) > \nu(p)$, then add to $\widehat{H}$ $(\mu(p) - \nu(p))$ copies of an arbitrary tree β of smoke nodes of $\widehat{H}$ rooted in $(p, 1)$. This tree exists because $p \in \|\mu\|$, and so $p \in \|\nu\|$. The addition of the copies of β maintains the "no smoke without fuel" property, because it was already fulfilled in $\widehat{H}$ by the nodes of β.

The smoke nodes of $\widehat{H'}(1)$ thus constructed are labelled by μ, and fuel and cargo nodes are neither added nor removed. We prove that $\widehat{H'}$ is realizable. Let $t_1^{k_1} \cdots t_{h-1}^{k_{h-1}}$ be a sequence that realizes $\widehat{H}$, for some transitions $t_1, \ldots, t_{h-1} \in T$ and numbers $k_1, \ldots, k_{h-1} \geq 0$.

Removing trees of smoke nodes from $\widehat{H'}$ does not affect realizability: if there is a smoke p_o-node labeled by the observed place of t_i in some level $\widehat{H}(i)$ with a child labeled the same in $\widehat{H}(i+1)$, then there is also a pair of such fuel p_o-node in $\widehat{H}(i)$ and $\widehat{H}(i+1)$ by property of smoke nodes. This pair of fuel nodes is still in $\widehat{H'}$ because we only remove trees of smoke nodes. Removing the trees translates as decreasing the iterations of some transitions in the realizing sequence of $\widehat{H}$. The trees of smoke nodes that we add to $\widehat{H'}$ also do not affect realizability: they only increase the iterations of the transitions in the realizing sequence, as in the proof of the Replacement Theorem. ◀

Deciding the Existence of Cut-Off in Parameterized Rendez-Vous Networks

Florian Horn
Université de Paris, IRIF, CNRS, France
florian.horn@irif.fr

Arnaud Sangnier
Université de Paris, IRIF, CNRS, France
sangnier@irif.fr

Abstract

We study networks of processes which all execute the same finite-state protocol and communicate thanks to a rendez-vous mechanism. Given a protocol, we are interested in checking whether there exists a number, called a cut-off, such that in any networks with a bigger number of participants, there is an execution where all the entities end in some final states. We provide decidability and complexity results of this problem under various assumptions, such as absence/presence of a leader or symmetric/asymmetric rendez-vous.

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases Parameterized networks, Verification, Cut-offs

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.46

Related Version A full version of the paper is available at `https://arxiv.org/abs/2007.05789`.

Funding Partly supported by ANR FREDDA (ANR-17-CE40-0013).

1 Introduction

Networks with many identical processes. One of the difficulty in verifying distributed systems lies in the fact that many of them are designed for an unbounded number of participants. As a consequence, to be exhaustive in the analysis, one needs to design formal methods which takes into account this characteristic. In [21], German and Sistla introduce a model to represent networks with a fix but unbounded number of entities. In this model, each participant executes the same protocol and they communicate between each other thanks to rendez-vous (a synchronization mechanism allowing two entities to change their local state simultaneously). The number of participants can then be seen as a parameter of the model and possible verification problems ask for instance whether a property holds for all the values of this parameter or seeks for some specific value ensuring a good behavior. With the increasing presence of distributed mechanisms (mutual exclusion protocols, leader election algorithms, renaming algorithms, etc) in the core of our computing systems, there has been in the last two decades a regain of attention in the study of such parameterized networks.

Surprisingly, the verification of these parameterized systems is sometimes easier than the case where the number of participants is known. This can be explained by the following reason: in the parameterized case the procedure can adapt on demand the number of participants to build a problematic execution. It is indeed what happens with the liveness verification of asynchronous shared-memory systems. This problem is PSPACE-complete for a finite number of processes and in NP when this number is a parameter [14]. It is hence worth studying the complexity of the verification of such parameterized models and many recent works have attacked these problems considering networks with different means of communication. For instance in [16, 13, 7, 6] the participants communicate thanks to

broadcast of messages, in [11, 2] they use a token-passing mechanism , in [10] a message passing mechanism and in [18] the communication is performed through shared registers. The relative expressiveness of some of those models has been studied in [4]. Finally in his survey [15], Esparza shows that minor changes in the setting of parameterized networks, such as the presence of a controller (or equivalently a leader), might drastically change the complexity of the verification problems.

Cut-off to ease the verification. When one has to prove the correctness of a distributed algorithm designed to work for an unbounded number of participants, one technique consists in proving that the algorithm has a cut-off, i.e. a bound on the number of processes such that if it behaves correctly for this specific number of processes then it will still be correct for any bigger networks. Such a property allows to reduce the verification procedure to the analysis of the algorithm with a finite number of entities. Unfortunately, as shown in [3], many parameterized systems do not have a cut-off even for basic properties. Instead of checking whether a general class of models admits a cut-off, we propose in this work to study the following problem: given a representation of a system and a class of properties, does it admit a cutoff ? To the best of our knowledge, looking at the existence of a cutoff as a decision problem is a subject that has not received a lot of attention although it is interesting both practically and theoretically. First, in the case where this problem is decidable, it allows to find automatically cutoffs for specific systems even though they belong to a class for which there is no general results on the existence of cutoff. The search of cutoffs has been studied in [1] where the authors propose a semi-algorithm for verification of parameterized networks with respect to safety properties. This algorithm stops when a cutoff is found. However it is not stated how to determine the existence of this cutoff, neither if this is possible or not. In [25], the authors propose a way to compute dynamically a cutoff, but they consider systems and properties for which they know that a cutoff exists. Second, from the theoretical point of view, the cutoff decision problem is interesting because it goes beyond the classical problems for parameterized systems that usually seek for the existence of a number of participants which satisfies a property or check that a property hold for all possible number of participants. Note that in the latter case, one might be in a situation that for a property to hold a minimum number of participants is necessary (and below this number the property does not hold), such a situation can be detected with the existence of a cutoff but not with the simple universal quantification.

Rendez-vous networks. We focus on networks where the communication is performed by rendez-vous. There are different reasons for this choice. First, we are not aware of any technique to decide automatically the existence of a cut-off in parameterized systems, it is hence convenient to look at this problem in a well-known setting. Another aspect which motivates the choice of this model is that the rendez-vous communication corresponds to a well-known paradigm in the design of concurrent/distributed systems (for instance rendez-vous in the programming languages C or Java can be easily implemented thanks to wait/notify mechanisms). Rendez-vous communication seems as well a natural feature for parameterized systems used to model for instance crowds or biological systems (at some point we consider symmetric rendez-vous which can be seen less common in computing systems but make sense for these other applications). Last but not least, rendez-vous networks are very close to population protocols [5] for which there has been in the last years a regain of interest in the community of formal methods [17, 8, 9]. Population protocols and rendez-vous networks are both based on rendez-vous communication, but in population protocols it is furthermore

required that all the fair executions converge to some accepting set of configurations (see [17] for more details). In our case, we seek for the existence of an execution ending with all the processes in a final state. The similarities between the two models let us think that the formal techniques we use could be adapted for the analysis of some population protocols.

Our contributions. We study the Cut-off Problem (C.O.P.) for rendez-vous networks. It consists in determining whether, given a protocol labeled with rendez-vous primitives, there exists a bound B, such that in any networks of size bigger than B where the processes all run the same protocol there is an execution which brings all the processes to a final state. We assume furthermore that in our network, there could be one extra entity, called the leader, that runs its own specific protocol. We first show that C.O.P. is decidable by reducing it to a new decision problem on Petri nets. Unfortunately we show as well that it is non elementary thanks to a reduction from the reachability problem in Petri nets[12]. We then show that better complexity bounds can be obtained if we assume the rendez-vous to be symmetric (i.e. any process that requests a rendez-vous can as well from the same state accept one and vice-versa) or if we assume that there is no leader. For each of these restrictions, new algorithmic techniques for the analysis of rendez-vous networks are proposed. The following table sums up the complexity bounds we obtain.

Table 1 Complexity results obtained for the Cut-Off Problem.

	Asymmetric rendez-vous	Symmetric rendez-vous
Presence of a leader	Decidable and non-elementary	PSPACE
Absence of leader	EXPSPACE	NP

Due to lack of space, omitted details and proofs can be found in [23].

2 Modeling networks with rendez-vous communication

We write $\mathbb{N}$ to denote the set of natural numbers and $[i, j]$ to represent the set $\{k \in \mathbb{N} \mid i \leq k \text{ and } k \leq j\}$ for $i, j \in \mathbb{N}$. For a finite set E, the set $\mathbb{N}^E$ represents the multisets over E. For two elements $m, m' \in \mathbb{N}^E$, we denote $m+m'$ the multiset such that $(m+m')(e) = m(e)+m'(e)$ for all $e \in E$. We say that $m \leq m'$ if and only if $m(e) \leq m'(e)$ for all $e \in E$. If $m \leq m'$, then $m' - m$ is the multiset such that $(m' - m)(e) = m'(e) - m(e)$ for all $e \in E$. The size of a multiset m is given by $|m| = \Sigma_{e \in E} m(e)$. For $e \in E$, we use sometimes the notation e for the multiset m verifying $m(e) = 1$ and $m(e') = 0$ for all $e' \in E \setminus \{e\}$ and the notation $\langle\langle e1, e1, e2, e3\rangle\rangle$ to represent the multiset with four elements $e1, e1, e2$ and $e3$.

2.1 Rendez-vous protocols

We are now ready to define our model of networks. We assume that all the entities in the network (called sometimes processes) behave similarly following the same protocol except one entity, called the leader, which might behave differently. The communication in the network is pairwise and is performed by rendez-vous through a communication alphabet Σ. Each entity can either request a rendez-vous, with the primitive $?a$, or answer to a rendez-vous, with the primitive $!a$ where a belongs to Σ. The set of actions is hence $RV(\Sigma) = \{?a, !a \mid a \in \Sigma\}$.

▶ **Definition 1** (Rendez-vous protocol). *A* rendez-vous protocol $\mathcal{P}$ *is a tuple* $\langle Q, Q_P, Q_L, \Sigma, q_i, q_f, q_i^L, q_f^L, E\rangle$ *where* Q *is a finite set of states partitioned into the processes states* Q_P *and the leader states* Q_L, Σ *is a finite alphabet,* $q_i \in Q_P$ *[resp.* $q_i^L \in Q_L$*] is the initial state of the processes [resp. of the leader],* $q_f \in Q_P$ *[resp.* $q_f^L \in Q_L$*] is the final state of the processes [resp. of the leader], and* $E \subseteq (Q_P \times RV(\Sigma) \times Q_P) \cup (Q_L \times RV(\Sigma) \times Q_L)$ *is the set of edges.*

A configuration of the rendez-vous protocol $\mathcal{P}$ is a multiset $C \in \mathbb{N}^Q$ verifying that there exists $q \in Q_L$ such that $C(q) = 1$ and $C(q') = 0$ for all $q' \in Q_L \setminus \{q\}$, in other words there is a single entity corresponding to the leader. The number of processes in a configuration C is given by $|C| - 1$. We denote by $\mathcal{C}^{(n)}$ the set of configurations C involving n processes, i.e. such that $|C| = n + 1$. The initial configuration with n processes $C_i^{(n)}$ is such that $C_i^{(n)}(q_i) = n$ and $C_i^{(n)}(q_i^L) = 1$ and $C_i^{(n)}(q) = 0$ for all $q \in Q \setminus \{q_i, q_i^L\}$. Similarly the final configuration with n processes $C_f^{(n)}$ verifies $C_f^{(n)}(q_f) = n$ and $C_f^{(n)}(q_f^L) = 1$ and $C_f^{(n)}(q) = 0$ for all $q \in Q \setminus \{q_f, q_f^L\}$. Hence in an initial configuration all the entities are in their initial state and in a final configuration they are all in their final state. The notation $\mathcal{C}$ represents the whole set of configurations equals to $\bigcup_{n\in\mathbb{N}} \mathcal{C}^{(n)}$.

We are now ready to formalize the behavior of a rendez-vous protocol. In this matter, we define the relation $\rightarrow \subseteq \bigcup_{n\geq 1} \mathcal{C}^{(n)} \times \mathcal{C}^{(n)}$ as follows : $C \rightarrow C'$ if, and only if, there is $a \in \Sigma$ and two edges $(q_1, ?a, q_2), (q_1', !a, q_2') \in E$ such that $C(q_1) > 0$ and $C(q_1') > 0$ and $C(q_1) + C(q_1') \geq 2$ and $C' = C - (q_1 + q_1') + (q_2 + q_2')$. Intuitively it means that in C there is one entity in q_1 that requests a rendez-vous and one entity in q_1' that answers to it and they both change their state to respectively q_2 and q_2'. We need the hypothesis $C(q_1) + C(q_1') \geq 2$ in case $q_1 = q_1'$. We use $\rightarrow^*$ to represent the reflexive and transitive closure of $\rightarrow$. Note that if $C \rightarrow^* C'$ then $|C| = |C'|$, in other words there is no deletion or creation of processes during an execution.

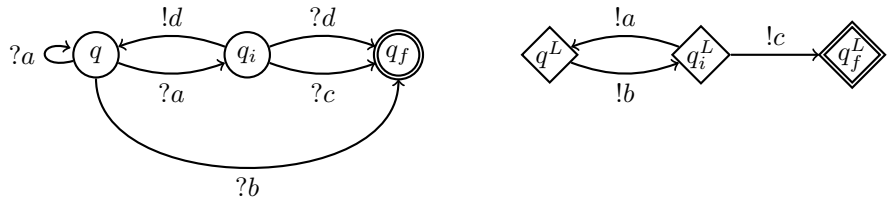


Figure 1 A rendez-vous protocol.

▶ **Example 2.** Figure 1 provides an example of rendez-vous protocol where the process states are represented by circles and the leader states by diamond.

2.2 The cut-off problem

We can now describe the problem we address. It consists in determining given a protocol whether there exists a number of processes such that if we put more processes in the network it is always possible to find an execution which brings all the entities from their initial state to their final state. This **cut-off problem (C.O.P.)** can be stated formally as follows:

- **Input:** A rendez-vous protocol $\mathcal{P}$;
- **Output:** Does there exist a cut-off $B \in \mathbb{N}$ such that $C_i^{(n)} \rightarrow^* C_f^{(n)}$ for all $n \geq B$?

▶ **Example 3.** The rendez-vous network represented in Figure 1 admits a cut-off equal to 3. For $n = 3$, we have indeed an execution $C_i^{(3)} \rightarrow^* C_f^{(3)}$: $\langle\langle q_i^L, q_i, q_i, q_i\rangle\rangle \xrightarrow{d} \langle\langle q_i^L, q_i, q, q_f\rangle\rangle \xrightarrow{a} \langle\langle q^L, q_i, q, q_f\rangle\rangle \xrightarrow{b} \langle\langle q_i^L, q_i, q_f, q_f\rangle\rangle \xrightarrow{c} \langle\langle q_f^L, q_f, q_f, q_f\rangle\rangle$ (we indicate for each transition the

label of the corresponding rendez-vous). For $n = 4$, the following sequence of rendez-vous leads to an execution $C_i^{(4)} \to^* C_f^{(4)}$: $\langle\langle q_i^L, q_i, q_i, q_i\rangle\rangle \xrightarrow{d} \langle\langle q_i^L, q_i, q_i, q, q_f\rangle\rangle \xrightarrow{a} \langle\langle q^L, q_i, q_i, q_i, q_f\rangle\rangle \xrightarrow{d} \langle\langle q^L, q_i, q, q_f, q_f\rangle\rangle \xrightarrow{b} \langle\langle q_i^L, q_i, q_f, q_f, q_f\rangle\rangle \xrightarrow{c} \langle\langle q_f^L, q_f, q_f, q_f, q_f\rangle\rangle$. Then for any $n > 4$, we can always come back to the case where $n = 3$ (if n is odd) or $n = 4$ (if n is even). In fact, we can always let 3 or 4 processes in q_i and move pairwise the other processes, one in q and one in q_f. Then the processes in q can be brought in q_f thanks to the rendez-vous a and b and the leader loop between q_i^L and q^L. Note that if we delete the edge $(q, ?a, q_i)$, this protocol does not admit anymore a cut-off but for all odd number $n \geq 3$, we have $C_i^{(n)} \to^* C_f^{(n)}$.

2.3 Petri nets

As we shall see there are some strong connections between rendez-vous protocols and Petri nets, this is the reason why we recall the definition of this latter model.

▶ **Definition 4** (Petri net). *A Petri net $\mathcal{N}$ is a tuple $\langle P, T, Pre, Post\rangle$ where P is a finite set of places, T is a finite set of transitions, $Pre : T \mapsto \mathbb{N}^P$ is the precondition function and $Post : T \mapsto \mathbb{N}^P$ is the postcondition function.*

A marking of a Petri net is a multiset $M \in \mathbb{N}^P$. A Petri net defines a transition relation $\Rightarrow \subseteq \mathbb{N}^P \times T \times \mathbb{N}^P$ such that $M \overset{t}{\Rightarrow} M'$ for $M, M' \in \mathbb{N}^P$ and $t \in T$ if and only if $M \geq Pre(t)$ and $M' = M - Pre(t) + Post(t)$. The intuition behind Petri nets is that marking put tokens in some places and each transition consumes with Pre some tokens and produces others thanks to $Post$ in order to create a new marking. We write $M \Rightarrow M'$ iff there exists $t \in T$ such that $M \overset{t}{\Rightarrow} M'$. Given a marking $M \in \mathbb{N}^P$, the reachability set of M is the set $Reach(M) = \{M' \in \mathbb{N}^P \mid M \Rightarrow^* M'\}$ where $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$. One famous problem in Petri nets is the **reachability problem**:

- **Input:** A Petri net $\mathcal{N}$ and two markings M and M';
- **Output:** Do we have $M' \in Reach(M)$?

This problem is decidable [32, 27, 28, 29] and non elementary [12]. Another similar problem that we will refer to and which is easier to solve is the **reversible reachability problem**:

- **Input:** A Petri net $\mathcal{N}$ and two markings M and M';
- **Output:** Do we have $M' \in Reach(M)$ and $M \in Reach(M')$?

It has been shown in [31] to be EXPSPACE-complete.

3 Back and forth between rendez-vous protocols and Petri nets

3.1 From Petri nets to rendez-vous protocols

We will see here how the reachability problem for Petri nets can be reduced to the C.O.P. which gives us a non-elementary lower bound for this latter problem. We consider in the sequel a Petri net $\mathcal{N} = \langle P, T, Pre, Post\rangle$ and two markings $M, M' \in \mathbb{N}^P$. Without loss of generality we can assume that M and M' are of the following form: there exists $p_i \in P$ such that $M(p_i) = 1$ and $M(p) = 0$ for all $p \in P \setminus \{p_i\}$ and there exists $p_f \in P$ such that $M'(p_f) = 1$ and $M'(p) = 0$ for all $p \in P \setminus \{p_f\}$. Taking these restrictions on the markings does not alter the complexity of the reachability problem.

We build from $\mathcal{N}$ a rendez-vous protocol $\mathcal{P}_\mathcal{N}$ which admits a cut-off if and only if $M' \in Reach(M)$. The states of the processes in $\mathcal{P}_\mathcal{N}$ are matched to the places of $\mathcal{N}$, the number of processes in a state corresponding to the number of tokens in the associated place, and the leader is in charge to move the processes in order to simulate the changing

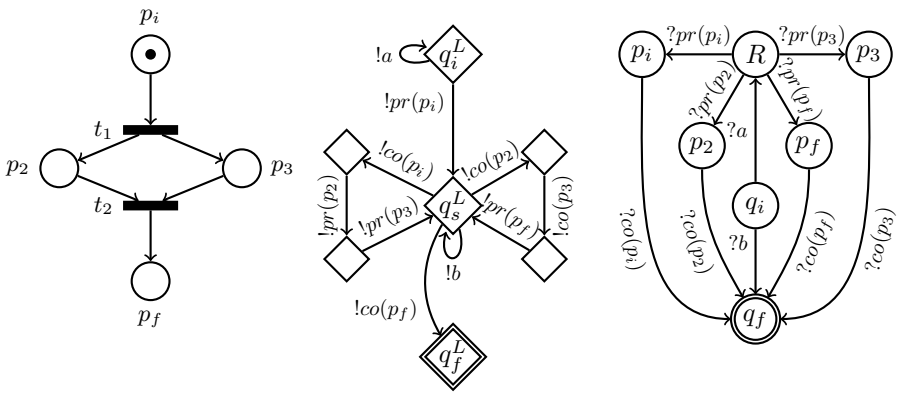


Figure 2 A Petri net $\mathcal{N}$ and its associated rendez-vous network $\mathcal{P}_\mathcal{N}$.

on the number of tokens. The protocol is equipped with an extra state R, the reserve state, where the leader stores at the beginning of the simulation the number of processes which will simulate the tokens: when a transition produces a token in a place p, the leader moves a process from R to p and when it consumes a token from a place p, the leader moves a process from p to q_f. Figure 2 provides an example of a Petri net and its associated rendez-vous network. In this net, the transition letter a is used to put as many processes as necessary to simulate the number of tokens in the places in the reserve state R. The letters $pr(p_j)$ are used to simulate the production of a token in the place p_j by moving a process from R to p_j and the letter $co(p_j)$ are used to simulate the consumption of a token in the place p_j by moving a process from p_j to q_f. It is then easy to see that each loop on the state q_s^L simulates a transition of the Petri net whereas the transition from q_i^L to q_s^L is used to build the initial marking and the transition from q_s^L to q_f^L is used to delete one token from the single place p_f and move the corresponding process to q_f. Finally, the letter b is used to ensure the cutoff property by moving from q_i to q_f the extra processes not needed to simulate the tokens. This construction gives us a hardness result for the C.O.P. thanks to the fact that the reachability problem in Petri nets is non-elementary [12].

▶ **Theorem 5.** *The C.O.P. is non-elementary.*

3.2 From rendez-vous protocols to Petri nets

We now show how to encode the behavior of a rendez-vous protocol into a Petri net and give a reduction from the C.O.P. to a problem on the built Petri net. We consider a rendez-vous protocol $\mathcal{P} = \langle Q, Q_P, Q_L, \Sigma, q_i, q_f, q_i^L, q_f^L, E\rangle$. From $\mathcal{P}$, we build a Petri net $\mathcal{N}_\mathcal{P} = \langle P, T, Pre, Post\rangle$ with $P = \{p_q \mid q \in Q\}$ and $T = \{t_i, t_f^L\} \cup \{t_{(q_1,q_2,a,q_1',q_2')} \mid q_1, q_2, q_1', q_2' \in Q$ and $a \in \Sigma$ and $(q_1, !a, q_1'), (q_2, ?a, q_2') \in E\}$. Intuitively in $\mathcal{N}_\mathcal{P}$, we have a place for each state of $\mathcal{P}$, the transition t_i puts tokens corresponding to new processes in the place corresponding to the initial state q_i, the transition t_f^L consumes a token in the place corresponding to the final state of the leader q_f^L and each transition $t_{(q_1,q_2,a,q_1',q_2')}$ simulates the protocol respecting the associated semantics (it checks that there is one process in q_1 another one in q_2 and that they can communicate thanks to the communication letter $a \in \Sigma$ moving to q_1' and q_2'). Figure 3 represents the Petri net $\mathcal{N}_\mathcal{P}$ for the protocol $\mathcal{P}$ of Figure 1 (the transitions are only labeled with the letter of the rendez-vous).

Unfortunately we did not find a way to reduce directly the C.O.P. to the reachability problem in Petri nets which would have lead directly to the decidability of C.O.P. However we will see how the C.O.P. on $\mathcal{P}$ can lead to a decision problem on $\mathcal{N}_\mathcal{P}$. We consider the initial

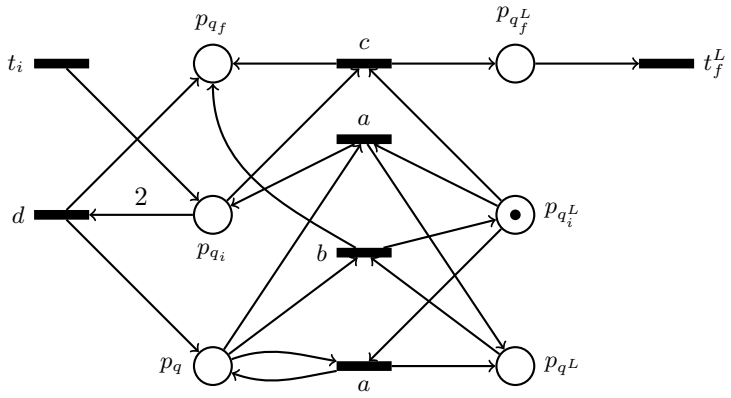


Figure 3 The Petri net $\mathcal{N}_\mathcal{P}$ for the protocol $\mathcal{P}$ of Figure 1.

marking $M_0 \in \mathbb{N}^P$ such that $M_0(p_{q_i^L}) = 1$ and $M_0(p) = 0$ for all $p \in P \setminus \{p_{q_i^L}\}$ and the family of markings $(M_f^{(n)})_{\{n\in\mathbb{N}\}}$ such that $M_f^{(n)}(p_{q_f}) = n$ and $M_f^{(n)}(p) = 0$ for all $p \in P \setminus \{p_{q_f}\}$. From the way we build the Petri net $\mathcal{N}_\mathcal{P}$, we deduce the following lemma:

▶ **Lemma 6.** *For all $n \in \mathbb{N}$, $C_i^{(n)} \to^* C_f^{(n)}$ in $\mathcal{P}$ iff $M_f^{(n)} \in Reach(M_0)$ in $\mathcal{N}_\mathcal{P}$.*

This leads us to propose a cut-off problem for Petri nets, which asks whether given an initial marking and a specific place, there exists a bound $B \in \mathbb{N}$ such that for all $n \geq B$ it is possible to reach a marking with n tokens in the specific place and none in the other. This **single place cut-off problem (single place C.O.P.)** can be stated formally as follows:

- **Input:** A Petri net $\mathcal{N}$, an initial marking M_0 and a place p_f;
- **Output:** Does there exist $B \in \mathbb{N}$ such that for all $n \geq B$, we have $M^{(n)} \in Reach(M_0)$ in $\mathcal{N}$ where $M^{(n)}$ is the marking verifying $M^{(n)}(p_f) = n$ and $M^{(n)}(p) = 0$ for all $p \in P \setminus \{p_f\}$?

Thanks to Lemma 6, we can then conclude the following proposition which justifies the introduction of the single place C.O.P. in our context.

▶ **Proposition 7.** *The C.O.P. reduces to the single place C.O.P.*

4 Solving C.O.P. in the general case

We show how to solve the C.O.P. by solving the single place C.O.P. To the best of our knowledge this latter problem has not yet been studied and we do not see direct connections with existing studied problems on Petri nets. It amounts to check if for some $B \in \mathbb{N}$ we have $\{M \in \mathbb{N}^P \mid M(p) = 0 \text{ for all } p \in P \setminus \{p_f\} \text{ and } M(p_f) \geq B\} \subseteq Reach(M_0)$. We know from [26] that the projection of the reachability set on the single place p_f is semilinear (that can be represented by a Presburger arithmetic formula), however this does not help us since we furthermore require the other places different from p_f to be empty.

4.1 Formal tools and associated results

For $\mathbf{P}, \mathbf{P}' \subseteq \mathbb{N}^n$, we let $\mathbf{P} + \mathbf{P}' = \{p + p' \mid p \in \mathbf{P} \text{ and } p' \in \mathbf{P}'\}$ and we shall sometimes identify an element $p \in \mathbb{N}^n$ with the singleton $\{p\}$. A subset $\mathbf{P}$ of $\mathbb{N}^n$ for $n > 0$ is said to be *periodic* iff $\mathbf{0} \in \mathbf{P}$ and $\mathbf{P} + \mathbf{P} \subseteq \mathbf{P}$. Such a periodic set $\mathbf{P}$ is *finitely generated* if there exists a finite set of elements $\{\mathbf{p}_1, \ldots, \mathbf{p}_k\} \subset \mathbb{N}^n$ such that $\mathbf{P} = \{\lambda_1.\mathbf{p}_1 + \ldots + \lambda_k.\mathbf{p_k} \mid \lambda_i \in \mathbb{N} \text{ for all } i \in [1, k]\}$.

A *semilinear set* of $\mathbb{N}^k$ is then a finite union of sets of the form $\mathbf{b} + \mathbf{P}$ where $\mathbf{b} \in \mathbb{N}^k$ and $\mathbf{P}$ is finitely generated. Semilinear sets are particularly useful tools because they are closed under the classical operations (union, complement and projection) and they provide a finite representation of infinite sets of vectors of naturals. Furthermore they can be represented by logical formulae expressed in Presburger arithmetic which is the decidable first-order theory of natural numbers with addition. A formula $\phi(x_1, \ldots, x_k)$ of Presburger arithmetic with free variables $x_1, \ldots, x_k$ defines a set $[\![\phi]\!] \subseteq \mathbb{N}^k$ given by $\{\mathbf{v} \in \mathbb{N}^k \mid \mathbf{v} \models \phi\}$ (here $\models$ is the classical satisfiability relation for Presburger arithmetic and it holds true if the formula holds when replacing each x_i by $\mathbf{v}[i]$). In [22], it was proven that a set $S \subseteq \mathbb{N}^k$ is semilinear iff there exists a Presburger formula ϕ such that $S = [\![\phi]\!]$. Note that the set $\{M \in \mathbb{N}^P \mid M(p) = 0 \text{ for all } p \in P \setminus \{p_f\}\}$ has a single interesting component, the other being 0. We will hence need the following result to show it is indeed semilinear.

▶ **Lemma 8.** *Every periodic subset $\mathbf{P} \subseteq \mathbb{N}$ is semilinear.*

We now recall some connections between Petri nets and semilinear sets. Let $\mathcal{N} = \langle P, T, Pre, Post\rangle$ be a Petri net with $P = \{p_1, \ldots, p_k\}$, this allows us to look at the markings as elements of $\mathbb{N}^k$ or of $\mathbb{N}^P$. Given a language of finite words of transitions $L \subseteq T^*$ and a marking M, let $Reach(M, L)$ be the reachable markings produced by L from M defined by $\{M' \subseteq \mathbb{N}^k \mid \exists w \in L \text{ such that } M \overset{w}{\Rightarrow} M'\}$ where we extend in the classical way the relation $\Rightarrow$ over words of transitions by saying $M \overset{\varepsilon}{\Rightarrow} M$ and if $w = t.w'$, we have $M \overset{w}{\Rightarrow} M'$ iff there exists M'' such that $M \overset{t}{\Rightarrow} M'' \overset{w'}{\Rightarrow} M'$. A flat expression of transitions is a regular expression over T of the form $T_1 T_2 \ldots T_\ell$ where each T_i is either a finite word in T^* or of the form w^* with $w \in T^*$. For a flat expression FE, we denote by $L(FE)$ its associated language. In [20], the following result relating flat expressions of transitions and their produced reachability set is given (it has then been extended to more complex systems [19]).

▶ **Proposition 9** ([20])**.** *Let $\mathcal{N} = \langle P, T, Pre, Post\rangle$ be a Petri net, FE a flat expression of transitions and $M \in \mathbb{N}^P$ a marking. Then $Reach(M, L(FE))$ is semilinear (and the corresponding Presburger formula can be computed).*

4.2 Deciding if a bound is a single-place cut-off

We prove that if one provides a bound $B \in \mathbb{N}$, we are able to decide whether it corresponds to a cut-off as defined in the single place C.O.P. Let $\mathcal{N} = \langle P, T, Pre, Post\rangle$ be a Petri net with an initial marking $M_0 \in \mathbb{N}^P$, a specific place $p_f \in P$ and a bound $B \in \mathbb{N}$. We would like to decide whether the following inclusion holds $\{M \in \mathbb{N}^P \mid M(p) = 0 \text{ for all } p \in P \setminus \{p_f\} \text{ and } M(p_f) \geq B\} \subseteq Reach(M_0)$. An important point to decide this inclusion lies in the fact that the set $\{M \in \mathbb{N}^P \mid M(p) = 0 \text{ for all } p \in P \setminus \{p_f\} \text{ and } M(p_f) \geq B\}$ is semilinear and this allows us to use a method similar to the one proposed in [24] to check whether the reachability set of a Petri net equipped with a semilinear set of initial markings is universal. One key point is the following result which is a reformulation of a Lemma in [30]. This result was originally stated for Vector Addition System with States (VASS), but it is well known that a Petri net can be translated into a VASS with an equivalent reachability set.

▶ **Proposition 10** ([24, Theorem 1])**.** *Let $\mathcal{N} = \langle P, T, Pre, Post\rangle$ be a Petri net, $M \in \mathbb{N}^P$ a marking and $S \subseteq \mathbb{N}^P$ a semilinear set of markings. If $S \subseteq Reach(M)$ then there is a flat expression FE of transitions such that $S \subseteq Reach(M, L(FE))$.*

Following the technique used in [24], this proposition provides us a tool to solve our inclusion problem. We use two semi-procedures, one searches for a $M' \in \{M \in \mathbb{N}^P \mid M(p) = 0 \text{ for all } p \in P \setminus \{p_f\} \text{ and } M(p_f) \geq B\}$ but not in $Reach(M_0)$ and the other one searches a flat expression of transitions FE such that $\{M \in \mathbb{N}^P \mid M(p) = 0 \text{ for all } p \in P \setminus \{p_f\} \text{ and } M(p_f) \geq B\} \subseteq Reach(M_0, L(FE))$.

▶ **Proposition 11.** *For a Petri net $\mathcal{N} = \langle P, T, Pre, Post\rangle$, a marking $M_0 \in \mathbb{N}^P$, a place $p_F \in P$ and a bound $B \in \mathbb{N}$, testing whether $\{M \in \mathbb{N}^P \mid M(p) = 0$ for all $p \in P \setminus \{p_f\}$ and $M(p_f) \geq B\} \subseteq Reach(M_0)$ is decidable.*

4.3 Finding the bound

We now show why the single-place C.O.P. is decidable. Let $\mathcal{N} = \langle P, T, Pre, Post\rangle$ be a Petri net with a marking $M_0 \in \mathbb{N}^P$ and a place $p_f \in P$. One key aspect is that the set of markings reachable from M_0 with no token in the other places except p_f is semilinear. This is a consequence of the following proposition.

▶ **Proposition 12** ([30, Lemma IX.1])**.** *Let $S \subseteq \mathbb{N}^P$ be a semilinear set of markings. Then the set $Reach(M_0) \cap S$ is a finite union of sets $\mathbf{b} + \mathbf{P}$ where $\mathbf{b} \in \mathbb{N}^P$ and $\mathbf{P} \subseteq \mathbb{N}^P$ is periodic.*

From this proposition and Lemma 8, we can deduce the following result.

▶ **Proposition 13.** *$Reach(M_0) \cap \{M \in \mathbb{N}^P \mid M(p) = 0$ for all $p \in P \setminus \{p_f\}\}$ is semilinear.*

Another key point for the decidability of the single-place C.O.P. is the ability to test whether the intersection of the reachability set of a Petri net with a linear set is empty. In fact, it reduces to the reachability problem.

▶ **Lemma 14.** *If $S \subseteq \mathbb{N}^P$ is a linear set of the form $\mathbf{b} + \mathbf{P}$ where $\mathbf{P}$ is finitely generated, then testing whether $Reach(M_0) \cap S = \emptyset$ is decidable.*

The previous results allow us to design two semi-procedures to decide the single place C.O.P. The first one enumerates the $B \in \mathbb{N}$ and uses the result of Proposition 11 to check if one is a cut-off. The other one uses the fact that if there does not exist a cut-off then the set $\{M \notin Reach(M_0) \mid M(p) = 0$ for all $p \in P \setminus \{p_f\}\}$ is semi-linear (by Proposition 13) and infinite and it includes a semi-linear set of the form $\{\mathbf{b} + \lambda.\mathbf{p} \mid \lambda \in \mathbb{N}\}$ with $\mathbf{b}, \mathbf{p} \in \mathbb{N}^P$ and $\mathbf{0} < \mathbf{p}$. In this latter case we have $Reach(M_0) \cap \{\mathbf{b} + \lambda.\mathbf{p} \mid \lambda \in \mathbb{N}\} = \emptyset$ and we use the result of Lemma 14 to enumerate the $\mathbf{b}, \mathbf{p}$ and find a pair satisfying this property.

▶ **Theorem 15.** *The single place C.O.P. is decidable.*

Thanks to Proposition 7, we obtain the result which concludes this section.

▶ **Corollary 16.** *The C.O.P. is decidable.*

5 The specific case of symmetric rendez-vous

Even though the C.O.P. is decidable, the lower bound is quite bad as mentioned in Theorem 5 and the decision procedure presented in the proof of Theorem 15 is quite technical. We show here that for a specific family of rendez-vous protocols, solving C.O.P. is easier.

5.1 Definition and basic properties

A rendez-vous protocol $\mathcal{P} = \langle Q, Q_P, Q_L, \Sigma, q_i, q_f, q_i^L, q_f^L, E\rangle$ is *symmetric* if it respects the following property: for all $q, q' \in Q$ and $a \in \Sigma$, we have $(q, !a, q') \in E$ iff $(q, ?a, q') \in E$. In this context we denote such transitions by (q, a, q'). We furthermore assume w.l.o.g. that in the underlying graph of $\mathcal{P}$ for every states q in Q_P there is a path from q_i to q and a path from q to q_f (otherwise an initial configuration can never reach a configuration with a process in q or from a configuration with a process in q a final configuration can never been reached). We now work under these hypotheses.

In symmetric rendez-vous protocols, it is always possible to bring in any state as many pairs of processes one desires from the initial state q_i and to remove as many pairs of processes (and bring them to the final state q_f). To perform such actions, it is enough to move pairs of processes following the same path (as the rendez-vous are symmetric, this is allowed by the semantics of rendez-vous protocols). We now state these properties formally. Let $\mathcal{P} = \langle Q, Q_P, Q_L, \Sigma, q_i, q_f, q_i^L, q_f^L, E\rangle$ be a symmetric rendez-vous protocol.

▶ **Lemma 17.** *Let $C \in \mathcal{C}$ verifying $C_i^{(|C|-1)} \to^* C$. Then:*

1. *for all $C' \in \mathcal{C}$ such that $C(q) \leq C'(q)$ and $(C(q) = C'(q)) \mod 2$ for all $q \in Q$, we have $C_i^{(|C'|-1)} \to^* C'$,and,*
2. *for all $C' \in \mathcal{C}$ such that $|C'| = |C|$ and $C'(q) \leq C(q)$ for all $q \in Q \setminus \{q_f\}$ and $(C(q) = C'(q)) \mod 2$ for all $q \in Q$, we have $C_i^{(|C'|-1)} \to^* C'$.*

As a consequence, we show that there is a cut-off in $\mathcal{P}$ iff a final configuration with an even number and another one with an odd number of processes are reachable in $\mathcal{P}$.

▶ **Lemma 18.** *There exists $B \in \mathbb{N}$ such that $C_i^{(n)} \to^* C_f^{(n)}$ for all $n \geq B$ iff there exists an even $n_{\mathsf{E}} \in \mathbb{N}$ and an odd $n_{\mathsf{O}} \in \mathbb{N}$ such that $C_i^{(n_{\mathsf{E}})} \to^* C_f^{(n_{\mathsf{E}})}$ and $C_i^{(n_{\mathsf{O}})} \to^* C_f^{(n_{\mathsf{O}})}$.*

5.2 The even-odd abstraction

We now present our tool to decide C.O.P. for a symmetric rendez-vous protocol $\mathcal{P} = \langle Q, Q_P, Q_L, \Sigma, q_i, q_f, q_i^L, q_f^L, E\rangle$. We build an abstraction of the transition system $(\mathcal{C}, \to)$ where we only remember the state of the leader and whether the number of processes in each state is even (denoted by E) or odd (O). Let $\widehat{\mathsf{E}} = \mathsf{O}$ and $\widehat{\widehat{\mathsf{E}}} = \mathsf{E}$. The set of even-odd configurations is $\Gamma_{\mathsf{EO}} = Q_L \times \{\mathsf{E}, \mathsf{O}\}^{Q_P}$. To an even-odd configuration $(q^L, \gamma) \in \Gamma_{\mathsf{EO}}$, we associate the set of configurations $[\![(q^L, \gamma)]\!] \subseteq \mathcal{C}$ such that $[\![(q^L, \gamma)]\!] = \{C \in \mathcal{C} \mid C(q^L) = 1 \text{ and } C(q) = 0 \mod 2 \text{ iff } \gamma(q) = \mathsf{E}\}$. We now define the even-odd transition relation $\dashrightarrow \subseteq \Gamma_{\mathsf{EO}} \times E \times E \times \Gamma_{\mathsf{EO}}$. We have $(q_1^L, \gamma_1) \overset{e,e'}{\dashrightarrow} (q_2^L, \gamma_2)$ iff one the following conditions holds:

1. $e = (q_1^L, a, q_2^L)$ and $e' = (q_1, a, q_2)$ belongs to $Q_P \times RV(\Sigma) \times Q_P$ and if $q_1 = q_2$ then $\gamma_2 = \gamma_1$ else $\gamma_2(q_1) = \widehat{\gamma_1(q_1)}$, $\gamma_2(q_2) = \widehat{\gamma_1(q_2)}$ and $\gamma_2(q) = \gamma_1(q)$ for all $q \in Q_P \setminus \{q_1, q_2\}$.
2. $e, e' \in Q_P \times RV(\Sigma) \times Q_P$ and $q_1^L = q_2^L$ and $e = (q_1, a, q_2)$ and $e' = (q_3, a, q_4)$ and there exists $\gamma' \in \{\mathsf{E}, \mathsf{O}\}^{Q_P}$ such that:
   - if $q_1 = q_2$ then $\gamma' = \gamma_1$ else $\gamma'(q_1) = \widehat{\gamma_1(q_1)}$, $\gamma'(q_2) = \widehat{\gamma_1(q_2)}$ and $\gamma'(q) = \gamma_1(q)$ for all $q \in Q_P \setminus \{q_1, q_2\}$, and,
   - if $q_3 = q_4$ then $\gamma_2 = \gamma'$ else $\gamma_2(q_3) = \widehat{\gamma'(q_3)}$, $\gamma_2(q_4) = \widehat{\gamma'(q_4)}$ and $\gamma_2(q) = \gamma'(q)$ for all $q \in Q_P \setminus \{q_3, q_4\}$.

The relation $\overset{e,e'}{\dashrightarrow}$ reflects how the parity of the number of processes changes when performing a rendez-vous involving edges e and e'. For instance, the first case illustrates a rendez-vous between the leader and a process, hence the parity of the number of states in q_1 and in

q_2 changes except when these two control states are equal. The second case deals with a rendez-vous between two processes and it is cut in two steps to take care of the cases like for instance $q_1 \neq q_2$ and $q_3 \neq q_4$ and $q_1 \neq q_4$ and $q_2 = q_3$; in fact here the parity of the number of processes in q_2 should not change, since the first transition adds one process to q_2 and the second one removes one from it. We write $(q_1^L, \gamma_1) \dashrightarrow (q_2^L, \gamma_2)$ iff there exists $e, e' \in E$ such that $(q_1^L, \gamma_1) \overset{e,e'}{\dashrightarrow} (q_2^L, \gamma_2)$ and $\dashrightarrow^*$ denotes the reflexive and transitive closure of $\dashrightarrow$.

As said earlier, $(\Gamma_{\mathsf{EO}}, \dashrightarrow)$ is an abstraction of $(\mathcal{C}, \to)$. We will prove that this abstraction is enough to solve the C.O.P. For this, we define the following abstract configurations in Γ_{EO}:

- $(q_i^L, \gamma_i^{\mathsf{E}})$ and $(q_f^L, \gamma_f^{\mathsf{E}})$ are such that $\gamma_i^{\mathsf{E}}(q) = \gamma_f^{\mathsf{E}}(q) = \mathsf{E}$ for all $q \in Q_P$;
- $(q_i^L, \gamma_i^{\mathsf{O}})$ and $(q_f^L, \gamma_f^{\mathsf{O}})$ are such that $\gamma_i^{\mathsf{O}}(q) = \gamma_f^{\mathsf{O}}(q) = \mathsf{E}$ for all $q \in Q_P \setminus \{q_i, q_f\}$ and $\gamma_i^{\mathsf{O}}(q_f) = \gamma_f^{\mathsf{O}}(q_i) = \mathsf{E}$ and $\gamma_i^{\mathsf{O}}(q_i) = \gamma_f^{\mathsf{O}}(q_f) = \mathsf{O}$.

Note that we have then $\{C_i^{(n)} \mid n \text{ is even}\} \subseteq [\![(q_i^L, \gamma_i^{\mathsf{E}})]\!]$ and $\{C_i^{(n)} \mid n \text{ is odd}\} \subseteq [\![(q_i^L, \gamma_i^{\mathsf{O}})]\!]$ and $\{C_f^{(n)} \mid n \text{ is even}\} \subseteq [\![(q_f^L, \gamma_f^{\mathsf{E}})]\!]$ and $\{C_f^{(n)} \mid n \text{ is odd}\} \subseteq [\![(q_f^L, \gamma_f^{\mathsf{O}})]\!]$. According to the definitions of the relations $\to$ and $\dashrightarrow$, we can easily deduce this first result.

▶ **Lemma 19** (Completeness). *Let $n \in \mathbb{N}$. If $C_i^{(n)} \to^* C_f^{(n)}$ and n is even [resp. n is odd] then $(q_i^L, \gamma_i^{\mathsf{E}}) \dashrightarrow^* (q_f^L, \gamma_f^{\mathsf{E}})$ [resp. $(q_f^L, \gamma_i^{\mathsf{O}}) \dashrightarrow^* (q_f^L, \gamma_f^{\mathsf{O}})$].*

The two next lemmas show that our abstraction is sound for C.O.P. The first one can be proved by induction on the length of the path in $(\Gamma_{\mathsf{EO}}, \dashrightarrow)$ using Point 1. of Lemma 17.

▶ **Lemma 20.** *If $(q_i^L, \gamma_i^{\mathsf{E}}) \dashrightarrow^* (q^L, \gamma)$ [resp. $(q_i^L, \gamma_i^{\mathsf{O}}) \dashrightarrow^* (q^L, \gamma)$] then there exists $n \in \mathbb{N} \setminus \{0\}$ such that n is even [resp. n is odd] and $C_i^{(n)} \to^* C$ with $C \in [\![(q^L, \gamma)]\!]$.*

Using Point 2. of Lemma 17 we obtain the soundness of our abstraction.

▶ **Lemma 21** (Soundness). *If $(q_i^L, \gamma_i^{\mathsf{E}}) \dashrightarrow^* (q_f^L, \gamma_f^{\mathsf{E}})$ [resp. $(q_i^L, \gamma_i^{\mathsf{O}}) \dashrightarrow^* (q_f^L, \gamma_f^{\mathsf{O}})$] then there exists $n \in \mathbb{N}$ such that n is even [resp. n is odd] and $C_i^{(n)} \to^* C_f^{(n)}$.*

Thanks to the Lemmas 18, 19 and 21 to solve the C.O.P. when the considered rendez-vous protocol is symmetric it is enough to check whether $(q_i^L, \gamma_i^{\mathsf{E}}) \dashrightarrow^* (q_f^L, \gamma_f^{\mathsf{E}})$ and $(q_i^L, \gamma_i^{\mathsf{O}}) \dashrightarrow^* (q_f^L, \gamma_f^{\mathsf{O}})$. But since the transition system $(\Gamma_{\mathsf{EO}}, \dashrightarrow)$ has a finite number of vertices whose number is bounded by $|Q_L| \cdot 2^{|Q_P|}$, these two reachability questions can be solved in NPSPACE in $|Q|$. By Savitch's theorem, we obtain the following result.

▶ **Theorem 22.** *C.O.P. restricted to symmetric rendez-vous protocols is in PSPACE.*

6 Supressing the leader

6.1 Definition and properties

A rendez-vous protocol $\mathcal{P} = \langle Q, Q_P, Q_L, \Sigma, q_i, q_f, q_i^L, q_f^L, E \rangle$ has *no leader* when $Q_L = \{q_f^L\}$ and $q_i^L = q_f^L$ and the transition relation does not refer to the state in Q_L, i.e. $E \subseteq Q_P \times RV(\Sigma) \times Q_P$. We can then assume that $\mathcal{P} = \langle Q_P, \Sigma, q_i, q_f, E \rangle$ and delete any reference to the leader state. We suppose again w.l.o.g. that in the considered rendez-vous protocols without leader there is a path from q_i to q and a path from q to q_f for all q in Q_P. Rendez-vous protocols with no leader enjoy some properties easing the resolution of the C.O.P.

▶ **Lemma 23.** *Let $\mathcal{P} = \langle Q_P, \Sigma, q_i, q_f, E \rangle$ be a rendez-vous protocol with no leader. Then the following properties hold:*

1. *If $C_i^{(n)} \to^* C_f^{(n)}$ and $C_i^{(m)} \to^* C_f^{(m)}$ for $m, n \in \mathbb{N}$, then $C_i^{(n+m)} \to^* C_f^{(n+m)}$.*
2. *There exists $B \in \mathbb{N}$ such that $C_i^{(n)} \to^* C_f^{(n)}$ for all $n \geq B$ iff there exists $N \in \mathbb{N}$ such that $C_i^{(N)} \to^* C_f^{(N)}$ and $C_i^{(N+1)} \to^* C_f^{(N+1)}$.*

Proof.

1. This point is a direct consequence of the semantics of rendez-vous protocols associated with the fact that there is no leader. In fact assume $C_i^{(n)} \to^* C_f^{(n)}$ and $C_i^{(m)} \to^* C_f^{(m)}$. And consider the configuration C such that $C(q_i) = m$, $C(q_f) = n$ and $C(q) = 0$ for all $q \in Q_P \setminus \{q_i, q_f\}$. Then it is clear that we have $C_i^{(n+m)} \to^* C \to^* C_f^{(n+m)}$, the first part of this execution mimicking the execution $C_i^{(n)} \to^* C_f^{(n)}$ and the last part mimics the execution $C_i^{(m)} \to^* C_f^{(m)}$ on the m processes left in q_i in C.
2. If there exists $B \in \mathbb{N}$ such that $C_i^{(n)} \to^* C_f^{(n)}$ for all $n \geq B$, then we have $C_i^{(B)} \to^* C_f^{(B)}$ and $C_i^{(B+1)} \to^* C_f^{(B+1)}$. Assume now that there exists $N \in \mathbb{N}$ such that $C_i^{(N)} \to^* C_f^{(N)}$ and $C_i^{(N+1)} \to^* C_f^{(N+1)}$. We show that for all $n \geq N^2$, we have $C_i^{(n)} \to^* C_f^{(n)}$. Let $n \geq N^2$ and let $R \in [0, N-1]$ be such that $(n = R) \mod N$. By definition of the modulo, there exists $A \geq 0$ such that $n = A \cdot N + R$. Since $n \geq N^2$, we have necessarily $A \geq N$. As a consequence we can rewrite n as: $n = R \cdot (N+1) + (A - R) \cdot N$. But then since $C_i^{(N)} \to^* C_f^{(N)}$, by 1. we have $C_i^{((A-R)\cdot N)} \to^* C_f^{((A-R)\cdot N)}$ and since $C_i^{(N+1)} \to^* C_f^{(N+1)}$, by 1. we have $C_i^{(R\cdot(N+1))} \to^* C_f^{(R\cdot(N+1))}$. By a last application of 1. we get $C_i^{(n)} \to^* C_f^{(n)}$. ◀

6.2 The symmetric case

We will now see how the procedure proposed in the proof of Theorem 22 to solve in polynomial space the C.O.P. for symmetric rendez-vous protocols can be simplified when there is no leader. Let $\mathcal{P} = \langle Q_P, \Sigma, q_i, q_f, E \rangle$ be a symmetric rendez-vous protocol with no leader and let $(\Gamma_{\mathsf{EO}}, \dashrightarrow)$ be the abstract transition system of $(\mathcal{C}, \to)$ as defined in Section 5.2. If we adapt the results of Lemmas 18, 19 and 21 to the no leader case, we deduce that to solve the C.O.P. it is enough to check whether $\gamma_i^{\mathsf{E}} \dashrightarrow^* \gamma_f^{\mathsf{E}}$ and $\gamma_i^{\mathsf{O}} \dashrightarrow^* \gamma_f^{\mathsf{O}}$ (we have deleted the leader states from these results). Note that by definition $\gamma_i^{\mathsf{E}} = \gamma_f^{\mathsf{E}}$, hence the only thing to verify is if $\gamma_i^{\mathsf{O}} \dashrightarrow^* \gamma_f^{\mathsf{O}}$ holds. This check can be made efficiently using the fact that there is no leader, because any reodering of a path is still a path in $(\Gamma_{\mathsf{EO}}, \dashrightarrow)$ (since we do not need to worry anymore about the leader state) and we can delete the pairs of edges that consecutively repeat since they have the same action on the parity.

▶ **Lemma 24.** *If $\gamma \dashrightarrow^* \gamma'$ then there exists $k \leq |E|^2$ and $e_1, e_1', e_2, e_2', \ldots, e_k, e_k' \in E$ such that $\gamma \overset{e_1,e_1'}{\dashrightarrow} \gamma_1 \overset{e_2,e_2'}{\dashrightarrow} \ldots \overset{e_k,e_k'}{\dashrightarrow} \gamma'$.*

It means that if $\gamma_i^{\mathsf{O}} \dashrightarrow^* \gamma_f^{\mathsf{O}}$ then there is a path of polynomial length (in the size of $\mathcal{P}$) between these two abstract configurations. It is hence enough to guess such a sequence of polynomial length and to check that it effectively corresponds to a path in $(\Gamma_{\mathsf{EO}}, \dashrightarrow)$.

▶ **Theorem 25.** *C.O.P. for symmetric rendez-vous protocols with no leader is in NP.*

6.3 Upper bound for the C.O.P. with no leader

We now prove that the C.O.P. for rendez-vous protocols with no leader reduces to the reversible reachability problem in Petri nets. Let $\mathcal{P} = \langle Q_P, \Sigma, q_i, q_f, E \rangle$ be a rendez-vous protocol with no leader and such that w.l.o.g. there is no edge going out of q_f[1].

[1] To achieve this, we can simply duplicate q_f adding a new final state q_f' and for each edge going into q_f we add an edge from the same state to q_f'

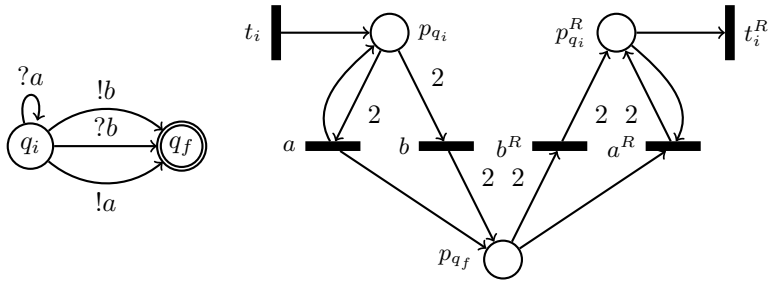


Figure 4 A rendez-vous protocol with no leader $\mathcal{P}$ and the associated Petri net $\mathcal{N}'_{\mathcal{P}}$.

Let $\mathcal{N}_{\mathcal{P}} = \langle P, T, Pre, Post\rangle$ be the Petri net whose construction is provided in Section 3.2 (where we have removed all the places corresponding to leader states as well as the transition t^L_f). From $\mathcal{N}_{\mathcal{P}}$, we build the reverse Petri net $\mathcal{N}^R_{\mathcal{P}}$ obtained by keeping the same set of places and reversing all the transitions. Formally $\mathcal{N}^R_{\mathcal{P}} = \langle P^R, T^R, Pre^R, Post^R\rangle$, where $P^R = \{p^R \mid p \in P\}$, $T^R = \{t^R \mid t \in T\}$ and for all $p^R \in P^R$ and $t^R \in T^R$, we have $Pre^R(t^R)(p^R) = Post(t)(p)$ and $Post^R(t^R)(p^R) = Pre(t)(p)$. Let M^R_0 be the marking such that $M^R_0(p^R) = 0$ for all $p^R \in P^R$ and $(M^{R,(n)}_f)_{\{n\in\mathbb{N}\}}$ be the family of markings verifying $M^{R,(n)}_f(p^R_{q_f}) = n$ and $M^{R,(n)}_f(p) = 0$ for all $p \in P^R \setminus \{p^R_{q_f}\}$. A direct consequence of Lemma 6 and of the definition of $\mathcal{N}^R_{\mathcal{P}}$ is that $C^{(n)}_i \to^* C^{(n)}_f$ iff $M^R_0 \in Reach(M^{R,(n)}_f)$ for all $n \in \mathbb{N}$.

From $\mathcal{N}_{\mathcal{P}}$ and $\mathcal{N}^R_{\mathcal{P}}$, we build the Petri net $\mathcal{N}'_{\mathcal{P}}$ obtained by taking the disjoint unions of places and transitions of the two nets except for the place p_{q_f} and $p^R_{q_f}$ which are merged in a single place p_{q_f}. Formally, $\mathcal{N}'_{\mathcal{P}} = \langle P', T', Pre', Post'\rangle$ where $P' = (P \cup P^R) \setminus \{p^R_{q_f}\}$, $T' = T \cup T^R$, $Pre'(t)(p) = Pre(t)(p)$ and $Post'(t)(p) = Post(t)(p)$ and $Pre'(t)(p^R) = Post'(t)(p^R) = 0$ for all $p \in P$, $p^R \in P^R$ and $t \in T$, $Pre'(t^R)(p^R) = Pre^R(t^R)(p^R)$ and $Post'(t^R)(p^R) = Post^R(t^R)(p^R)$ and $Pre'(t^R)(p) = Post'(t^R)(p) = 0$ for all $p^R \in P^R$, $p \in P \setminus \{p_{q_f}\}$ and $t \in T$, and $Pre'(t^R)(p_{q_f}) = Pre^R(t^R)(p^R_{q_f}))$ and $Post'(t^R)(p_{q_f}) = Post^R(t^R)(p^R_{q_f}))$ (this last case corresponds to the merging of p_{q_f} and $p^R_{q_f}$). Figure 4 provides an example of this latter Petri net.

We now explain why this new net is useful to solve the C.O.P. when there is no leader. First remember that thanks to Point 2. of Lemma 23 it is enough to check whether there exists $N \in \mathbb{N}$ such that $C^{(N)}_i \to^* C^{(N)}_f$ and $C^{(N+1)}_i \to^* C^{(N+1)}_f$. Intuitively, in $\mathcal{N}'_{\mathcal{P}}$ this property will be witnessed by the fact that we can bring $N+1$ tokens in p_{q_f} using transitions in T and remove N tokens from p_{q_f} thanks to the transitions in T^R letting hence one token in p_{q_f} and similarly if there is already a token in p_{q_f} we can bring N others and remove afterwards $N+1$. As for $\mathcal{N}_{\mathcal{P}}$, we let M_0 be the marking with no token, and $(M^{(n)})_{\{n\in\mathbb{N}\}}$ be the family of markings such that $M^{(n)}(p_{q_f}) = n$ and $M^{(n)}(p) = 0$ for all $p \in P' \setminus \{p_{q_f}\}$. Note that since there is no leader, we have here $M_0 = M^{(0)}$. The next lemma states the correctness of our reduction to the reversible reachability problem.

▶ **Lemma 26.** *There exists $N \in \mathbb{N}$ such that $C^{(N)}_i \to^* C^{(N)}_f$ and $C^{(N+1)}_i \to^* C^{(N+1)}_f$ iff $M^{(1)} \in Reach(M_0)$ and $M_0 \in Reach(M^{(1)})$ in the Petri net $\mathcal{N}'_{\mathcal{P}}$.*

Since we know that the reversible reachability problem for Petri net is EXPSPACE-complete [31], we obtain the following complexity result.

▶ **Theorem 27.** *C.O.P. restricted to rendez-vous protocols with no leader is in EXPSPACE.*

We were not able to propose a lower bound for the C.O.P. apart for the general case, but when there is no leader, we know that there is a protocol which admits a cut-off whose value is exponential in the size of a protocol. This protocol is shown on Figure 5. To bring

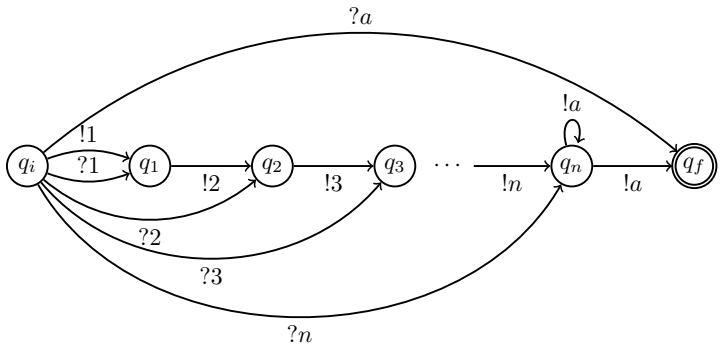


Figure 5 A rendez-vous protocol with no leader and an exponential cut-off.

a process in q_1, we need in fact two processes, to bring a process in q_2 and empty q_1, we need four processes and so on. The letter a is then used to ensure that as soon as we have processes only in q_n and in q_i (and at least one of them in each of these states), there is a way to bring all of them in q_f.

7 Conclusion

We have shown here that the C.O.P. is decidable for rendez-vous networks. Furthermore we have provided complexity upper bounds when considering restrictions on the networks such as symmetric rendez-vous or absence of leader. Unfortunately, we did not succeed in finding matching lower bounds. Reducing other problems to the C.O.P. is in fact tedious without leader or when allowing only symmetric rendez-vous, because it is then quite hard to enforce that a specific number of processes are in some states which is a property that is in general needed to design reductions. However we have some hope to either improve our upper bounds or find matching lower bounds. We wish as well to understand in which matters the techniques we used could be adapted to other parameterized systems and more specifically to population protocols. Finally, one of the justification to consider the cutoff problem is that in some distributed systems it could be the case that a correctness property does not hold for any number of processes, but that a minimal number of participants is needed to reach a goal. It could be interesting to study a variant of our cutoff problem where we do not require all the processes to reach a final state but we want to know given a number of processes how many among them can be brought in such a state. An interesting property could be to check whether there exists a bound b such that for any number of processes, the minimal number that can not be brought to a final state by any execution is always lower than b. In such networks, it would mean that at most b entities have to be sacrificed to let the others reach the final state.

References

1 Parosh Aziz Abdulla, Frédéric Haziza, and Lukás Holík. Parameterized verification through view abstraction. *STTT*, 18(5):495–516, 2016.

2 Benjamin Aminof, Swen Jacobs, Ayrat Khalimov, and Sasha Rubin. Parametrized model checking of token-passing systems. In *VMCAI'14*, volume 8318 of *LNCS*, pages 262–281. Springer-Verlag, 2014.

3 Benjamin Aminof, Tomer Kotek, Sasha Rubin, Francesco Spegni, and Helmut Veith. Parameterized model checking of rendezvous systems. *Distributed Computing*, 31(3):187–222, 2018.

4 Benjamin Aminof, Sasha Rubin, and Florian Zuleger. On the expressive power of communication primitives in parameterised systems. In *LPAR'15*, volume 9450 of *LNCS*, pages 313–328. Springer-Verlag, 2015.

5 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

6 Nathalie Bertrand, Patricia Bouyer, and Anirban Majumdar. Reconfiguration and message losses in parameterized broadcast networks. In *CONCUR'19*, volume 140 of *LIPIcs*, pages 32:1–32:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

7 Nathalie Bertrand, Miheer Dewaskar, Blaise Genest, Hugo Gimbert, and Adwait Amit Godbole. Controlling a population. *Logical Methods in Computer Science*, 15(3), 2019.

8 Michael Blondin, Javier Esparza, and Stefan Jaax. Peregrine: A tool for the analysis of population protocols. In *CAV'18*, volume 10981 of *LNCS*, pages 604–611. Springer, 2018.

9 Michael Blondin, Javier Esparza, and Stefan Jaax. Expressive power of broadcast consensus protocols. In *CONCUR'19*, volume 140 of *LIPIcs*, pages 31:1–31:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

10 Benedikt Bollig, Paul Gastin, and Len Schubert. Parameterized verification of communicating automata under context bounds. In *RP'14*, volume 8762 of *LNCS*, pages 45–57. Springer-Verlag, 2014.

11 Edmund M. Clarke, Muralidhar Talupur, Tayssir Touili, and Helmut Veith. Verification by network decomposition. In *CONCUR'04*, volume 3170 of *LNCS*, pages 276–291. Springer-Verlag, 2004.

12 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for petri nets is not elementary. In *STOC'19*, pages 24–33. ACM, 2019.

13 Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In *CONCUR'10*, volume 6269 of *LNCS*, pages 313–327. Springer-Verlag, 2010.

14 Antoine Durand-Gasselin, Javier Esparza, Pierre Ganty, and Rupak Majumdar. Model checking parameterized asynchronous shared-memory systems. *Formal Methods in System Design*, 50(2-3):140–167, 2017.

15 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In *STACS'14*, volume 25 of *LIPIcs*, pages 1–10. Leibniz-Zentrum für Informatik, 2014.

16 Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *LICS'99*, pages 352–359. IEEE Comp. Soc. Press, July 1999.

17 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Inf.*, 54(2):191–215, 2017.

18 Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. In *CAV'13*, volume 8044 of *LNCS*, pages 124–140. Springer-Verlag, 2013.

19 Alain Finkel and Jérôme Leroux. How to compose presburger-accelerations: Applications to broadcast protocols. In *FST TCS'02*, volume 2556 of *LNCS*, pages 145–156. Springer, 2002.

20 Laurent Fribourg. Petri nets, flat languages and linear arithmetic. In *WFLP'00*, pages 344–365, 2000.

21 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992.

22 Seymour Ginsburg and Edwin H. Spanier. Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.

23 Florian Horn and Arnaud Sangnier. Deciding the existence of cut-off in parameterized rendez-vous networks. *CoRR*, abs/2007.05789, 2020. `arXiv:2007.05789`.

24 Petr Jancar, Jérôme Leroux, and Grégoire Sutre. Co-finiteness and co-emptiness of reachability sets in vector addition systems with states. In *PETRI NETS'18*, volume 10877 of *LNCS*, pages 184–203. Springer, 2018.

25 Alexander Kaiser, Daniel Kroening, and Thomas Wahl. Dynamic cutoff detection in parameterized concurrent programs. In *CAV'10*, volume 6174 of *LNCS*, pages 645–659. Springer, 2010.

26 Hans Kleine Büning, Theodor Lettmann, and Ernst W. Mayr. Projections of vector addition system reachability sets are semilinear. *Theor. Comput. Sci.*, 64(3):343–350, 1989.

27 S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *STOC'82*, pages 267–281. ACM, 1982.

28 Jean-Luc Lambert. A structure to decide reachability in petri nets. *Theor. Comput. Sci.*, 99(1):79–104, 1992.

29 Jérôme Leroux. Vector addition system reachability problem: a short self-contained proof. In *POPL'11*, pages 307–316. ACM, 2011.

30 Jérôme Leroux. Presburger vector addition systems. In *LICS'13*, pages 23–32. IEEE Computer Society, 2013.

31 Jérôme Leroux. Vector addition system reversible reachability problem. *Logical Methods in Computer Science*, 9(1), 2013.

32 Ernst W. Mayr. An algorithm for the general petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.

Parametrized Universality Problems for One-Counter Nets

Shaull Almagor
Technion – Israel Institute of Technology, Haifa, Israel

Udi Boker
Interdisciplinary Center (IDC) Herzliya, Israel

Piotr Hofman
University of Warsaw, Poland

Patrick Totzke
University of Liverpool, UK

Abstract

We study the language universality problem for One-Counter Nets, also known as 1-dimensional Vector Addition Systems with States (1-VASS), parameterized either with an initial counter value, or with an upper bound on the allowed counter value during runs. The language accepted by an OCN (defined by reaching a final control state) is monotone in both parameters. This yields two natural questions: 1) does there exist an initial counter value that makes the language universal? 2) does there exist a sufficiently high ceiling so that the bounded language is universal?

Although the ordinary universality problem is decidable (and Ackermann-complete) and these parameterized variants seem to reduce to checking basic structural properties of the underlying automaton, we show that in fact both problems are undecidable. We also look into the complexities of the problems for several decidable subclasses, namely for unambiguous, and deterministic systems, and for those over a single-letter alphabet.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Counter net, VASS, Unambiguous Automata, Universality

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.47

Related Version A full version of the paper is [5], available at `https://arxiv.org/abs/2005.03435`.

Funding *Shaull Almagor*: European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 837327.
Udi Boker: Israel Science Foundation grant 1373/16.
Piotr Hofman: Supported by the NCN grant 2017/27/B/ST6/02093.

Acknowledgements We are grateful for fruitful discussions during the Autoboz'2019 workshop.

1 Introduction

One-Counter Nets (OCNs) are finite-state machines equipped with an integer counter that cannot decrease below zero and which cannot be explicitly tested for zero. They are the same as 1-dimensional Vector Addition Systems (or Petri nets with exactly one unbounded place). In order to use them as formal language acceptors we assume that transitions are labelled with letters from a finite alphabet and that some states are marked as accepting.

OCNs are a syntactic restriction of One-Counter Automata – Minsky Machines with only one counter, which can have zero-tests, i.e., transitions that depend on the counter value being exactly zero. If counter updates are restricted to ± 1, the model corresponds to Pushdown automata with a single-letter stack alphabet. OCNs are one of the simplest types of discrete infinite-state systems, which makes them suitable for exploring the decidability border of classical decision problems from automata and formal-language theory.

Universality Problems. The universality problem for a class of automata asks if a given automaton accepts all words over its input alphabet. Due to their lack of an explicit zero-test, OCNs are monotone with respect to counter values: if it is possible to make an a-labelled step from a configuration with state p and counter n to state q with counter $n+d$, written as $(p,n) \xrightarrow{a} (q, n+d)$ here, then the same holds for any larger counter value $m \geq n$: $(p,m) \xrightarrow{a} (q, m+d)$. Consequently, if we define the language via acceptance by reaching a final control state, then for all states s and $n \leq m \in \mathbb{N}$, the language $\mathcal{L}(s,n)$ of the initial configuration (s,n) is included in that of (s,m). This motivates our first variation of the universality problem. The *Initial-Value Universality* problem asks if there exists a sufficiently large initial counter to make the resulting language universal.

Input:	An OCN with alphabet Σ and an initial state s_0.
Question:	Does there exist $c_0 \in \mathbb{N}$ such that $\mathcal{L}(s_0, c_0) = \Sigma^*$?

The second question we consider is the *Bounded Universality* problem, which asks if there exists a large enough upper bound on the counter so that every word can be accepted via a run that remains within this bound. Writing $\mathcal{L}^{\leq b}(s_0, c_0) \subseteq \Sigma^*$ for the b-bounded language from configuration (s_0, c_0), the decision problem is as follows.

Input:	An OCN with alphabet Σ, an initial state s_0, and $c_0 \in \mathbb{N}$.
Question:	Does there exist $b \in \mathbb{N}$ such that $\mathcal{L}^{\leq b}(s_0, c_0) = \Sigma^*$?

The motivation for studying these parameterized problems comes from the observation that the "vanilla" universality problem, without existentially quantifying over parameters, is decidable, but Ackermann-complete [15], and the lower bound depends strongly on the assumption that we start with a fixed initial counter (and that its value is not bounded). The two new variants of the universality problem relax these assumptions in an attempt to allow efficient decision procedures via simple cycle analysis or similar.

Our Results. We show that both initial-value universality and bounded universality are undecidable (Section 3). The proofs use techniques from weighted automata [12, 4], reducing the halting problem of two-counter machines to our setting.

In light of these negative results, we proceed to study restricted classes of OCNs, for which the problems become decidable, as we elaborate below. In most cases, the complexity crucially depends on how transition updates are encoded: we consider both the case of "succinct", binary-encoded updates, and the case of unary-encoded updates, which corresponds to systems where transitions can only update the counter by ± 1.

The most intricate and interesting case is that of OCNs over a single-letter alphabet (Section 4). In order to analyze this model, we split universality to criteria on "short" words, and on longer words that admit a cyclic behavior. In particular, we devise a canonical representation of "pumpable" paths, akin to the so-called linear-path schemes [18, 7]. We show that the complexity of some of the problems is coNP complete, where others range between coNP and $\mathsf{coNP}^{\mathsf{NP}}$ (see Tables 1 and 2).

We then consider deterministic, and unambiguous OCNs (Sections 5 and 6, respectively). For such systems, deciding (bounded) universality problems mostly reduces to checking simple conditions on the cyclic structure of the control automaton underlying the OCN. Based on known (but in some cases very recent) results on unambiguous finite automata and vector-addition systems, we derive relatively low complexity upper bounds, in polynomial time (assuming unary encoding) and space (assuming binary encoding). Tables 1 and 2 summarize the status quo, following our results.

Table 1 The complexity of the universality problems of one-counter nets in which weights are encoded in unary.

Unary encoding	Universality		Initial-Value Universality		Bounded Universality	
	Singleton Alphabet	General Alphabet	Singleton Alphabet	General Alphabet	Singleton Alphabet	General Alphabet
Deterministic	L Theorem 28	NL-comp. Theorem 26	L Theorem 28	NL-comp. Theorem 26	L Theorem 28	NL-comp. Theorem 26
Unambiguous	NL Theorem 31	NC^2; NL-hard [11]	NL Theorem 34	NC^2 Theorem 34	NL Theorem 36	NC^2 Theorem 36
Non-deterministic	coNP-comp. Theorem 10	Ackermann [15]	coNP-comp. Theorem 15	Undecidable Theorem 1	coNP-comp. Theorem 22	Undecidable Theorem 2

Table 2 The complexity of the bounded universality problems of one-counter nets in which weights are encoded in binary.

Binary encoding	Universality		Initial-Value Universality		Bounded Universality	
	Singleton Alphabet	General Alphabet	Singleton Alphabet	General Alphabet	Singleton Alphabet	General Alphabet
Deterministic	NC^2 Theorem 28	NC Theorem 26	NC^2 Theorem 28	NC^2 Theorem 34	NC^2 Theorem 28	NC Theorem 26
Unambiguous	coNP-comp. Theorem 12	PSPACE; coNP-hard [11]	NC^2 Theorem 34	NC^2 Theorem 34	$coNP^{NP}$ Theorem 22	PSPACE Theorem 36
Non-deterministic	$coNP^{NP}$ Theorem 12	Ackermann [15]	coNP-comp. Theorem 15	Undecidable Theorem 1	$coNP^{NP}$ Theorem 22	Undecidable Theorem 2

Related work. The undecidability of language universality for pushdown automata is textbook. In his 1973 PhD thesis [24], Valiant showed that the problem remains undecidable for the strictly weaker model of one-counter automata (OCA, with zero tests) by recognizing the complement of all accepting runs of a two-counter machine. Language inclusion is undecidable for the further restricted model of OCNs [14]. If one considers ω-regular languages defined by OCNs with Büchi acceptance condition then the resulting universality problem is undecidable [8].

On the positive side, universality is decidable for vector addition systems [16] and Ackermann-complete for the special case of OCNs [15]. One-counter systems have received some attention in regards to checking bisimulation and simulation relations, which under-approximate language equivalence (and inclusion, respectively) and are computationally simpler. For OCAs/OCNs, bisimulation is PSPACE-complete [9], while weak bisimulation is undecidable for OCNs [19]. Both strong and weak simulation are PSPACE-complete for OCNs, and checking if an OCN simulates an OCA is decidable [1].

Universality problems for OCNs over single-letter alphabets are related to the termination problem for VASS, which asks if there exists an infinite run. Non-termination naturally corresponds to the property that $a^n \in \mathcal{L}(s_0, \mathbf{v_0})$, i.e., all finite words are accepted, assuming that all states are accepting. Termination reduces to boundedness (finiteness of the reachability set) which is EXPSPACE-complete [21, 13] in general and PSPACE-complete for systems with fixed dimensions [22]. In contrast, the *structural* termination problem (there exists no infinite run, regardless of the initial configuration) is equivalent to finding an executable cycle that is non-decreasing on all dimensions, and can be solved in polynomial time [17].

Finally, the idea to existentially quantify over some initial resource is commonplace in the formal verification literature. Examples include unknown initial-credit problems for energy games [10, 1] and R-Automata [3], timed Petri nets [2], and inclusion problems for weighted automata [12, 4].

2 Preliminaries

One-Counter Nets. A *one-counter net* (OCN) is a finite directed graph where edges carry both an integer weight and a letter from a finite alphabet. We write $\mathcal{A} = (\Sigma, Q, s_0, \delta, F)$ for the net $\mathcal{A}$ where Q is a finite set of *states*, Σ is a finite set of *letters*, $s_0 \in Q$ is an *initial state*, $\delta \subseteq Q \times \Sigma \times \mathbb{Z} \times Q$ is the *transition* relation, and $F \subseteq Q$ are the *accepting* states.

For a transition $t = (s, a, e, s') \in \delta$ we write $\mathit{effect}(t) \stackrel{\mathit{def}}{=} e$ for its (counter) *effect*, and write $\|\delta\|$ for the largest absolute effect among all transitions. By the *underlying automaton* of an OCN we mean the NFA obtained from the OCN by disregarding the transition effects.

A path in the OCN is a sequence $\pi = (s_1, a_1, e_1, s_2)(s_2, a_2, e_2, s_3)\dots(s_k, a_k, e_k, s_{k+1}) \in \delta^*$. Such a path π is a *cycle* if $s_1 = s_{k+1}$, and is a *simple cycle* if no other cycle is a proper infix of it. We say that the path above *reads* word $a_1 a_2 \dots a_k \in \Sigma^*$ and is accepting if $s_{k+1} \in F$. Its $\mathit{effect}(\pi) \stackrel{\mathit{def}}{=} \sum_{i=1}^{k} e_i$ is the sum of its transition effects . Its *height* is the maximal effect of any prefix and, similarly, its *depth* is the inverse of the minimal effect of any prefix.

An OCN naturally induces an infinite-state labelled transition system in which each *configuration* is a pair $(s, c) \in Q \times \mathbb{N}$ comprising a state and a non-negative integer. We call such a configuration *final*, or *accepting*, if $s \in F$. Every letter $a \in \Sigma$ induces a step relation $\xrightarrow{a} \subseteq (Q \times \mathbb{N})^2$ between configurations where, for every two configurations (s, c) and (s', c'),

$$(s, c) \xrightarrow{a} (s', c') \iff (s, a, d, s') \in \delta \quad \text{and } c' = c + d.$$

A *run* on a word $w = a_1 a_2 \dots a_k \in \Sigma^*$ is a path in this induced infinite system; that is, a sequence $\rho = (s_0, c_0), (s_1, c_1), (s_2, c_2), \dots (s_k, c_k)$ such that $(s_{i-1}, c_{i-1}) \xrightarrow{a_i} (s_i, c_i)$ holds for all $1 \le i \le k$. Naturally, a run uniquely describes a path in the underlying finite OCN. Conversely, for every such path and initial counter value $c_0 \in \mathbb{N}$, there is at most one corresponding run: A path π is *executable from* c_0 if its depth is at most c_0 (that is, we do not allow the counter to become negative). A run as above is called a (simple) *cycle* if its underlying path is a (simple) cycle. It is *accepting* if it ends in an accepting configuration. We call a run *bounded* by $b \in \mathbb{N}$ if $c_i \le b$ for all $0 \le i \le k$.

For any fixed initial configuration (s, c), we define its *language* $\mathcal{L}_{\mathcal{A}}(s, c) \subseteq \Sigma^*$ to contain exactly all words on which an accepting run starting in (s, c) exists. (We omit the subscript $\mathcal{A}$ if the OCN is clear from context.) Similarly, the *b-bounded language* $\mathcal{L}^{\le b}(s, c)$ is the set of those words on which there is a b-bounded run starting in (s, c).

The OCN is *deterministic* if for every pair $(s, a) \in Q \times \Sigma$ there is at most one pair $(d, q) \in \mathbb{N} \times Q$ with $(s, a, d, s') \in \delta$. A net together with an initial configuration (s_0, c_0) is *unambiguous* if for every word $w \in \Sigma^*$ there is at most one accepting run starting in (s_0, c_0).

Two-Counter Machines. A two-counter machine (Minsky Machine) $\mathcal{M}$ is a sequence $(l_1, \dots, l_n)$ of commands involving two counters x and y. We refer to $\{1, \dots, n\}$ as the *locations* of the machine. There are five possible forms of commands: `inc(c)`, `dec(c)`, `goto` l_i, `halt`, `if c=0 goto` l_i `else goto` l_j, where $c \in \{x, y\}$ is a counter and $1 \le i, j \le n$ are locations. The counters are initially set to 0. Since we can always check whether $c = 0$ before a `dec(c)` command, we assume that the machine never reaches `dec(c)` with $c = 0$. That is, the counters never have negative values.

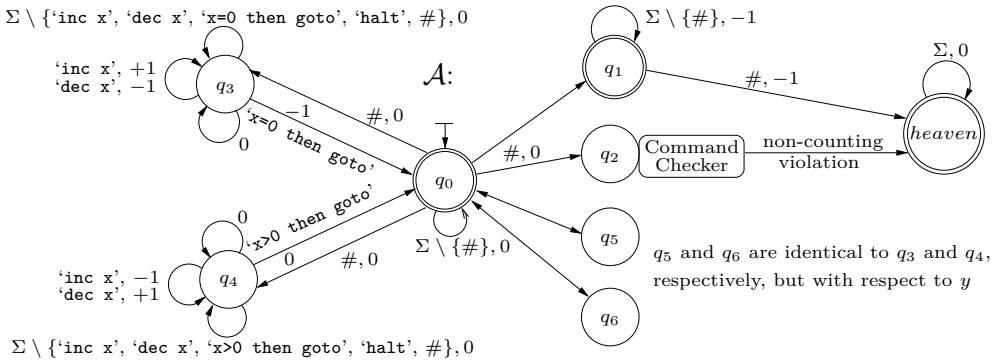


Figure 1 The one-counter net $\mathcal{A}$ from the proof of Theorem 1.

3 Undecidability

We show that both initial-value universality and bounded universality are undecidable by reduction from the undecidable halting problem of two-counter machines (2CM) [20].

The idea underlying both reductions is that the initial counter value, or the bound on the allowed counter, prescribes a bound on the number of steps until the OCN must make a decision weather the input word, which encodes a prefix of the run of the 2CM, either halts or cheats. After this decision the OCN is reset and continues to read the remaining word within an adjusted bound. If the decision was correct then the bound remains the same and otherwise, it is strictly reduced. The existence of a halting run of the 2CM now implies that its length corresponds to a sufficient initial bound for this simulating OCN to be universal. Conversely, if the run of the machine does not halt then for every bound n, there exists a non-cheating, and non-terminating prefix of length n. Repeating this prefix n times witnesses non-universality for the simulating OCN with initial counter n.

3.1 Initial-Value Universality

Given a two-counter machine $\mathcal{M}$, we construct a one-counter net $\mathcal{A}$ as follows (see Figure 1). Intuitively, an input word w to $\mathcal{A}$ is a sequence of segments separated by #, where each segment is a sequence of commands from $\mathcal{M}$. Accordingly, the alphabet of $\mathcal{A}$ consists of # and all possible commands of $\mathcal{M}$.

We build $\mathcal{A}$ to accept w, once starting with a big enough initial counter value, if one of the following conditions holds: i) one of w's segments is shorter than the length of the (legal halting) run of $\mathcal{M}$; or ii) one of w's segments does not respect the control structure underlying $\mathcal{M}$, which is called a "non-counting cheat" here; or iii) all of w's segments do not describe a prefix of the run of $\mathcal{M}$, making "counting cheats". The OCN reads every segment in between two #'s starting in, and returning to, a central state q_0.

Non-counting cheats are easy to verify – for every line l of $\mathcal{M}$, there is a corresponding state q in $\mathcal{A}$, and when $\mathcal{A}$ is at state q and reads a letter a, $\mathcal{A}$ checks if a matches the command in l. For example, if l ='`goto i`' and a = '`inc x`', the transition from q goes to a forever accepting state (*heaven*), and if a ='`goto i`', it goes to the state of $\mathcal{A}$ that corresponds to the line l_i. This is the "command-checker gadget" of $\mathcal{A}$.

Counting cheats are more challenging to verify, as OCNs cannot branch according to a counter value. We consider separately "positive cheats" and "negative cheats". The former stands for the case that the input letter is '`x=0 then goto`' (or '`y=0 then goto`') while the

value of x (or y) in the legal run of $\mathcal{M}$ should be positive. The latter stands for the case that the input letter is '`x>0 then goto`' (or '`y>0 then goto`') while the value of x (or y) in the legal run of $\mathcal{M}$ should be 0.

Positive cheats can be verified by directly simulating the respective counter of $\mathcal{M}$ using the counter in $\mathcal{A}$ (states q_3 and q_5 in Figure 1). Once the cheat occurs, $\mathcal{A}$ can return to q_0 with a penalty of -1, and since the counter in $\mathcal{M}$ is positive, we are guaranteed that the counter in $\mathcal{A}$ did not decrease since leaving q_0, allowing $\mathcal{A}$ to continue the run.

For verifying a negative cheat, we simulate the counting of $\mathcal{M}$ by an "opposite-counting" in $\mathcal{A}$ (states q_4 and q_6 in Figure 1), whereby an increment of the counter in $\mathcal{M}$ results in a decrement of the counter in $\mathcal{A}$, and vice versa – once the cheat occurs, $\mathcal{A}$ can return to q_0 with no penalty, and since the counter in $\mathcal{M}$ is 0, we are guaranteed that the counter in $\mathcal{A}$ did not decrease since leaving q_0, allowing $\mathcal{A}$ to continue the run.

Formally, we construct $\mathcal{A}$ from $\mathcal{M}$ as follows.

- The alphabet Σ of $\mathcal{A}$ consists of # and the descriptive commands for the counter machine $\mathcal{M}$: '`inc x`', '`inc y`', '`dec x`', '`dec y`', '`halt`', and for every line i of $\mathcal{M}$, the commands '`goto i`', '`x=0 then goto i`', '`y=0 then goto i`', '`x>0 then goto i`', and '`y>0 then goto i`'.
- The initial state q_0 is accepting, it has a self transition over $\Sigma \setminus \{\#\}$ and nondeterministic transitions to the states $q_1 \dots q_6$ over #, all with weight 0.
- There is a *heaven* state, which is accepting, and has a self loop over Σ with weight 0.
- The state q_1 is accepting and intuitively allows to accept short segments between consecutive #'s: It has a self transition over $\Sigma \setminus \{\#\}$ and a transition to *heaven* over #, all with weight -1.
- The state q_2 starts the command-checker gadget, which looks for a non-counting violation of $\mathcal{M}$'s commands (which is a simple regular check). Once reaching a violation it goes to *heaven*. All of its transitions are with weight 0. If it does not find a violation, it cannot continue the run.
- The state q_3 is a positive-cheat checker for $\mathcal{M}$'s counter x. It has a self loop over '`inc x`' with weight $+1$ and over '`dec x`' with weight -1. Over '`x=0 then goto`' it can nondeterministically choose between a self loop with weight 0 and a transition to q_0 with weight -1. Over the rest of the alphabet lettres, except for '`halt`' and #, it has a self loop with weight 0. (Over '`halt`' and # it cannot continue the run.)
- The state q_4 is a negative-cheat checker for $\mathcal{M}$'s counter x. It has a self loop over '`inc x`' with weight -1 and over '`dec x`' with weight $+1$. Over '`x>0 then goto`' it can nondeterministically choose between a self loop with weight 0 and a transition to q_0 with weight 0. Over the rest of the alphabet lettres, except for '`halt`' and #, it has a self loop with weight 0.
- The states q_5 and q_6 provide positive-cheat checker and negative-cheat checker for $\mathcal{M}$'s counter y, respectively, analogously to states q_3 and q_4.

▶ **Theorem 1.** *The initial-value universality problem for one-counter nets is undecidable.*

Proof. We show that a given two-counter machine $\mathcal{M}$ halts if and only if the corresponding one-counter net $\mathcal{A}$, as constructed in Section 3.1, is initial-value universal.

$\Rightarrow$: *When $\mathcal{M}$ halts*, its (legal) run has some length $n-1$. We claim that $\mathcal{A}$ is universal with the initial value n.

Consider some word w over the alphabet of $\mathcal{A}$. We shall describe an accepting run ρ of $\mathcal{A}$ on w. Until the first occurrence of #, the run ρ is deterministically in q_0, which is accepting. We show that for every segment between two consecutive #'s, as well as the segment after

the last #, the run ρ may either reach *heaven* or reach q_0 with counter value at least n (and remains there until the next # or the end of the word), from which it follows that ρ is accepting.

If the segment is shorter than n, q_0 can choose to go to q_1 over #, and from there it will reach heaven. If the segment is longer than n, it cannot describe the legal run of $\mathcal{M}$. Then, it must cheat within up to n steps. We show that each of the 5 possible cheats fulfills the claim.

1. If it makes a non-counting cheat, q_0 will go to q_2 over #, and will reach *heaven*. (This is also the case if it has additional letters different from # after the '`halt`' letter.)
2. If it makes a positive cheat on x, q_0 will go to q_3 upon reading the next #. When the cheat occurs, the value of x is positive, while reading the letter '`x=0 then goto`'. Notice that the value of $\mathcal{A}$'s counter is accordingly bigger than its value when entering q_3 (and by the inductive assumption bigger than n). Then, q_3 goes to q_0 with weight -1, guaranteeing that $\mathcal{A}$'s counter value is at least n. Notice that the counter value cannot go below n at any point, since $\mathcal{M}$ cannot make the value of x negative without a counting cheat. (We equipped $\mathcal{M}$ with a counter check before every decrement.)
3. If it makes a negative cheat on x, q_0 will go to q_4. Then, when the cheat occurs, the value of x is 0, while there is the letter '`x>0 then goto`'. Notice that the value of $\mathcal{A}$'s counter is accordingly exactly its value when entering q_3 (and by the inductive assumption at least n). Then, q_4 goes to q_0 with weight 0, guaranteeing that $\mathcal{A}$'s counter value is at least n. Notice that the counter might go below n between getting to q_4 and returning to q_0. Yet, since the violation must occur within up to n steps, and the value of the counter when entering q_4 is at least n, we are guaranteed to be able to properly continue with the run, as the counter need not go below 0.

4-5. Analogously, if it makes a positive or negative cheat over y, the choice of q_0 will be q_5 or q_6, respectively.

$\Leftarrow$: *When $\mathcal{M}$ does not halt*, for every positive integer n, we build the word w_n and show that it is not accepted by $\mathcal{A}$ with an initial counter value n.

The word w_n consists of $n+1$ segments between #'s, where each segment is the prefix of length $n+1$ of the (legal) run of $\mathcal{M}$. Consider the possible runs of $\mathcal{A}$ on w_n. It cannot go from q_0 to q_1, because it will stop after n steps. It also cannot go to q_2, because there is no cheating. We show that if it goes to $q_3..q_6$, it must return to q_0 before the next #, while decreasing the value of $\mathcal{A}$'s counter, which can be done only n times until the run stops.

If it goes to q_3, it must return to q_0 upon some '`x=0 then goto`', as it cannot continue the run on #. Yet, as there is no cheating, it returns to q_0 when $x = 0$, which implies that $\mathcal{A}$'s counter has the same value as when entering q_3, and due to the -1 weight of the transition to q_0, it returns to q_0 while decreasing the value of $\mathcal{A}$'s counter by 1. An analogous argument follows if it goes to q_5.

If it goes to q_4, it must return to q_0 upon some '`x>0 then goto`', as it cannot continue the run on #. Yet, as there is no cheating, it returns to q_0 while the value of x is indeed strictly positive, which implies that the value of $\mathcal{A}$'s counter is smaller than the value it had when entering q_4, and therefore due to the 0-weight transition to q_0, it returns to q_0 with a smaller value of $\mathcal{A}$'s counter. An analogous argument follows if it goes to q_6. ◀

3.2 Bounded Universality

We show that the problem is undecidable by making some changes to the undecidability proof of the initial-value universality problem.

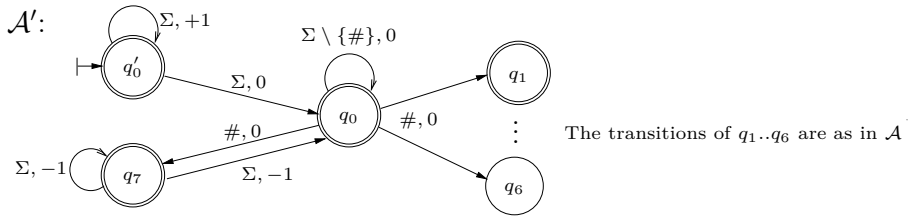


Figure 2 The one-counter net $\mathcal{A}'$ from the proof of Theorem 2.

Given a two-counter machine $\mathcal{M}$, we construct a one-counter net $\mathcal{A}'$ that is similar to $\mathcal{A}$, as constructed above, except for the following changes (see Figure 2):

- There is an additional state q_0' that is accepting, it is the new initial state, and it has a nondeterministic choice over Σ of either taking a self loop with weight $+1$ or going to q_0 with weight 0.
- The state q_0 is no longer initial, and it has an additional transition over $\#$ to a new state q_7 with weight 0.
- The state q_7 is accepting, and it has nondeterministic choice over Σ of either taking a self loop with weight -1 or going to q_0 with weight -1.

Now $\mathcal{M}$ halts if and only if $\mathcal{A}'$ is bounded universal for an initial counter value 0. We refer the reader to the full version [5] for a detailed proof.

▶ **Theorem 2.** *The bounded universality problem for one-counter nets is undecidable.*

4 Singleton Alphabet

In this section we study universality problems on OCN over singleton alphabets. The universality problem for NFA over singleton alphabets is already coNP-hard [23], a lower bound which trivially carries over to all problems considered here[1].

For simplicity, we identify languages $L \subseteq \{a\}^*$ with their Parikh image, so that the universality problems ask if the (bounded) language of a given OCN equals $\mathbb{N}$. Throughout this section, fix an OCN $\mathcal{A} = (\Sigma, Q, s_0, \delta, F)$.

We start by sketching our approach. Observe that the language of an OCN is not universal iff the OCN does not accept some word w. To show that such w exists, we distinguish between two cases: either w is "relatively short", in which case we use a guess-and-check approach to find it, or it is long, in which case we deduce its existence by analyzing some cyclic behaviour of the OCN. The details of both the guess-and-check elements and the cyclic behaviour depend on the encoding of the weights and the variant of universality.

4.1 Universality

We start by describing a procedure to decide the ordinary universality problem for OCN over singleton alphabets – with fixed initial configuration and no bounds on the counter.

Consider a cycle $\gamma = s_1, s_2, \ldots, s_k$ (with $s_1 = s_k$). Recall that $\mathit{effect}(\gamma)$ is the sum of weights along γ and $\mathit{depth}(\gamma)$ is the inverse of the lowest effect along the prefixes of γ. We call $1 \leq d \leq k$ a *nadir* of γ if it is the index of a prefix that attains the depth of γ. That is,

[1] The proof in [23, Theorem 6.1] in fact shows NP-completeness of the problem of whether two regular expressions over $\{0\}$ define different languages. Hardness is shown by reduction from Boolean satisfiability to non-universality of expressions using prime-cycles, and it is straightforward to rephrase it in terms of DFAs.

$\mathit{effect}(s_1,\ldots,s_d) = -\mathit{depth}(\gamma)$. We say that γ is *positive* if $\mathit{effect}(\gamma)$ is positive (and similarly for negative, non-negative, zero, etc.). We call γ *good* if it a simple, non-negative cycle, and $\mathit{depth}(\gamma) = 0$.

▶ **Observation 3.** *If γ is non-negative and it has a nadir d, then the* shifted cycle $\gamma^{\leftarrow d} \stackrel{def}{=} s_d\, s_{d+1}, \cdots, s_k, s_2, \cdots, s_d$ *is good. Similarly, if γ is negative, then $\mathit{effect}(\gamma^{\leftarrow d}) = -\mathit{depth}(\gamma^{\leftarrow d})$.*

For a state $r \in Q$ and an initial configuration s_0, c_0, let $\mathcal{L}^r(s_0,c_0) \subseteq \mathcal{L}(s_0,c_0)$ be the language of words accepted by a run that visits r.

The first tool we use in studying the universality problem is a canonical form for accepting runs, akin to *linear path schemes* of [18, 7].

▶ **Definition 4** (Linear Forms). *A path π is in* linear form *if there exist simple cycles $\gamma_1,\ldots,\gamma_k$ and paths $\tau_0,\ldots,\tau_k$ such that $\pi = \tau_0\gamma_1^{e_1}\tau_1\cdots\tau_{k-1}\gamma_k^{e_k}\tau_k$ for some numbers $e_1,\ldots,e_k \in \mathbb{N}$, and such that every non-negative cycle γ_i, is taken from a nadir, and so is executable with any counter value.*

We call e_i the exponent *of γ_i, and we refer to $\tau_0\gamma_1\tau_1\ldots\gamma_k\tau_k$ as the* underlying path *of π. The* length *of the linear form is the length of the underlying path.*

A linear form is described by the components above, where the exponents are given in binary. In the following, we show that every path can be transformed to a path in linear form with a small description size.

▶ **Lemma 5.** *Let π be an executable path of length n from (p,c) to (q,c'). Then there exists an executable path π' of length n in linear form whose length is at most $2|Q|^2$, from (p,c) to (q,c'') with $c'' \geq c'$.*

Proof Sketch: π' is obtained from π in two steps, namely rearranging simple cycles, and then choosing a small set of "representative" simple cycles to replace others. The crux of the proof is the first step, where instead of simply moving a cycle, we also shift it so that it is taken from its nadir. Then, for every set of simple cycles of the same length and on the same state, we take the one with maximal effect as a representative. ◀

We now turn to identify states that have a special significance in analyzing universality.

▶ **Definition 6.** *Let* Pump $\subseteq Q$ *be the set of states that admit good cycles. For each such state r fix a shortest good cycle γ_r.*

Intuitively, a state r is in Pump if it has a cycle that can be taken with any counter value, any number of times. That is, it can be used to "pump" the length of the word. Another important property is that if a path never visits a state in Pump then *all* its simple cycles must be negative. Indeed, any non-negative cycle must contain a non-negative simple cycle and any state at a nadir of such cycle must be in Pump.

If however, a state in Pump occurs along an accepting run, we can accept the same word using a run in a short linear form, as we now show.

▶ **Lemma 7.** *There exists a bound $B_1 \in \mathsf{poly}(|Q|, \|\delta\|)$ such that, for every $n \in \mathbb{N}$, if n is accepted by a run that visits a state $r \in$ Pump, then n has an accepting run of the form $\eta_1\gamma_r^t\eta_2$ for paths η_1,η_2 of length at most B_1.*

Proof Sketch. Using Lemma 5, we split an accepting run on n that visits r to the form π_1, r, π_2 where π_1 and π_2 are in linear form. Then, we successively shorten π_1 and π_2 by eliminating simple cycles along them, and instead pumping the non-negative cycle γ_r. Some careful accounting is needed so that the length of the path is maintained, and so that it remains executable. ◀

We now characterize the regular language $\mathcal{L}^r(s_0, c_0)$ using a DFA of bounded size.

▶ **Lemma 8.** *There exists a bound $B_2 \in \mathsf{poly}(\|\delta\| \cdot |Q|)$ such that, for every $r \in \mathrm{Pump}$, there exists a DFA that accepts $\mathcal{L}^r(s_0, c_0)$ and is of size at most B_2.*

Define $\mathcal{P} \stackrel{def}{=} \bigcup_{r \in \mathrm{Pump}} \mathcal{L}^r(s_0, c_0)$. Notice that $\mathcal{P} \subseteq \mathcal{L}(s_0, c_0)$ and that $\mathcal{L}(s_0, c_0) \setminus \mathcal{P}$ must be finite. Indeed, if $w \in \mathcal{L}(s_0, c_0) \setminus \mathcal{P}$ then it can only be accepted by runs with *only* negative cycles, of which there are finitely many. In particular, if $\mathbb{N} \setminus \mathcal{P}$ is infinite, then $\mathcal{L}(s_0, c_0) \neq \mathbb{N}$.

Using the bounds from Lemma 8, we have the following.

▶ **Lemma 9.** *There exists $B_3 \in \mathsf{poly}(\|\delta\|, |Q|)$ such that $\mathcal{L}(s_0, c_0) \neq \mathbb{N}$ if, and only if, there exists $n \in \mathbb{N}$ such that either $n < B_2$ and $n \notin \mathcal{L}(s_0, c_0)$, or $B_3^{|Q|} \leq n \leq 2B_3^{|Q|}$ and $n \notin \mathcal{P}$.*

Lemma 9 suggests the following algorithmic scheme for deciding non-universality: non-deterministically either (1) guess $n < B_3$, and check that $n \notin \mathcal{L}(s_0, c_0)$, or (2) guess $B_3^{|Q|} \leq n \leq 2B_3^{|Q|}$ and check that $n \notin \mathcal{L}^r(s_0, c_0)$ for all $r \in \mathrm{Pump}$, which implies that $n \notin \mathcal{P}$.

Note that even if the transitions are encoded in unary, n still needs to be guessed in binary for part (2) (and also for part (1) if the encoding is binary). The complexity of the checks involved in both parts of the algorithm depend on the encoding of the transitions, and are handled separately in the following.

Unary Encoding. If the transitions are encoded in unary, then B_3 is polynomial in the size of the OCN. Consequently, we can check for $n < B_3$ whether $n \in \mathcal{L}(s_0, c_0)$ by simulating the OCN for n steps, while keeping track of the maximal run to each state. Indeed, due to the monotonicity of executability of OCN paths it suffices to remember, for each state s, the maximal possible counter-value c so that (s, c) is reachable via the current prefix, which must be a number $\leq c_0 + n \cdot \|\delta\|$ or $-\infty$ (to represent that no configuration (s, c) can be reached).

Next, in order to check whether $n \notin \mathcal{L}^r(s_0, c_0)$ for all $r \in \mathrm{Pump}$ for $B_3^{|Q|} \leq n \leq 2B_3^{|Q|}$ written in binary, we notice that since B_3 is polynomial in the description of the OCN, then the size of each DFA for $\mathcal{L}^r(s_0, c_0)$ constructed as per Lemma 8 is polynomial in the OCN. Since the proof in Lemma 8 is constructive, we can obtain an explicit representation of these DFAs. Finally, given a DFA (or indeed, and NFA) over a singleton alphabet and n written in binary, we can check whether n is accepted in time $O(\log n)$ by repeated squaring of the transition matrix for the DFA [23]. We conclude with the following.

▶ **Theorem 10.** *The universality problem for singleton-alphabet one-counter nets with transitions encoded in unary is in* coNP*, and is thus* coNP*-complete.*

Binary Encoding. When the transitions are encoded in binary, B_3 is potentially exponential in the encoding of the OCN. Thus, naively adapting the methods taken in the unary case (with basic optimization) will lead to a PSPACE algorithm for universality (using Savitch's Theorem). As we now show, by taking a different approach, we can obtain an upper bound of $\mathsf{coNP}^{\mathsf{NP}}$, placing the problem in the second level of the polynomial hierarchy.

In order to obtain this bound, we essentially show that given n encoded in binary, checking whether n is accepted by the OCN can be done in NP. This is based on the linear form of Lemma 5.

▶ **Lemma 11.** *Let $\pi = \tau_0 \gamma_1^{e_1} \tau_1 \cdots \tau_{k-1} \gamma_k^{e_k} \tau_k$ be a run in linear form, then we can check whether π is executable from counter value c in time polynomial in the description of π.*

Lemma 11 shows that, given n in binary, we can check whether $n \in \mathcal{L}(s_0, c_0)$ in NP. Indeed, we guess the structure of an accepting run in linear form (including the exponents of the cycles), and check in polynomial time whether this run is executable, and whether it is accepting.

In order to complete our algorithmic scheme for universality, it remains to show how we can check in NP, given n in binary, whether $n \notin \mathcal{L}^r(s_0, c_0)$ for every r. In contrast to the case of unary encoding, this is fairly simple.

Given r, we can construct an OCN $\mathcal{A}^r$ such that $\mathcal{L}_{\mathcal{A}^r}(s_0, c_0) = \mathcal{L}^r_{\mathcal{A}}(s_0, c_0)$ by taking two copies of $\mathcal{A}$, and allowing a transition to the second copy only once r is reached. The accepting states are then those of the second copy. Thus, checking whether $n \notin \mathcal{L}^r(s_0, c_0)$ amounts to checking whether $n \notin \mathcal{L}_{\mathcal{A}^r}(s_0, c_0)$. We can now complete the algorithmic scheme.

▶ **Theorem 12.** *The universality problem for singleton-alphabet one-counter nets with transitions encoded in binary is in* $\mathsf{coNP}^{\mathsf{NP}}$.

4.2 Initial-Value Universality

The characterization of universality given in Lemma 9 can be simplified in the case of initial-value universality, in the sense that the freedom in choosing an initial value allows us to work with the underlying automaton of the OCN, disregarding the transition effects. This also allows us to obtain the same complexity results under unary and binary encodings.

Recall that Pump is the set of states that admit good cycles (see Definition 6). Let $\mathcal{N}$ be the underlying NFA of $\mathcal{A}$. For a state $r \in$ Pump, define $\mathcal{L}^r_{\mathcal{N}}(s_0)$ to be the set of words accepted by $\mathcal{N}$ via a run that visits r. Overloading the notation of Section 4.1, we define $\mathcal{P} \stackrel{def}{=} \bigcup_{r \in \text{Pump}} \mathcal{L}^r_{\mathcal{N}}(s_0)$.

▶ **Lemma 13.** *There exists* c_0 *such that* $\mathcal{L}_{\mathcal{A}}(s_0, c_0) = \mathbb{N}$ *iff* $\mathcal{L}_{\mathcal{N}}(s_0) = \mathbb{N}$ *and* $\mathbb{N} \setminus \mathcal{P}$ *is finite.*

Following similar arguments to those in Lemmas 7 and 8, and using the fact that we work with the underlying NFA, we can show the following.

▶ **Lemma 14.** *There exists a bound* $B_4 \in \mathsf{poly}(|Q|)$ *such that, for every* $r \in$ Pump *there exists a DFA that accepts* $\mathcal{L}^r(s_0)$ *and which is of size at most* B_4.

We can now solve the initial-value universality problem.

▶ **Theorem 15.** *The initial-value universality problem for one-counter nets (in unary or binary encoding) is* coNP*-complete.*

Proof. First, observe that the problem is coNP-hard by reduction from the universality problem for NFAs. We now turn to show the upper bound.

By Lemma 13, it is enough to decide whether $\mathcal{L}_{\mathcal{N}}(s_0) = \mathbb{N}$ and $\mathbb{N} \setminus \mathcal{P}$ is finite. Checking whether $\mathcal{L}_{\mathcal{N}}(s_0) = \mathbb{N}$, i.e., deciding the universality problem for NFA over a single-letter alphabet, can be done in coNP [23].

By Lemma 14, there exists a DFA $\mathcal{D}$ for $\mathbb{N} \setminus \mathcal{P}$ of size at most $M = B_4^{|Q|}$, by taking the intersection of the respective DFAs over every $r \in$ Pump. Thus, $\mathbb{N} \setminus \mathcal{P}$ is infinite iff $\mathcal{D}$ accepts a word of length $M < n \leq 2M$ (as such a word induces infinitely many other words). Thus, we can decide in NP whether $\mathbb{N} \setminus \mathcal{P}$ is infinite, by guessing $M < n \leq 2M$, and checking that it is in $\mathcal{L}^r(s_0)$ for every $r \in$ Pump (using repeated squaring on the respective DFAs).

We conclude that both checking whether $\mathcal{L}_{\mathcal{N}}(s_0) = \mathbb{N}$ and whether $\mathbb{N} \setminus \mathcal{P}$ is finite can be done in coNP, and so the initial value universality problem is also in coNP. ◀

4.3 Bounded Universality

For bounded universality, the states in Pump are not restrictive enough: in order to keep the counter bounded, a state must admit a 0-effect cycle. However, these cycles need not be simple. Thus, we need to adjust our definitions somewhat. Fortunately, however, once the correct definitions are in place, most of the proofs carry out similarly to those of Section 4.1.

▶ **Definition 16.** *A state $q \in Q$ is* stable *if either:*
1. *it is at the nadir of a simple positive cycle, and admits a negative cycle, or*
2. *it is at the nadir of a simple zero cycle.*

We denote by Stable *the set of stable states.*

Identifying stable states can be done in polynomial time (see e.g. Lemma 24). The motivation behind this definition is to identify states that admit a zero-effect (not necessarily simple) cycle.

▶ **Lemma 17.** *There exists a bound $B_5 \in \mathsf{poly}(|Q|, \|\delta\|)$ such that, every stable state q admits a zero cycle of length and depth at most B_5.*

By Lemma 17 we can fix, for each $q \in$ Stable, some zero-cycle ζ_q with effect and depth bounded by B_5. Recall that $\mathcal{L}^r(s_0, c_0)$ is the set of words that are accepted with a path that passes through r. Let $\mathcal{S} \stackrel{def}{=} \bigcup_{r \in \text{Stable}} \mathcal{L}^r(s_0, c_0)$. We prove an analogue of Lemma 7.

▶ **Lemma 18.** *There exists a bound $B_6 \in \mathsf{poly}(|Q|, \|\delta\|)$ such that every $n \in \mathcal{L}^r(s_0, c_0)$ has an accepting run of the form $\eta_1 \zeta_r^t \eta_2$ for paths η_1, η_2 of length at most B_6.*

Proof. The proof follows *mutatis-mutandis* that of Lemma 7, with one important difference: before replacing cycles with iterations of the zero cycle ζ_r, we replace a bounded number of cycles with the positive cycle on r, on which r is at a nadir,[2] so that the counter value goes above $depth(\zeta_r)$, enabling us to take ζ_r arbitrarily many times. Note that this lengthens the prefix η_1 at most polynomially in $(|Q| \cdot \|\delta\|)$. ◀

Lemma 18 implies that every word $n \in \mathcal{S}$ can be accepted by a run whose counter values are bounded because there must by an accepting run that, except for some bounded prefix and suffix, only iterates some zero-cycle ζ_r. More precisely, we have the following.

▶ **Theorem 19.** *There exists $B_6 \in \mathsf{poly}(|Q|, \|\delta\|)$ such that every word $n \in \mathcal{S}$ is accepted by a run whose counter value remains below $2B_6 + c_0$.*

In addition, Lemma 18 immediately gives us (with an identical proof) an analogue of Lemma 8.

▶ **Lemma 20.** *There exists a bound $B_7 \in \mathsf{poly}(|Q|, \|\delta\|)$ such that, for every $r \in$ Stable there exists a DFA that accepts $\mathcal{L}^r(s_0, c_0)$ and is of size at most B_7.*

We can now characterize bounded universality in terms of $\mathcal{S}$, the set of stable states.

▶ **Lemma 21.** *$\mathcal{L}(s_0, c_0)$ is bounded-universal if, and only if, the underlying automaton $\mathcal{N}$ is universal ($\mathcal{L}_{\mathcal{N}}(s_0) = \mathbb{N}$) and $\mathbb{N} \setminus \mathcal{S}$ is finite.*

Finally, checking whether $\mathbb{N} \setminus \mathcal{S}$ is finite can be done similarly to Section 4.1 (and the complexity depends on the transition encoding), by checking that a candidate word n of bounded length is not in $\mathcal{L}^r(s_0, c_0)$ for all stable states r. We conclude with the following.

▶ **Theorem 22.** *Bounded universality of one-counter nets is* coNP*-complete assuming unary encoding, and in* $\mathsf{coNP}^{\mathsf{NP}}$ *assuming binary encoding.*

[2] That is, unless r is the nadir of a zero cycle, in which case the proof requires no changes.

5 Deterministic Systems

We turn to deterministic one-counter nets (DOCNs) for which the underlying finite automaton is a DFA. We assume without loss of generality that the graphs underlying the DOCNs are connected, i.e., that all states are reachable from the initial state.

For such systems, (bounded) universality problems can be decided by checking a suitable combination of simple conditions on cycles and short words. Lemma 24 lists these conditions and upper complexity bounds for checking them. We then show which combination allows to solve each decision problem (Lemma 25). All mentioned upper bounds follow either easily from first principles, or from the result that the state reachability problem (a.k.a., coverability) for OCN is in NC [6, Theorem 15]. We will also use the following fact which follows from [25].

▶ **Lemma 23.** *Given a set $S = \{\alpha_1, \alpha_2 \dots \alpha_n\}$ of integers written in binary, the question whether the sum of all elements in S is non-negative is in NC^2.*

▶ **Lemma 24** (Basic Conditions)**.** *Consider the following conditions on a deterministic one-counter net $\mathcal{A} = (\Sigma, Q, s_0, \delta, F)$, initial value $c_0 \in \mathbb{N}$, and bound $b \in \mathbb{N}$.*
(C1) *The underlying automaton is universal.*
(C2) *Every word w of length $|w| \leq |Q|$ is in $\mathcal{L}(s_0, c_0)$*
(C3) *Every word w of length $|w| \leq |Q|$ is in $\mathcal{L}^{\leq b}(s_0, c_0)$*
(C4) *All simple cycles have non-negative effect.*
(C5) *All simple cycles have 0-effect.*
Condition (C1) can be checked in non-deterministic logspace (NL), independently of the encoding of numbers. All other conditions can be verified in NL assuming unary encoding, and in NC (conditions (C4) and (C5) even in NC^2) assuming binary encoding.

▶ **Lemma 25.** *Consider a deterministic one-counter net with initial state s_0.*
1. *For any $c_0 \in \mathbb{N}$, the language $\mathcal{L}(s_0, c_0)$ is universal if, and only if, all simple cycles are non-negative (C4), and all words shorter than the number of states are accepting (C2).*
2. *There exists an initial counter value $c_0 \in \mathbb{N}$ such that $\mathcal{L}(s_0, c_0)$ is universal if, and only if, all simple cycles are non-negative (C4), and the underlying automaton is universal (C1).*
3. *For any $c_0 \in \mathbb{N}$, there exists a bound $b \in \mathbb{N}$ such that the bounded language $\mathcal{L}^{\leq b}(s_0, c_0)$ is universal if, and only if, (C5) the effect of all simple cycles is 0 and (C3) all words shorter than the number of states are in $\mathcal{L}^{\leq b'}(s_0, c_0)$ for $b' \stackrel{def}{=} |Q| \cdot \|\delta\|$.*

The following is a direct consequence of Lemmas 24 and 25.

▶ **Theorem 26.** *The universality, initial-value universality, and bounded universality problems for deterministic one-counter nets are in NL assuming unary encoding, and in NC assuming binary encoding.*

For the special case of DOCN over single letter alphabets, it is possible to derive even better upper bounds, based on the particular shape of the underlying automaton.

Recall that a deterministic automaton over a singleton alphabet is in the shape of a lasso: it consists of an acyclic path that ends in a cycle.

▶ **Lemma 27.** *For any given deterministic one-counter net $\mathcal{A} = (\Sigma, Q, s_0, \delta, F)$ with $|\Sigma| = 1$ and $c_0, b \in \mathbb{N}$, one can verify in deterministic logspace (L) that (C1) the underlying DFA is universal. Moreover, conditions (C2), (C3), (C4), and (C5) as defined in Lemma 24 can be verified in L assuming unary encodings and in NC^2 assuming binary encodings.*

Using Lemma 27 and the characterisation of the three universality problems by Lemma 25, we get the desired complexity upper bounds.

▶ **Theorem 28.** *The universality, initial-value universality, and bounded universality problems of deterministic one-counter nets over a singleton alphabet are in* L *assuming unary encoding and in* NC^2 *assuming binary encoding.*

6 Unambiguous Systems

In line with the usual definition of unambiguous finite automata, we call an OCN with a given initial configuration *unambiguous* iff for every word in its language there exists exactly one accepting run. Since the language of an OCN depends in a monotone fashion on the initial counter value, there is also a related, but different, notion of unambiguity. We call an OCN (which has a fixed initial state s_0) *structurally unambiguous* if the unambiguity condition holds for every initial counter c_0. Notice that every OCN that has an unambiguous underlying automaton is necessarily structurally unambiguous. We will show (Lemma 32) that these conditions are in fact equivalent.

In [11], the complexity of the universality problem for unambiguous vector addition systems with states (VASSs) was studied. In particular, for unambiguous OCNs, it is shown that checking universality is in NC^2 and NL-hard, assuming unary encoded inputs, and in PSPACE and coNP-hard, assuming binary encoding. The special case of unambiguous OCN over a single letter alphabet is not considered there, nor are the initial-counter – and bounded universality problems. We discuss these problems in the remainder of this section.

We assume w.l.o.g, that for any given OCN, all states in the underlying automaton are reachable from the initial state, and that from every state it is possible to reach an accepting state. States that do not satisfy these properties can be removed in NL. Moreover, all algorithms we propose need to check universality for the underlying automaton, and hence rely on the following computability result (see [26] for a proof for general alphabet, and the full paper for singleton alphabet).

▶ **Lemma 29.** *Universality of an unambiguous finite automaton over single letter alphabet is in* NL*, and over general alphabet is in* NC^2*.*

We will start by considering the universality problem for unambiguous OCNs over a single letter alphabet. Here, unambiguity implies a strong restriction on accepting runs: if a run is accepting then it contains at most one positive cycle (which may be iterated multiple times).

▶ **Lemma 30.** *Let* $\pi = \pi_1\pi_2\pi_3$ *be an accepting run where* π_2 *is a positive simple cycle. Then* $\pi_3 = \pi_2^k\pi_4$ *for some* $k \in \mathbb{N}$ *and acyclic path* π_4*.*

Proof. Assume towards contradiction that there is an accepting run $\pi = \pi_1\pi_2\pi_3\pi_4\pi_5$, where π_2 is a positive simple cycle and π_4 is a simple cycle. Based on this we show that the system cannot be unambiguous. Let $c = |Q| \cdot \|\delta\|$ and denote by $|\pi|$ the length of path π.

Since π_2 has a positive effect, it follows that $\pi' = \pi_1\pi_2^{|\pi_4|+c\cdot|\pi_2|}\pi_3\pi_4\pi_5$ is an accepting run. But there is a second run that reads the same word, namely $\pi'' = \pi_1\pi_2^{c\cdot|\pi_2|}\pi_3\pi_4^{|\pi_2|}\pi_5$. The second run is indeed a run as the increment along $\pi_2^{c\cdot|\pi_2|}$ is bigger than any possible negative effect of $\pi_4^{|\pi_2|}$. Moreover the lengths of both runs are the same as $\pi_2^{|\pi_4|} = \pi_4^{|\pi_2|}$. ◀

A consequence of Lemma 30 is that if along any accepting run the value of the counter exceeds $B_0 = |Q| \cdot \|\delta\|$ then it cannot drop to zero afterwards, as it would require at least one negative cycle to do so. One can therefore encode all counter values up to B_0 into the finite-state control and solve universality for the resulting UFA. Lemma 29 thus yields the following.

▶ **Theorem 31.** *The universality problem of unary encoded unambiguous one-counter nets over a singleton alphabet is in* NL.

We consider next the initial-value universality problem for unambiguous OCNs. Since whether an OCN is unambiguous depends on the initial counter value, the initial-value universality problem is only meaningful for structurally unambiguous systems, those which are unambiguous regardless of the initial counter. We first observe a simple fact about these definitions.

▶ **Lemma 32.** *An OCN is structurally unambiguous if and only if its underlying automaton is unambiguous.*

▶ **Lemma 33.** *Consider a structurally unambiguous OCN with initial state s_0. There exists an initial counter c_0 so that $\mathcal{L}(s_0, c_0) = \Sigma^*$ if, and only if, the underlying automaton is universal and has no negative cycles.*

The following is a direct consequence of Lemma 33 and the complexity bounds provided by Lemmas 24 and 29, for the cycle condition *(C4)*.

▶ **Theorem 34.** *The initial-value universality problem of structurally unambiguous one-counter nets is in* NC^2 *assuming binary encoding, and in* NL *assuming unary encoding and single-letter alphabets.*

Finally, we turn our attention to the bounded universality problem for unambiguous OCNs. This turns out to be quite easy, due to the following observation.

▶ **Lemma 35.** *If an unambiguous OCN is bounded universal then no accepting run contains a positive cycle.*

▶ **Theorem 36.** *The bounded universality problem of unambiguous one-counter nets with unary-encoded transition weights is in* NC^2*, and in* NL *if the alphabet has only one letter, and for binary-encoded transition weights it is in* PSPACE.

References

1. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Piotr Hofman, Richard Mayr, K. Narayan Kumar, and Patrick Totzke. Infinite-state energy games. In *ACM/IEEE Symposium on Logic in Computer Science (LICS)*. ACM, 2014. doi:10.1145/2603088.2603100.
2. Parosh Aziz Abdulla, Mohamed Faouzi Atig, Richard Mayr Radu Ciobanu, and Patrick Totzke. Universal safety for timed Petri nets is PSPACE-complete. In *International Conference on Concurrency Theory (CONCUR)*, 2018. doi:10.4230/LIPIcs.CONCUR.2018.6.
3. Parosh Aziz Abdulla, Pavel Krcal, and Wang Yi. R-automata. In *International Conference on Concurrency Theory (CONCUR)*, 2008.
4. S. Almagor, U. Boker, and O. Kupferman. What's decidable about weighted automata? In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2011.
5. Shaull Almagor, Udi Boker, Piotr Hofman, and Patrick Totzke. Parametrized universality problems for one-counter nets. *CoRR*, 2020. arXiv:2005.03435.
6. Shaull Almagor, Nathann Cohen, Guillermo A. Pérez, Mahsa Shirmohammadi, and James Worrell. Coverability in 1-vass with disequality tests. In *International Conference on Concurrency Theory (CONCUR)*, 2020.

7 Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in two-dimensional vector addition systems with states is PSPACE-complete. In *ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.14.

8 Stanislav Böhm, Stefan Göller, Simon Halfon, and Piotr Hofman. On Büchi One-Counter Automata. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.14.

9 Stanislav Böhm, Stefan Göller, and Petr Jančar. Bisimilarity of one-counter processes is pspace-complete. In *International Conference on Concurrency Theory (CONCUR)*, 2010.

10 Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, Nicolas Markey, and Jiří Srba. Infinite runs in weighted timed automata with energy constraints. In *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, 2008. doi:10.1007/978-3-540-85778-5_4.

11 Wojciech Czerwiński, Diego Figueira, and Piotr Hofman. Universality Problem for Unambiguous VASS. In *International Conference on Concurrency Theory (CONCUR)*, 2020.

12 A. Degorre, L. Doyen, R. Gentilini, J.F. Raskin, and S. Torunczyk. Energy and mean-payoff games with imperfect information. In *Computer Science Logic (CSL)*, 2010.

13 Stéphane Demri. On selective unboundedness of VASS. *Journal of Computer and System Sciences*, 2013.

14 Piotr Hofman, Slawomir Lasota, Richard Mayr, and Patrick Totzke. Simulation problems over one-counter nets. *Logical Methods in Computer Science*, 2016. doi:10.2168/LMCS-12(1:6)2016.

15 Piotr Hofman and Patrick Totzke. Trace inclusion for one-counter nets revisited. *Theoretical Computer Science*, 2017. doi:10.1016/j.tcs.2017.05.009.

16 Petr Jancar, Javier Esparza, and Faron Moller. Petri Nets and Regular Processes. *Journal of Computer and System Sciences*, 1999.

17 S. Rao Kosaraju and Gregory F. Sullivan. Detecting cycles in dynamic graphs in polynomial time (preliminary version). In *Symposium on Theory of Computing (STOC)*. ACM, 1988.

18 Jérôme Leroux and Grégoire Sutre. On flatness for 2-dimensional vector addition systems with states. In *International Conference on Concurrency Theory (CONCUR)*. Springer Berlin Heidelberg, 2004.

19 Richard Mayr. Undecidability of weak bisimulation equivalence for 1-counter processes. In *International Colloquium on Automata, Languages and Programming (ICALP)*. Springer, 2003.

20 M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1 edition, 1967.

21 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 1978. doi:10.1016/0304-3975(78)90036-1.

22 Louis E. Rosier and Hsu-Chun Yen. A multiparameter analysis of the boundedness problem for vector addition systems. In *International Symposium on Fundamentals of Computation Theory (FCT)*. Springer Berlin Heidelberg, 1985.

23 L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time(preliminary report). In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*. ACM, 1973. doi:10.1145/800125.804029.

24 Leslie G. Valiant. *Decision Procedures for Families of Deterministic Pushdown Automata*. PhD thesis, University of Warwick, 1973. URL: http://wrap.warwick.ac.uk/34701/.

25 Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag, 1999.

26 Tzeng Wen-Guey. On path equivalence of nondeterministic finite automata. *Information Processing Letters*, 1996. doi:10.1016/0020-0190(96)00039-7.

Reachability in Fixed Dimension Vector Addition Systems with States

Wojciech Czerwiński
University of Warsaw, Poland
wczerwin@mimuw.edu.pl

Sławomir Lasota
University of Warsaw, Poland
sl@mimuw.edu.pl

Ranko Lazić
University of Warwick, Coventry, UK
R.S.Lazic@warwick.ac.uk

Jérôme Leroux
CNRS & University of Bordeaux, France
jerome.leroux@labri.fr

Filip Mazowiecki
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany
filipm@mpi-sws.org

Abstract

The reachability problem is a central decision problem in verification of vector addition systems with states (VASS). In spite of recent progress, the complexity of the reachability problem remains unsettled, and it is closely related to the lengths of shortest VASS runs that witness reachability.

We obtain three main results for VASS of fixed dimension. For the first two, we assume that the integers in the input are given in unary, and that the control graph of the given VASS is flat (i.e., without nested cycles). We obtain a family of VASS in dimension 3 whose shortest runs are exponential, and we show that the reachability problem is NP-hard in dimension 7. These results resolve negatively questions that had been posed by the works of Blondin et al. in LICS 2015 and Englert et al. in LICS 2016, and contribute a first construction that distinguishes 3-dimensional flat VASS from 2-dimensional ones. Our third result, by means of a novel family of products of integer fractions, shows that 4-dimensional VASS can have doubly exponentially long shortest runs. The smallest dimension for which this was previously known is 14.

2012 ACM Subject Classification Theory of computation → Concurrency; Theory of computation → Verification by model checking; Theory of computation → Logic and verification

Keywords and phrases reachability problem, vector addition systems, Petri nets

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.48

Related Version `https://arxiv.org/abs/2001.04327`

Funding *Wojciech Czerwiński*: Supported by the ERC grant LIPA, agreement no. 683080.
Sławomir Lasota: Supported by the NCN grant 2017/27/B/ST6/02093.
Ranko Lazić: Supported by EPSRC grant EP/P020992/1.
Jérôme Leroux: Supported by ANR grant ANR-17-CE40-0028.

Acknowledgements We thank Matthias Englert for inspiring conversations.

1 Introduction

Context

Vector addition systems with states (shortly, VASS) [20, cf. Section 5.1], [24], vector addition systems without states (shortly, VAS) [27], and Petri nets [37], are equally expressive with well-known straightforward mutual translations. They form a long established model of concurrency with extensive applications in modelling and analysis of hardware [7, 28], software [19, 6, 25] and database [4, 5] systems, as well as chemical [1], biological [36, 2] and business [43, 32] processes (where the references are illustrative).

Two central decision problems in the context of formal verification based on that model are the following. Stated in terms of the first formalism, the input of both problems is a VASS $\mathcal{V}$, and two configurations $p(\mathbf{v})$ and $q(\mathbf{w})$.

Coverability asks whether $\mathcal{V}$ has a run starting at $p(\mathbf{v})$ and finishing at some configuration $q(\mathbf{w}')$ such that $\mathbf{w}' \geq \mathbf{w}$. Thus the final configuration of the run needs to have control that is in the given target state q and resources that are component-wise no smaller than the given target vector $\mathbf{w}$. In applications, $q(\mathbf{w})$ is typically seen as a minimal unsafe configuration, and the coverability problem is fundamental for verifying safety properties.

Reachability asks whether $\mathcal{V}$ has a run starting at $p(\mathbf{v})$ and finishing at $q(\mathbf{w})$. Thus the run needs to reach the given target configuration exactly. It has turned out that verification of liveness properties amounts to solving the reachability problem [22]. Moreover, a plethora of problems from formal languages [10], logic [26, 13, 12, 8], concurrent systems [18, 16], process calculi [35], linear algebra [23] and other areas (the references are again illustrative, cf. Schmitz's recent survey [40]) are inter-reducible with the reachability problem.

The coverability problem was found ExpSpace-complete already in the 1970s [33, 38], and the reachability problem was proved decidable in the early 1980s [34]. However, the complexity of the latter has become one of the most studied open questions in the theory of verification. The best upper and lower bounds are both very recent, and are given by an Ackermannian function [29] and a tower of exponentials [11], respectively.

Fixed Dimension VASS

The gaps in the state of the art on the complexity of the reachability problem are particularly vivid when the dimension is fixed. For concreteness, we focus on VASS, bearing in mind that corresponding statements in terms of VAS or Petri nets can be obtained by means of standard translations (we refer to [40, Section 2.1] for details, noting that in some cases the dimension is affected by a small additive constant). The only broadly settled cases are for dimensions 1 and 2, as shown in the following table, where "unary" and "binary" specify how the integers in the input to the reachability problem are encoded.

	unary VASS	binary VASS
dimension 1	NL-complete [42]	NP-complete [21]
dimension 2	NL-complete [14]	PSpace-complete [3]

For dimensions $d \geq 3$, the best known bounds are from [29] and [11], namely membership of the fast-growing primitive recursive class $\mathbf{F}_{d+4}$ and hardness for $(d-13)$-ExpSpace when $d \geq 14$, respectively, which hold with both unary and binary encodings. In particular, for $3 \leq d < 14$, no better lower bounds have been known than NL for unary VASS and PSpace for binary VASS, whereas the $\mathbf{F}_{d+4}$ upper bound is far above elementary already for $d = 3$.

Flat Control

The structural restriction of flatness, which is essentially that the control graph contains no nested cycles, has long played a prominent role in a number of settings in verification, cf. e.g. [9]. In fact, all the tight upper bounds for dimensions 1 and 2 recalled above can be seen as due to the effective flattability of 2-dimensional VASS [30]. Regarding the complexity of reachability for flat VASS, there has been a marked contrast in the state of the art depending on the encoding.

Binary: Thanks to reducibility to existential Presburger arithmetic [17, 3], we have NP membership, even when the dimension is not fixed. And already for dimension 1, we have NP hardness.

Unary: With the exception of dimensions 1 and 2 for which we have the NL memberships, no better upper bound than NP has been known in dimension 3 or higher. And for any fixed dimension, no better lower bound than NL has been obtained.

Interestingly, from the results of Rosier and Yen [39], we have that the coverability problem for fixed dimension flat VASS is in NP with the binary encoding and in NL with the unary encoding, which not provably better than the reachability problem as just discussed.

Main Results

The NL memberships of reachability for unary VASS in dimension 2 and of coverability for unary VASS in any fixed dimension were obtained by proving that polynomially bounded witnessing runs always exist. It is therefore pertinent to ask:

> *Do polynomially bounded witnessing runs exist for reachability for unary flat VASS in fixed dimensions greater than* 2*?*

Our first main result, presented in Section 3, provides a negative answer immediately in dimension 3. We believe this is very significant for the continuing quest to understand the reachability problem, for which as we have seen there is currently a huge complexity gap already in dimension 3. Namely, 3-dimensional VASS have so far been distinguished from 2-dimensional VASS only by means of the infamous example of Hopcroft and Pansiot [24, proof of Lemma 2.8], which shows that, in contrast to the latter, the former do not have semi-linear reachability sets and are hence not flattable. However, we now have a new distinguishing feature which is present even under the restriction of flatness.

Even if polynomially bounded witnessing runs do not exist, it is conceivable that the decision problem nevertheless has low complexity, so we next ask:

> *Is reachability for unary flat VASS in NL in fixed dimensions greater than* 2*?*

We show that this is unlikely in Section 4, where our second main result establishes NP hardness in dimension 7. This provides the first concrete indication that the reachability problem is harder than the coverability problem for fixed dimension flat VASS.

Lastly, we turn to binary VASS in fixed dimensions d, where without the flat assumption, the enormous complexity gap between PSpace hardness and $\mathbf{F}_{d+4}$ membership remains for $3 \leq d \leq 13$. Given that exponentially bounded witnessing runs exist for $d = 2$ [14] (which yields PSpace membership) but not for $d = 14$ [11], we ask:

> *Do exponentially bounded witnessing runs exist for reachability for binary VASS in fixed dimensions from* 3 *to* 13*?*

A negative answer is provided in Section 5 by our third main result, which exhibits a family of 4-dimensional VASSes whose shortest witnessing runs are doubly exponentially long.

Technical Contributions

In all three of the main results, we make use of a key technical pattern first seen in [11], namely checking divisibility of a counter x by a large integer as follows: ensure that a counter y is initially equal to x, then multiply x weakly (which a priori may nondeterministically produce an erroneous smaller result) by many integer fractions greater 1 whose product is c/d, and finally verify that $\mathsf{x} = \mathsf{y} \cdot (c/d)$ by subtracting c from x and d from y repeatedly until they are both 0. The divisibility by the large integer is ensured because the check succeeds if and only if the weak multiplications are all exact. However, much additional development has been involved:

1. For the exponentially long shortest runs in Section 3, we employ the factorial fractions also seen in [11], but in reverse order, with the construction stripped to its essentials to minimise the dimension, and with a detailed divisibility analysis of large integers.
2. The NP hardness in Section 4 builds on the development in the previous section, adding careful machinery that facilitates exact computations on exponentially large integers.
3. To obtain the doubly exponentially long shortest runs in Section 5, we have developed an intricate new family of sequences of fractions, where in contrast to the much simpler factorial equations, the number of distinct fractions in a sequence is logarithmic in relation to both the numerators and the denominators as well as to the length of the sequence.

2 Preliminaries

Vector Addition Systems with States

A *vector addition system with states* in dimension d (d-VASS, or simply VASS if the dimension is irrelevant) is a pair $\mathcal{V} = (Q, T)$ consisting of a finite set Q of states and a finite set of transitions $T \subseteq Q \times \mathbb{Z}^d \times Q$. The size of a VASS is $|Q| + |T| \cdot s$, where s is the maximum on the representation size of a vector in T. A *configuration* of a d-VASS is a pair $(p, \mathbf{v}) \in Q \times \mathbb{N}^d$, denoted $p(\mathbf{v})$, consisting of a state p and a nonnegative integer vector $\mathbf{v}$. A run of a d-VASS is a sequence of configurations

$$p_0(\mathbf{v}_0),\ \ldots,\ p_k(\mathbf{v}_k), \tag{1}$$

such that for every $1 \leq i \leq k$ there is a transition $\alpha_i = (p_{i-1}, \mathbf{w}_i, p_i) \in T$ satisfying $\mathbf{v}_{i-1} + \mathbf{w}_i = \mathbf{v}_i$. The sequence of transitions $\alpha_1,\ \ldots,\ \alpha_k$ we call the *path* of the run (1).

We are interested in the complexity of the *reachability problem*: given a d-VASS and two configurations $p(\mathbf{v})$, $q(\mathbf{w})$ does there exist a run from $p(\mathbf{v})$ to $q(\mathbf{w})$. W.l.o.g. we can restrict $\mathbf{v} = \mathbf{w} = 0$ to be the zero vectors, as the general case polynomially reduces to such restricted case. Indeed, it suffices to add a new initial state whose only out-going transition adds $\mathbf{v}$, and likewise a new final state whose only in-going transition subtracts $\mathbf{w}$. In the sequel we usually assume that VASS is additionally equipped with a pair of configurations, a source $p(\mathbf{v})$ and a target $q(\mathbf{w})$, thus $\mathcal{V} = (Q, T, p(\mathbf{v}), q(\mathbf{w}))$. Thus we do not distinguish between a VASS and a VASS reachability instance. Runs from $p(\mathbf{v})$ to $q(\mathbf{w})$ we call *halting runs* of $\mathcal{V}$.

We study the reachability problem under two further restrictions. The first restriction assumes that the dimension d is fixed. In this case it may matter, for the complexity of the reachability problem, whether the numbers appearing in the vectors in T are encoded in unary or binary. We will thus distinguish these two cases, and speak of unary, respectively binary VASS. Note that in the unary case one can assume w.l.o.g. all vectors in T to be either the zero vector, or the unit vector $e_i = (0, \ldots, 0, 1, 0, \ldots, 0)$ with single 1 on some i-th coordinate, or inverse $-e_i$ thereof. The second restriction is *flatness* and concerns cycles

in runs (see e.g. [30, 3]). The path $\alpha_1, \ldots, \alpha_k$ of a run (1) is called *simple* if there is no repetition of states along the path; it is called *simple cycle* if there is no repetition of states along the path except for the first and the last states which are equal: $p_0 = p_k$. A VASS is *flat* if every state admits at most one simple cycle on it (i.e., the VASS has no nested cycles).

Counter Programs

We are going to represent VASSes by counter programs. A *counter program* is a numbered sequence of commands of the following types:

$\mathsf{x} \mathrel{+}= n$	(increment counter x by n)
$\mathsf{x} \mathrel{-}= n$	(decrement counter x by n)
goto L **or** L'	(jump to either line L or line L')

except that the first and the last command of the program, respectively, are of the form

initialise to 0	(initialise all counters to zero);
halt if $\mathsf{x}_1, \ldots, \mathsf{x}_l = 0$	(terminate provided all listed counters are zero).

(We note that in the unary case, increments $\mathsf{x} \mathrel{+}= m$ and decrements $\mathsf{x} \mathrel{-}= m$ can be written as m consecutive unitary increments $\mathsf{x} \mathrel{+}= 1$ and decrements $\mathsf{x} \mathrel{-}= 1$, respectively, introducing only linear blow-up. In the binary case this would lead to an exponential blow-up.) Indeed, a counter program $\mathcal{P}$ represents a VASS (in fact, a VASS reachability instance) of dimension equal to the number of counters used in $\mathcal{P}$, with a separate state for every line in $\mathcal{P}$. The increment and decrement commands in P are simulated by transition vectors of the VASS. The source and target configurations of the VASS correspond to the first and last line of $\mathcal{P}$. The size of the VASS is linear with respect to the size of the program. This convenient representation was adopted e.g. in [15, 11].

Accordingly with runs of a VASS, we speak of runs of a counter program (in particular, values of counters along a run are nonnegative) with the proviso that the initial value of all counters is 0. A run is *halting* if it has successfully executed its (necessarily last) **halt** command; otherwise, the run is *partial*. The reachability problem for a VASS translates into the question whether there exists a halting run in a counter program.

Note that a counter program does not need to test for zero all counters in the final **halt** command; for the sake of presentation it is convenient to allow for halting runs with non-zero final value of certain (irrelevant) counters. On the other hand, formally, our intention is that a counter program represents a VASS reachability instance with the zero target vector. This incompatibility can be circumvented by assuming that counter programs are implicitly completed with additional loops allowing to decrease every untested counter just before executing the **halt** command.

In case of fragments of counter programs which neither start with **initialise** nor end with **halt**, we consider explicit *initial* and *final* values of counters. Note however that due to nondeterministic **goto** command, final values are not uniquely determined by initial ones.

When writing counter program we use a syntactic sugar: we write **goto** L instead of **goto** L **or** L, and whenever a program repeats the block of commands in line 2 some nondeterministically chosen number of times (possibly zero, possibly infinite), as shown on the left, we use a shorthand as shown on the right:

1: **goto** 4 **or** 2
2: <iterated commands>
3: **goto** 1
4: <remaining commands>

1: **loop**
2: <iterated commands>
3: <remaining commands>

In the sequel we will only occasionally use **goto** commands *explicitly*. Observe that a counter program without explicit **goto** commands, but using *unnested* **loop** commands (which *implicitly* use **goto** commands), always represents a flat VASS.

Algorithm I Weak multiplication by $\frac{c}{d}$, for $c > d$.

```
1: loop
2:     x −= 1    y += 1
3: loop
4:     x += c    y −= d
```

We end this section with examples of counter programs that *weakly* compute a number b in some counter x, i.e., all runs end with $\mathsf{x} \leq b$, and there is a run that ends with $\mathsf{x} = b$. On the way we also introduce macros to be used later to facilitate writing complex programs. As a preparation, consider the program in Algorithm I which weakly multiplies the initial value of x by $\frac{c}{d}$.
Let x_0, y_0 and x_1, y_1 be initial and final values, respectively, of counters x, y. We claim that the sum of final values is at most $\frac{c}{d}$ times larger than the sum of initial values. Moreover, it is exactly $\frac{c}{d}$ times larger if, and only if, both loops are iterated *maximally*: the first loop exits only when the counter x, decreased in its every iteration, reaches the minimal possible value 0; and likewise the second loop exits only when the counter y reaches 0. Enforcing maximal iteration of loops will be our fundamental technical objective in the sequel.

▷ Claim 1. Let x', y' be the values of counters x, y at the exit from the first loop. Then $x_1 + y_1 \leq (x_0 + y_0) \cdot \frac{c}{d}$. Moreover, $x_1 = (x_0 + y_0) \cdot \frac{c}{d}$ if, and only if, $x' = y_1 = 0$.

Proof. As $x' + y' = x_0 + y_0$ and $c > d$ we get:

$$x_1 + y_1 \leq x' + \frac{c}{d} \cdot y' \leq \frac{c}{d} \cdot (x_0 + y_0). \tag{2}$$

We now concentrate on the second part of the claim. If $x' = y_1 = 0$ then $d \mid (x_0 + y_0)$ and thus $x_1 = (x_0 + y_0) \cdot \frac{c}{d}$. For the opposite direction, if $y_1 \neq 0$ then $x_1 < x_1 + y_1 \leq (x_0 + y_0) \cdot \frac{c}{d}$. If $x' \neq 0$ then by (2) we get

$$x_1 \leq x' + \frac{c}{d} \cdot y' < \frac{c}{d} \cdot (x' + y') = \frac{c}{d} \cdot (x_0 + y_0). \qquad \triangleleft$$

Algorithm II Program fragment $\mathcal{W}_b$.

```
1: initialise to 0
2: for i := m downto 0 do
3:     loop
4:         x −= 1    y += 1
5:     loop
6:         x += 2    y −= 1
7:     if b_i = 1 then x += 1
```

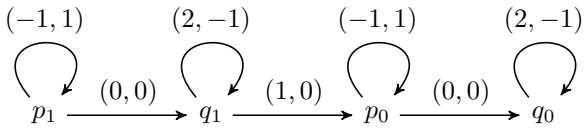


Figure 1 A 2-VASS represented by the program $\mathcal{W}_2$. The first coordinate corresponds to the value of counter x and the second one to the value of counter y.

Algorithm III Unfolding of macros in $\mathcal{W}_2$.

```
1:  initialise to 0
2:  loop
3:      x −= 1    y += 1
4:  loop
5:      x += 2    y −= 1
6:  x += 1
7:  loop
8:      x −= 1    y += 1
9:  loop
10:     x += 2    y −= 1
```

The counter program $\mathcal{W}_b$ shown in Algorithm II weakly computes a number b, assuming that $b_m \dots b_0 = \textsc{Bin}(b)$ is the binary representation of b (the oldest bit $b_m = 1$). The **halt** command is omitted as no zero-testing is relevant in this example. We use **for** and **if then** preprocessing macros with the following semantics. The macro

for i := m **downto** 0 **do** <program fragment>

is understood as $(m+1)$-fold repetition of copies of <program fragment>:

<program fragment> $(i = m)$
<program fragment> $(i = m-1)$
...
<program fragment> $(i = 0)$

for $i = m, m-1, \dots, 0$. It is important that i is not a counter but a meta-variable that is treated as a constant in every program fragment. By convention we use different fonts for counters and meta-variables: i is a counter while i is a meta-variable. Furthermore in every copy, say for $i = k$, at every appearance of the macro **if** $\varphi(i)$ **then** <optional program fragment>, the formula $\varphi(i)$ is evaluated and, if it evaluates positively then macro is replaced by <optional program fragment>, otherwise it is removed. Specifically, consider for example $m = 1$ and $b = 2$, i.e., $b_1 = 1$ and $b_0 = 0$. Unfolding of the macros appearing in $\mathcal{W}_2$ yields the counter program shown in Algorithm III. Figure 1 shows the corresponding 2-VASS. We remark that the programs in Algorithm I and Algorithm II represent flat VASS.

Clearly, we do not want **for** and **if then** to be full-fledged commands operating on program counters, as this would make counter programs as powerful as Minsky machines, hence undecidable. They are just pre-processing macros that operate on meta-variables i only, and constitute syntactic sugar helpful in writing repetitive program fragments.

▶ **Proposition 2.** *The program $\mathcal{W}_b$ weakly computes b.*

Proof. By Claim 1, the program $\mathcal{W}_b$ weakly multiplies x by 2 in lines (3)–(6). Combining this with addition of a bit in (7) gives weak computation of b. ◀

3 Exponential Shortest Runs

▶ **Theorem 3.** *There is a family of unary flat 3-VASS $(\mathcal{V}_n)_{n\in\mathbb{N}}$ of size $\mathcal{O}(n^2)$ such that every halting run of $\mathcal{V}_n$ is of length exponential in n.*

Algorithm IV Counter program $\mathcal{P}_n$.

```
1:  initialise to 0
2:  x += 1    y += 1
3:  loop
4:      x += 1    y += 1
5:  for i := n down to 1 do
6:      loop
7:          x −= 1    z += 1
8:      loop
9:          x += i + 1    z −= i
10: loop
11:     x −= n + 1    y −= 1
12: halt if y = 0.
```

In this section we prove the theorem. The VASS $\mathcal{V}_n$ are represented by the counter programs $\mathcal{P}_n$ shown in Algorithm IV. The idea of multiplying by consecutive fractions $\frac{2}{1}, \frac{3}{2} \dots \frac{n+1}{n}$ comes from [11] (cf. Algorithms I,II therein), however, we need to apply the multiplications in the reverse order. The size of $\mathcal{P}_n$ is quadratic in n, as the **for** macro unfolds n times, and the constants appearing in the increment/decrement commands, like $i+1$ in x += $i+1$, are written in unary. Consider any run that reaches (but not yet executes) line 12. For every $i = n, \dots, 1$ let x_i and z_i be the values of counters x and z, respectively, at the exit from the loop in lines 8–9. Similarly, let x'_i be the value of counter x at the exit from the loop in lines 6–7. For uniformity we write x_{n+1} and z_{n+1} for the values of x and z, respectively, just before entering the **for** macro, and call these values initial. Notice that x_{n+1} is equal to the value of counter y at that point and $z_{n+1} = 0$. By Claim 1 we derive:

▷ Claim 4. For all $i = 1, \dots, n$, we have $x_i + z_i \leq (x_{i+1} + z_{i+1}) \cdot \frac{i+1}{i}$.

We focus on runs that *maximally iterate* both inner loops, by which we mean:

- the value of x is 0 at the exit of the loop in lines 6–7;
- the value of z is 0 at the exit of the loop in lines 8–9;

▷ Claim 5. We have $x_1 \leq x_{n+1} \cdot (n+1)$. The equality holds if, and only if, $z_i = x'_i = 0$ for all $i = 1, \dots, n$.

Proof. By Claim 4 we get $x_1 + z_1 \leq (x_{n+1} + z_{n+1}) \cdot \prod_{i=1}^{n} \frac{i+1}{i} = (\mathsf{x}_{n+1} + \mathsf{z}_{n+1}) \cdot (n+1)$. Since $z_{n+1} = 0$ this implies the inequality.

Now we step to the second part of the claim. If $z_i = x'_i = 0$ for all $i = 1, \dots, n$ then by Claim 1 we get $x_i = x_{i+1} \cdot \frac{i+1}{i}$ for every i, which implies $x_1 = x_{n+1} \cdot (n+1)$.

Conversely, suppose for some i we have $z_i \neq 0$ or $x'_i \neq 0$. Then by Claim 1 we get $x_i + z_i < (x_{i+1} + z_{i+1}) \cdot \frac{i+1}{i}$. Combined with Claim 4 this yields $x_1 + z_1 < (x_{n+1} + z_{n+1}) \cdot (n+1)$, which concludes the proof as $z_{n+1} = 0$. ◁

For a finite subset $X \subseteq \mathbb{N}$ of natural numbers, we write $\textsc{Lcm}(X)$ for the least common multiple of all numbers in X. We will use the number $N(n)$ defined as

$$N(n) \quad := \quad \frac{\textsc{Lcm}(\{2,\ldots,n+1\})}{n+1}.$$

The following two claims conclude the proof of Theorem 3.

▷ Claim 6. The function N grows exponentially with respect to n. The binary representation $\textsc{Bin}(N(n))$ is computable in time polynomial with respect to n.

Proof. For the exponential upper bound we recall that $N(n) \leq n!$ For the exponential lower bound we use the prime number theorem (proved independently by Jacques Hadamard and Charles Jean de la Vallée Poussin in 1896): the number of primes $\pi(n)$ between 2 and n is at least $\pi(n) \geq c \cdot n^\epsilon - 1$ for some constants $c > 0$ and $0 < \epsilon < 1$. Since $\textsc{Lcm}(\{2 \ldots n+1\})$ must be divisible by all prime numbers between 2 and $n+1$ and each prime number is at least 2 we get $\textsc{Lcm}(\{2,\ldots,n+1\}) \geq 2^{\pi(n+1)}$.

$N(n)$ is computed by exhaustive enumeration of all non-divisors of $n+1$, computing their prime decompositions, and combining them into prime decomposition of $N(n)$. ◁

▷ Claim 7. For every initial value x_{n+1} of counter x there is at most one halting run. Such a run exists if, and only if, x_{n+1} is a positive multiple of $N(n)$.

Proof. Recall that the last loop in $\mathcal{P}_n$ in line 11 decreases simultaneously y by 1 and x by $n+1$. Therefore, the run halts only if $\mathsf{x} \geq \mathsf{y} \cdot (n+1)$. By Claim 5 we have $\mathsf{x} \leq \mathsf{y} \cdot (n+1)$. Thus every halting run satisfies the equality $\mathsf{x} = \mathsf{y} \cdot (n+1)$. By Claim 5 we know that is possible only if $\mathsf{z}_i = \mathsf{x}'_i = 0$ for all $i = 1,\ldots,n$, which uniquely determines the run for a given x_{n+1}. It remains to prove that a halting run exists if, and only if, x_{n+1} is a positive multiple of $N(n)$. Notice that by Claim 4 and Claim 5 in a halting run

$$\mathsf{x}_i = \mathsf{x}_{i+1} \cdot \frac{i+1}{i} = \ldots = \mathsf{x}_{n+1} \cdot \prod_{j=i}^{n} \frac{j+1}{j} = \mathsf{x}_{n+1} \cdot \frac{n+1}{i}.$$

Therefore $\mathsf{x}_{n+1} \cdot (n+1)$ must be always divisible by all numbers in $\{1 \ldots n\}$. Since $n+1$ divides $\mathsf{x}_{n+1} \cdot (n+1)$ as well, we deduce that $\textsc{Lcm}(\{2,\ldots,n+1\})$ divides $\mathsf{x}_{n+1} \cdot (n+1)$ which is equivalent to $N(n)$ divides x_{n+1}. Conversely, if x_{n+1} is a multiple of $N(n)$ then there is a run where all loops are iterated maximally, satisfying $\mathsf{x}_i = \mathsf{x}_{i+1} \cdot \frac{i+1}{i}$ and thus halting. ◁

4 NP-hardness

This section is devoted to proving NP-lower bound for flat VASS in fixed dimension.

▶ **Theorem 8.** *The reachability problem for unary flat 7-VASS is NP-hard.*

As mentioned in the introduction NP-membership is already known (even in binary VASS of unrestricted dimension). Thus as a corollary we get the following result.

▶ **Corollary 9.** *The reachability problem for flat d-VASS is NP-complete for fixed $d \geq 7$.*

To prove Theorem 8 we reduce from the Subset Sum problem: given a set of positive integers $S = \{s_1,\ldots,s_k\} \subseteq \mathbb{N} - \{0\}$ and an integer $s_0 > 0$, determine if some subset $R \subseteq S$ satisfies $\sum_{s \in R} s = s_0$. Note that all the numbers $s_0, s_1, \ldots, s_k$ are encoded in binary.

Fix an instance $s_0, s_1, \ldots, s_k$ of the Subset Sum problem and let n be the smallest natural number such that $N(n) \geq s_0, s_1, \ldots, s_k$. By Claim 6 the number n as well as the binary representation $\text{Bin}(N(n)) = b_m \ldots b_0$ is computable in time polynomial with respect to the sizes of binary representations of $s_0, s_1, \ldots, s_k$. Recall that $b_m = 1$. We are going to define a unary counter program $\mathcal{P}$ of polynomial size using 7 counters, as a function of $s_0, s_1, \ldots, s_k$ and n, which halts if, and only if, the instance $s_0, s_1, \ldots, s_k$ is positive.

The main obstacle is that the numbers in the Subset Sum problem are represented in binary, while the numbers in a counter program are to be represented in unary. Thus we have to exactly compute with exponential numbers, using a fixed number of 7 counters.

Construction of $\mathcal{P}$

We face the challenge by combining the weak computation given by Proposition 2 (that allows us to compute *at most* a required value b) with the insight of the proof of Theorem 3 (that enforces that the computed value is simultaneously *at least* b).

Algorithm V Counter program $\mathcal{I}$.

```
 1: initialise to 0
 2: x += 1   y += 1   e += 1   f += k + 1
 3: for i := m − 1 to 0 do
 4:     loop
 5:         x −= 1   x′ += 1
 6:         y −= 1   e −= 1   f −= k + 1
 7:     loop
 8:         x += 2   x′ −= 1
 9:         y += 2   e += 2   f += 2(k + 1)
10:     if b_i = 1 then
11:         x += 1  y += 1  e += 1  f += k + 1
12: for i := n down to 1 do
13:     loop
14:         x −= 1   z += 1
15:     loop
16:         x += i + 1   z −= i
17: loop
18:     x −= n + 1   y −= 1
19: halt if y = 0                    // removed in I′
```

Program $\mathcal{I}$ in Algorithm V implements this idea. The first half of the program, namely lines 1–11, weakly computes in counter e the value $N(n)$, and in counter f the value $N(n) \cdot (k+1)$, very much like the counter program $\mathcal{W}_b$. Note a slight difference compared to Algorithm II: the oldest bit $b_m = 1$ is treated in a different way than other bits b_i for $0 \leq i < m$, by initializing counters e and f to 1 and $k+1$, respectively, which excludes a trivial halting run that would never iterate any loop and end with the value of y equal 0. Then the second part of $\mathcal{I}$ checks, very much like the counter program $\mathcal{P}_n$, if the values are computed exactly. (Notice that lines (12)–(19) are exactly the same as lines 5–12 of Algorithm IV.) Using Claim 7 we get:

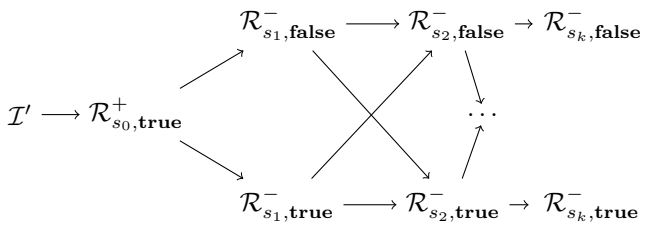
▷ Claim 10. Counter program $\mathcal{I}$ has exactly one halting run that computes $N(n)$ and $N(n) \cdot (k+1)$ in counters e and f, respectively, and 0 in the remaining counters $\mathsf{x}, \mathsf{x}', \mathsf{y}$ and z.

The program $\mathcal{P}$ (shown in Algorithm VI) consists of the program $\mathcal{I}'$ obtained from $\mathcal{I}$ by removing the last **halt** command. The remaining part of $\mathcal{P}$ exploits the values of counters e and f computed by $\mathcal{I}'$ to turn weak computations of exponential numbers into exact ones. It never modifies the counter y again, hence y is listed in the final **halt** command of $\mathcal{P}$, and uses a distinguished counter u, initially set to 0, a program fragment $\mathcal{R}^+_{s_0,\mathbf{true}}$, and a number of program fragments $\mathcal{R}^-_{s,p}$ for $s \in \{s_1, s_2 \dots s_k\}$ and $p \in \{\mathbf{true}, \mathbf{false}\}$. We call these program fragments *components*. In every halting run of $\mathcal{P}$, the component $\mathcal{R}^-_{s,\mathbf{true}}$ decrements u by s while the other component $\mathcal{R}^-_{s,\mathbf{false}}$ has no effect on counter u. Likewise, the component $\mathcal{R}^+_{s_0,\mathbf{true}}$ increments u by s_0. Finally, u is zero-tested by the final **halt** command.

Algorithm VI Program $\mathcal{P}$.

$\mathcal{I}'$
$\mathcal{R}^+_{s_0,\mathbf{true}}$
goto f_1 **or** t_1
f_1: $\mathcal{R}^-_{s_1,\mathbf{false}}$ **goto** f_2 **or** t_2
t_1: $\mathcal{R}^-_{s_1,\mathbf{true}}$ **goto** f_2 **or** t_2
f_2: $\mathcal{R}^-_{s_2,\mathbf{false}}$ **goto** f_3 **or** t_3
t_2: $\mathcal{R}^-_{s_2,\mathbf{true}}$ **goto** f_3 **or** t_3
...
f_k: $\mathcal{R}^-_{s_k,\mathbf{false}}$ **goto** h
t_k: $\mathcal{R}^-_{s_k,\mathbf{true}}$
h: **halt if** $\mathsf{y}, \mathsf{u}, \mathsf{f} = 0$

For $1 \le i \le k$, we use f_i, respectively t_i, to denote the the first line of the program fragment $\mathcal{R}^*_{s_i,\mathbf{false}}$, respectively $\mathcal{R}^*_{s_i,\mathbf{true}}$. Every component $\mathcal{R}^*_{s_i,p}$, for $0 \le i < k$ and $* \in \{+,-\}$, is followed by **goto** f_{i+1} **or** t_{i+1}. Thus for every $i = \{1 \dots k\}$ either $\mathcal{R}^-_{s_i,\mathbf{false}}$ or $\mathcal{R}^-_{s_i,\mathbf{true}}$ is executed, as shown by the following control flow diagram of $\mathcal{P}$:



Observe that every halting run of $\mathcal{P}$ determines a subset $R \subseteq \{1, \dots, k\}$ such that for $i \in R$ the component $\mathcal{R}^-_{s_i,\mathbf{true}}$ is executed, while for $i \notin R$ the component $\mathcal{R}^-_{s_i,\mathbf{false}}$ is executed.

The Components

The component $\mathcal{R}^*_{a,p}$ is shown in Algorithm VII. By $\textsc{Bin}_m(a) = a_m \dots a_0$ we mean the $(m+1)$-bit binary representation of the number $a < 2^{m+1}$, padded with leading 0 bits if needed. The aim of every $\mathcal{R}^*_{a,\mathbf{true}}$ is to increment (when $* = +$) or decrement (when $* = -$) a from the counter u, using the counters e and f to enforce exactness. After the auxiliary counter v is initialised to 1, in every iteration of the **for** loop (in lines 2-9) counter v is weakly multiplied by 2, so after i iterations its value is at most 2^i.

Algorithm VII Component $\mathcal{R}^*_{a,p}$.

```
1:  v += 1
2:  for j := 0 to m − 1 do
3:      loop
4:          v −= 1    v′ += 1
5:          if b_j = 1 then
6:              e −= 1  e′ += 1  f −= 1
7:          if p ∧ (a_j = 1) then u *= 1
8:      loop
9:          v += 2    v′ −= 1
10: loop
11:     v −= 1
12:     e −= 1    e′ += 1    f −= 1
13:     if p ∧ (a_m = 1) then u *= 1
14: loop
15:     e += 1    e′ −= 1
```

In lines 6 and 12 counter f is decremented, in both cases together with counter e, hence the total decrement of f is at most the initial value of counter e. Now recall that in a halting run of $\mathcal{P}$ the values of e and f output by $\mathcal{I}'$ are $N(n)$ and $N(n) \cdot (k+1)$, respectively. As every halting run of $\mathcal{P}$ passes through exactly $k+1$ components and f is zero-tested by the final **halt** command of $\mathcal{P}$, every of the components forcedly decrements f by exactly $N(n)$. Also forcedly, after i iterations of the for loop in lines 2-9 the value of counter v is exactly 2^i. This in consequence implies that the counter u is incremented (respectively, decremented) in lines 7 and 13 by exactly a times, hence by a in total. Lines 14-15 are to revert the roles of counters e and e'.

Note that the oldest bit a_m, irrespectively of its value 0 or 1, is treated differently (in lines 10-13) from the other bits $a_{m-1} \dots a_0$ of $\textsc{Bin}_m(a)$ (treated in the body of the **for** loop in lines 2–9). This is because the auxiliary counter v needs to be multiplied by 2 exactly m times, which happens in the course of m iterations of the **for** loop, while the number of bits in $\textsc{Bin}(a)$ is $m+1$, thus larger by 1. Consequently, in lines 10-13 the value of v is not flashed to v′ nor restored back from v′, and hence v is forcedly 0 at the end of $\mathcal{R}^*_{a,\mathbf{true}}$ and can be reused by the following commands. Note that the **if** macro is used in line 13 as, due to the choice of m, the oldest bit b_m of $\textsc{Bin}(N(n))$ is 1.

The above analysis applies equally well to every component $\mathcal{R}^*_{a,\mathbf{false}}$, as its computation is exactly the same as that of $\mathcal{R}^*_{a,\mathbf{true}}$, except that the value of u is not changed.

Dimension 7

To estimate the dimension of the VASS represented by $\mathcal{P}$, notice that $\mathcal{I}'$ uses counters x, x′, y, z, e, f and components $\mathcal{R}^*_{s_i,p}$ use additionally v, v′, e′, u. However, by Claim 10 the final values of x, x′, z computed by $\mathcal{I}'$ are 0 in every halting run of $\mathcal{P}$, hence the three counters can be reused in components, which reduces the number of counters to 7.

5 Doubly Exponential Shortest Runs

▶ **Theorem 11.** *There is a family of binary 4-VASS $(\mathcal{V}_n)_{n\in\mathbb{N}}$ of size $\mathcal{O}(n^3)$ such that every halting run of $\mathcal{V}_n$ is of length doubly exponential in n.*

In this section we prove the theorem. Define the *description size* of an irreducible fraction $\frac{p}{q}$ as $\max\{p,q\}$. We start with a key technical lemma stating existence of arbitrarily long increasing sequences of rationals greater than 1, of description size exponential with respect to k, with the property that the result of multiplying consecutive exponential powers of these rationals has only exponential (and not doubly exponential) description size.

▶ **Lemma 12.** *For each $k \geq 1$ there are k rational numbers*

$$1 < f_1 < \ldots < f_k = 1 + \frac{1}{4^k}, \tag{3}$$

of description size bounded by 4^{k^2+k}, such that the description size of f defined by

$$f = (f_k)^{2^k} \cdot \ldots \cdot (f_2)^{2^2} \cdot (f_1)^{2^1} \tag{4}$$

is bounded by $4^{2(k^2+k)}$.

Proof. For $1 \leq i \leq k$ put $r_i := \frac{4^k+2^{k-i}}{4^k}$, and observe the following (straightforward) equalities:

$$\left(\frac{1}{r_i}\right)^{2^1} \cdot \left(\frac{1}{r_i}\right)^{2^2} \cdot \ldots \cdot \left(\frac{1}{r_i}\right)^{2^{i-1}} \cdot r_i^{2^i} \;=\; r_i^2.$$

Multiplying all these equalities yields the equality:

$$f_1^{2^1} \cdot f_2^{2^2} \cdot \ldots \cdot f_k^{2^k} \;=\; f, \quad \text{where} \quad f_i \;=\; \frac{r_i}{r_{i+1} \cdot \ldots \cdot r_k} \qquad f \;=\; (r_1 \cdot \ldots \cdot r_k)^2 . \tag{5}$$

As numerators and denominators of all r_i are bounded by 4^{k+1}, numerators and denominators of all f_i are bounded by 4^{k^2+k}, and numerator and denominator of f are bounded by $4^{2(k^2+k)}$, as required.

It remains to argue that the (in)equalities (3) hold. We notice the following relation between r_i and r_{i-1}, for $1 < i \leq k$:

$$r_i^2 \;=\; \left(1 + \frac{2^{k-i}}{4^k}\right)^2 \;>\; 1 + \frac{2^{k+1-i}}{4^k} \;=\; r_{i-1}, \tag{6}$$

which implies

$$\frac{f_i}{f_{i-1}} \;=\; \frac{r_i \cdot (r_i \cdot \ldots \cdot r_k)}{r_{i-1} \cdot (r_{i+1} \cdot \ldots \cdot r_k)} \;=\; \frac{r_i^2}{r_{i-1}} \;>\; 1$$

and hence $f_1 < f_2 < \ldots < f_k$. For $i = k$ we have $f_k = r_k = 1 + \frac{1}{4^k}$. It thus remains to show $f_1 > 1$, which is equivalent to

$$r_1 > r_2 \cdot \ldots \cdot r_k. \tag{7}$$

By (6) we deduce $r_k^{2^i} > r_{k-i}$, by induction on i, which implies the following inequality:

$$r_k^{2^{k-1}-1} \;=\; r_k^{1+2+4+\ldots+2^{k-2}} \;>\; r_2 \cdot \ldots \cdot r_k.$$

For (7) it suffices to show, relying on the above inequality, that $r_1 > r_k^{2^{k-1}-1}$. Put $N := 2^{k-1}-1$ for convenience. We thus need to prove:

$$r_1 \;>\; \left(1 + \frac{1}{4^k}\right)^N. \tag{8}$$

By inspecting the expansion of the right-hand side

$$\left(1+\frac{1}{4^k}\right)^N \;=\; \sum_{i=0}^{N} \binom{N}{i} \cdot \frac{1}{4^{ik}}$$

we observe that the right-hand side is bounded by the sum of first N elements of a geometric progression, which, in turn, is bounded by the sum of the whole infinite one:

$$\left(1+\frac{1}{4^k}\right)^N \;\leq\; 1+\frac{N}{4^k}+\frac{N^2}{4^{2k}}+\ldots+\frac{N^N}{4^{Nk}} \;<\; \frac{1}{1-\frac{N}{4^k}}.$$

Thus for showing (8) it is sufficient to prove the inequality $r_1 > \frac{1}{1-\frac{N}{4^k}}$, which is equivalent to

$$\left(1-\frac{2^{k-1}-1}{4^k}\right)\left(1+\frac{2^{k-1}}{4^k}\right) \;>\; 1.$$

The latter inequality is easily verified to hold true as

$$\frac{1}{4^k} \;>\; \frac{2^{k-i}-1}{4^k}\cdot\frac{2^{k-i}}{4^k}.$$

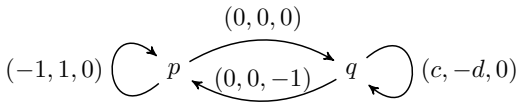
The inequality (8) is proved, and hence so is Lemma 12. ◀

A distinguished counter x in the 4-VASS $\mathcal{V}_k$ will play a special role: in every halting run, x will be exactly multiplied by consecutive powers as in (4). As the denominator of the irreducible form of f_k is at least 2, the counter x, just before the very first multiplication by $(f_k)^{2^k}$, must be divisible by the denominator of f_k to the power 2^k, which is doubly exponential in k. In consequence, every halting run has to be doubly exponentially long.

Algorithm VIII Program fragment $\mathcal{HP}(c,d)$; counters x, y and z correspond to dimension 1, 2 and 3, respectively, of the VASS.

```
1: loop
2:     loop
3:         x −= 1    y += 1
4:     loop
5:         x += c    y −= d
6:     z −= 1
```

As before, the main difficulty is to turn weak multiplications into exact ones. To this aim we will rely on Lemma 12 and on a well-known weakly exponentiating 3-VASS gadget of Hopcroft and Pansiot [24]:



The gadget is represented by the program fragment $\mathcal{HP}(c,d)$ shown in Algorithm VIII.

▶ **Proposition 13.** *Consider program fragment $\mathcal{HP}(c,d)$ for an irreducible fraction $\frac{c}{d} > 1$, and initial values x_0, y_0, z_0 of counters* x, y *and* z. *In every run, the respective final values x_1, y_1, z_1 satisfy*

$$x_1 + y_1 \leq (x_0+y_0)\cdot\left(\frac{c}{d}\right)^{z_0-z_1}.$$

Moreover, there is a run satisfying $x_1 = (x_0+y_0)\cdot\left(\frac{c}{d}\right)^{z_0}$ if, and only if, x_0+y_0 is divisible by d^{z_0}. In this case $y_1 = z_1 = 0$.

Proof. The two inner loops (lines 2–5) coincide with the counter program fragment shown in Algorithm I. As the outer loop is executed $z_1 - z_0$ times, the first part follows by Claim 1.

For the second part, assume $x_0 + y_0$ is divisible by d^{z_0}, and consider the unique run where all the loops are iterated maximally, by which we mean:

- the outer loop (lines 1–6) is executed exactly z_0 times;
- whenever execution of the first inner loop (lines 2–3) ends, the value of x is 0;
- whenever execution of the second inner loop (lines 4–5) ends, the value of y is 0;

Thus every execution of the two inner loops necessarily multiplies the sum $\mathsf{x} + \mathsf{y}$ by $\frac{c}{d}$, and consequently, after i iterations of the outer loop the values of respective counters x', y', z' satisfy

$$x' \;=\; (x_0 + y_0) \cdot \left(\frac{c}{d}\right)^{z_0 - i} \qquad y' \;=\; 0 \qquad z' \;=\; z_0 - i. \tag{9}$$

Repeating the multiplication z_0 times yields $x_1 = \left(\frac{c}{d}\right)^{z_0}$ and $y_1 = z_1 = 0$, as required.

Conversely, suppose $x_1 = (x_0 + y_0) \cdot \left(\frac{c}{d}\right)^{z_0}$. As c and d are co-primes, the sum of initial values $x_0 + y_0$ is thus forcedly divisible by d^{z_0}. By the first part we know that the outer loop has been iterated maximally, hence $z_1 = 0$. Then $y_1 = 0$ follows by the first part. ◀

Construction of $\mathcal{V}_k$

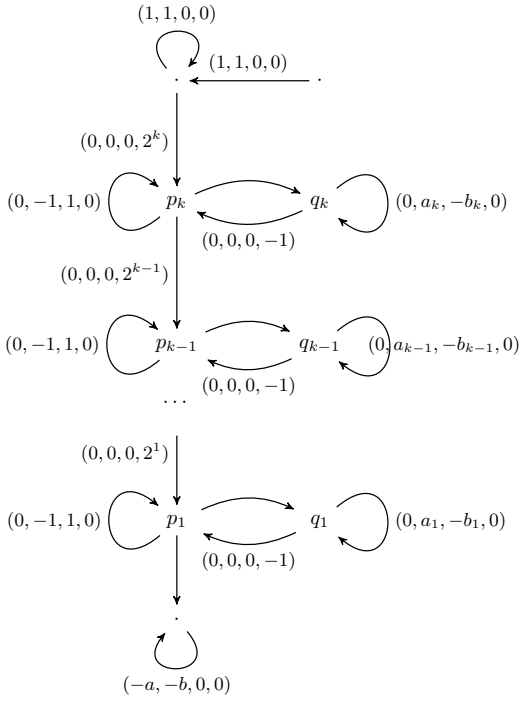
Fix $k \geq 1$. Let $f_i = \frac{a_i}{b_i}$, for $i \leq i \leq k$, be the fractions from Lemma 12, and let $f = \frac{a}{b}$ be the result of their multiplication as in (4). We thus have:

$$\left(\frac{a_1}{b_1}\right)^{2} \cdot \left(\frac{a_2}{b_2}\right)^{2^2} \cdot \ldots \cdot \left(\frac{a_k}{b_k}\right)^{2^k} \quad = \quad \frac{a}{b}. \tag{10}$$

Algorithm IX (on the left below) shows the counter program representing the 4-VASS V_k (on the right below), using four counters $\mathsf{t}, \mathsf{x}, \mathsf{y}$ and z. The constants appearing in increment and decrement commands are exponential in k, represented in binary in size $\mathcal{O}(k^2)$. The length of $\mathcal{V}_k$ is $\mathcal{O}(k)$ and hence its size is $\mathcal{O}(k^3)$.

Algorithm IX Program representing 4-VASS $\mathcal{V}_k$ shown on the right. Counters $\mathsf{t}, \mathsf{x}, \mathsf{y}$ and z correspond to consecutive dimensions.

```
1:  initialise to 0
2:  t += 1    x += 1
3:  loop
4:      t += 1    x += 1
5:  for  i  :=  k  down to  1 do
6:      z += 2^i
7:      loop
8:          loop
9:              x −= 1    y += 1
10:         loop
11:             x += a_i    y −= b_i
12:         z −= 1
13: loop
14:     t −= b    x −= a
15: halt if t = 0
```



▷ Claim 14. For every $k \geq 0$, the 4-VASS $\mathcal{V}_k$ has a halting run.

Proof. Put $N := \prod_{i=1\ldots k}(b_i)^{2^i}$. By performing the first loop (lines 3–4) exactly $N-1$ times, the run reaches the following valuation of counters $\mathsf{x}, \mathsf{y}, \mathsf{z}$:

$$x_k \;=\; N \qquad\qquad y_k \;=\; z_k \;=\; 0. \tag{11}$$

Notice that the outer loop (lines 7–12) coincides with the program fragment $\mathcal{HP}(a_i, b_i)$. We use the second part of Proposition 13 for consecutive iterations of the **for** macro. The proposition allows us to derive a run where the values x_j, y_j, z_j of counters x, y, z, after $k-j$ iterations of the **for** macro (for $j \in \{0, \ldots, k\}$), satisfy:

$$x_j \;=\; N \cdot \left(\frac{a_j}{b_j}\right)^{2^j} \cdot \;\ldots\; \cdot \left(\frac{a_k}{b_k}\right)^{2^k} \qquad y_j \;=\; z_j \;=\; 0. \tag{12}$$

Indeed, by induction with respect to $k-j$ (using (11) as induction base for $j = k$), we argue as follows: if (12) holds then x_j is divisible by $(b_{j-1})^{2^{j-1}}$, and hence by Proposition 13 there is a continuation of the run that yields

$$x_{j-1} \;=\; x_j \cdot \left(\frac{a_{j-1}}{b_{j-1}}\right)^{2^{j-1}} \qquad y_{j-1} \;=\; z_{j-1} \;=\; 0.$$

In consequence, for $j = 0$ we obtain, using (10):

$$x_0 \;=\; N \cdot \frac{a}{b} \qquad\qquad y_0 \;=\; z_0 \;=\; 0.$$

As the counter t is not modified inside the **for** loop (lines 5–12), its value is still equal to N after **for** loop is finished. Thus, by executing N iterations of the last loop (in lines 13–14) we reach the value 0 of all the four counters $\mathsf{t}, \mathsf{x}, \mathsf{y}, \mathsf{z}$ and hence halt in line 15. Summing up, every $\mathcal{V}_k$ admits a halting run. ◁

Proof of Theorem 11. We argue that every halting run of $\mathcal{V}_k$ has length at least doubly exponential in k. Consider an arbitrary halting run, i.e., a run reaching the final value $\mathsf{t} = 0$ in line (15). As before, let x_j, y_j and z_j, for $j = 0, \ldots, k$, stand for the values of counters x, y and z, respectively, after $k - j$ iterations of the **for** macro. Let $x_k = N \geq 1$ be the value of the counters t and x after exiting from the first loop (lines 3–4); cf. (11). The counter t is not modified inside the **for** loop (lines 5–12). Thus the last loop (in lines 13–14) has to be performed exactly $\frac{N}{b}$ times, which implies

$$x_0 \;\geq\; N \cdot \frac{a}{b}. \tag{13}$$

Let $n_k, n_{k-1}, \ldots, n_1$ stand for the number of iterations of the outer loop (lines 7–12) in consecutive iterations of the **for** macro. By the very structure of $\mathcal{V}_k$ we know that, for every $1 \leq i \leq k$,

$$\sum_{j=i}^{k} n_j \;\leq\; \sum_{j=i}^{k} 2^j. \tag{14}$$

We aim to show that the inequality (13) implies $n_j = 2^j$ for every $j \in \{1, \ldots, k\}$. As the outer loop (lines 7–12) coincides with the program fragment $\mathcal{HP}(a_i, b_i)$, we may apply the first part of Proposition 13 to derive, similarly as above:

$$x_j \;\leq\; N \cdot \left(\frac{a_j}{b_j}\right)^{n_j} \cdot \;\ldots\; \cdot \left(\frac{a_k}{b_k}\right)^{n_k}. \tag{15}$$

Claim 15 will imply that, roughly speaking, the biggest value of x_j is obtained, if in every unfolding of the **for** macro we perform the maximal possible number of iterations of the outer loop, and hence finish with the counter value $\mathsf{z} = 0$.

▷ Claim 15. Assuming (14), $\left(\frac{a_1}{b_1}\right)^{n_1} \cdot \left(\frac{a_2}{b_2}\right)^{n_2} \cdot \;\ldots\; \cdot \left(\frac{a_k}{b_k}\right)^{n_k} \;\leq\; \frac{a}{b}$. The equality holds if, and only if, $n_j = 2^j$ for all $j \in \{1, \ldots, k\}$.

Proof. For vectors $(n_1, \ldots, n_k)$ satisfying (14), we define the function $f(n_1, \ldots, n_k) = \left(\frac{a_1}{b_1}\right)^{n_1} \cdot \ldots \cdot \left(\frac{a_k}{b_k}\right)^{n_k}$. Thus (10) says that $f(2^1, \ldots, 2^k) = \frac{a}{b}$. Observe that any other vector $(n_1, \ldots, n_k)$ satisfying (14) is obtained from $(2^1, \ldots, 2^k)$ by applying a number of times one of the following two operations:

1. decrement some n_i by 1
2. decrement some n_i by 1 and increment n_{i-1} by 1.

As any of this operations strictly decreases the value of f, Claim 15 follows. ◁

By the first part of Claim 15, together with inequalities (14) and (15) we deduce $x_0 \;\leq\; N \cdot \frac{a}{b}$ which, combined with (13) yields the equality:

$$x_0 \;=\; N \cdot \frac{a}{b}.$$

The latter equality, together with the second part of Claim 15, implies $n_j = 2^j$ for all $j = 1 \ldots k$. As a consequence, the initial value N of x is, due to the second part of Proposition 13, divisible by $M = (b_k)^{2^k}$. As $1 < \frac{a_k}{b_k} < 2$, we have $b_k \geq 2$, and hence M is doubly exponential with respect to k. It follows that the length of the run is also doubly exponential, as the first inner loop, in the first iteration of the **for** macro ($i = k$), is necessarily executed $N - 1 \geq M - 1$ times. This concludes the proof of Theorem 11. ◀

6 Conclusion

Our three main results have provided non-trivial counter-examples that advance the state of the art in the challenging area of the complexity of the reachability problem for VASS (equivalently, VAS and Petri nets). We have focussed on fixed dimension, and in particular, answered a central question that had remained open since [3] and [14], namely whether reachability for flat VASS given in unary is decidable in nondeterministic logarithmic space for any fixed dimension, by establishing NP hardness in dimension 7. Two specific matters that remain unresolved by this work are: whether NP hardness of reachability for unary flat VASS is obtainable in any dimension less than 7 (and more than 2), and whether binary VASS in dimension 3 can have doubly exponential shortest reachability witnesses.

We also remark that, although it has never been made precise, there seems to be an intriguing deep connection between the still open gap from NL hardness to NP membership of reachability for unary flat 3-VASS and the still open gap from PSpace hardness to ExpSpace membership of coverability for 1-GVAS (1-VASS with pushdown) [31, 41]. Finally, we expect that the novel family of sequences of fractions developed in Section 5 will have applications beyond the result obtained here.

References

1 David Angeli, Patrick De Leenheer, and Eduardo D. Sontag. Persistence results for chemical reaction networks with time-dependent kinetics and no global conservation laws. *SIAM Journal of Applied Mathematics*, 71(1):128–146, 2011. doi:10.1137/090779401.

2 Paolo Baldan, Nicoletta Cocco, Andrea Marin, and Marta Simeoni. Petri nets for modelling metabolic pathways: a survey. *Natural Computing*, 9(4):955–989, 2010. doi:10.1007/s11047-010-9180-6.

3 Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in two-dimensional vector addition systems with states is PSPACE-complete. In *LICS*, pages 32–43. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.14.

4 Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27:1–27:26, 2011. doi:10.1145/1970398.1970403.

5 Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3):13:1–13:48, 2009. doi:10.1145/1516512.1516515.

6 Ahmed Bouajjani and Michael Emmi. Analysis of recursively parallel programs. *ACM Trans. Program. Lang. Syst.*, 35(3):10:1–10:49, 2013. doi:10.1145/2518188.

7 Frank P. Burns, Albert Koelmans, and Alexandre Yakovlev. WCET analysis of superscalar processors using simulation with coloured Petri nets. *Real-Time Systems*, 18(2/3):275–288, 2000. doi:10.1023/A:1008101416758.

8 Thomas Colcombet and Amaldev Manuel. Generalized data automata and fixpoint logic. In *FSTTCS*, volume 29 of *LIPIcs*, pages 267–278. Schloss Dagstuhl, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.267.

9 Hubert Comon and Véronique Cortier. Flatness is not a weakness. In *CSL*, volume 1862 of *LNCS*, pages 262–276. Springer, 2000. doi:10.1007/3-540-44622-2_17.

10 Stefano Crespi-Reghizzi and Dino Mandrioli. Petri nets and Szilard languages. *Information and Control*, 33(2):177–192, 1977. doi:10.1016/S0019-9958(77)90558-7.

11 Wojciech Czerwiński, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. In *STOC*, pages 24–33. ACM, 2019. doi:10.1145/3313276.3316369.

12 Normann Decker, Peter Habermehl, Martin Leucker, and Daniel Thoma. Ordered navigation on multi-attributed data words. In *CONCUR*, volume 8704 of *LNCS*, pages 497–511. Springer, 2014. doi:10.1007/978-3-662-44584-6_34.

13 Stéphane Demri, Diego Figueira, and M. Praveen. Reasoning about data repetitions with counter systems. *Logical Methods in Computer Science*, 12(3), 2016. doi:10.2168/LMCS-12(3:1)2016.

14 Matthias Englert, Ranko Lazić, and Patrick Totzke. Reachability in two-dimensional unary vector addition systems with states is NL-complete. In *LICS*, pages 477–484. ACM, 2016. doi:10.1145/2933575.2933577.

15 Javier Esparza. Decidability and complexity of Petri net problems — an introduction. In *Lectures on Petri Nets I*, volume 1491 of *LNCS*, pages 374–428. Springer, 1998. doi:10.1007/3-540-65306-6_20.

16 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Inf.*, 54(2):191–215, 2017. doi:10.1007/s00236-016-0272-3.

17 Laurent Fribourg and Hans Olsén. Proving safety properties of infinite state systems by compilation into Presburger arithmetic. In *CONCUR*, volume 1243 of *LNCS*, pages 213–227. Springer, 1997. doi:10.1007/3-540-63141-0_15.

18 Pierre Ganty and Rupak Majumdar. Algorithmic verification of asynchronous programs. *ACM Trans. Program. Lang. Syst.*, 34(1):6:1–6:48, 2012. doi:10.1145/2160910.2160915.

19 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992. doi:10.1145/146637.146681.

20 Sheila A. Greibach. Remarks on blind and partially blind one-way multicounter machines. *Theor. Comput. Sci.*, 7:311–324, 1978. doi:10.1016/0304-3975(78)90020-8.

21 Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In *CONCUR*, volume 5710 of *LNCS*, pages 369–383. Springer, 2009. doi:10.1007/978-3-642-04081-8_25.

22 Michel Hack. The recursive equivalence of the reachability problem and the liveness problem for Petri nets and vector addition systems. In *SWAT*, pages 156–164. IEEE Computer Society, 1974. doi:10.1109/SWAT.1974.28.

23 Piotr Hofman and Sławomir Lasota. Linear equations with ordered data. In *CONCUR*, volume 118 of *LIPIcs*, pages 24:1–24:17. Schloss Dagstuhl, 2018. doi:10.4230/LIPIcs.CONCUR.2018.24.

24 John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8:135–159, 1979. doi:10.1016/0304-3975(79)90041-0.

25 Alexander Kaiser, Daniel Kroening, and Thomas Wahl. A widening approach to multithreaded program verification. *ACM Trans. Program. Lang. Syst.*, 36(4):14:1–14:29, 2014. doi:10.1145/2629608.

26 Max I. Kanovich. Petri nets, Horn programs, linear logic and vector games. *Ann. Pure Appl. Logic*, 75(1–2):107–135, 1995. doi:10.1016/0168-0072(94)00060-G.

27 Richard M. Karp and Raymond E. Miller. Parallel program schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969. doi:10.1016/S0022-0000(69)80011-5.

28 Hélène Leroux, David Andreu, and Karen Godary-Dejean. Handling exceptions in Petri net-based digital architecture: From formalism to implementation on FPGAs. *IEEE Trans. Industrial Informatics*, 11(4):897–906, 2015. doi:10.1109/TII.2015.2435696.

29 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *LICS*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785796.

30 Jérôme Leroux and Grégoire Sutre. On flatness for 2-dimensional vector addition systems with states. In *CONCUR*, volume 3170 of *LNCS*, pages 402–416. Springer, 2004. doi:10.1007/978-3-540-28644-8_26.

31 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On the coverability problem for pushdown vector addition systems in one dimension. In *ICALP, Part II*, volume 9135 of *LNCS*, pages 324–336. Springer, 2015. doi:10.1007/978-3-662-47666-6_26.

32 Yuliang Li, Alin Deutsch, and Victor Vianu. VERIFAS: A practical verifier for artifact systems. *PVLDB*, 11(3):283–296, 2017. URL: http://www.vldb.org/pvldb/vol11/p283-li.pdf, doi:10.14778/3157794.3157798.

33 Richard J. Lipton. The reachability problem requires exponential space. Technical Report 62, Yale University, 1976. URL: http://cpsc.yale.edu/sites/default/files/files/tr63.pdf.

34 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984. doi:10.1137/0213029.

35 Roland Meyer. A theory of structural stationarity in the *pi*-calculus. *Acta Inf.*, 46(2):87–137, 2009. doi:10.1007/s00236-009-0091-x.

36 Mor Peleg, Daniel L. Rubin, and Russ B. Altman. Research paper: Using Petri net tools to study properties and dynamics of biological systems. *JAMIA*, 12(2):181–199, 2005. doi:10.1197/jamia.M1637.

37 Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, Universität Hamburg, 1962. URL: http://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/.

38 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

39 Louis E. Rosier and Hsu-Chun Yen. A multiparameter analysis of the boundedness problem for vector addition systems. *J. Comput. Syst. Sci.*, 32(1):105–135, 1986. doi:10.1016/0022-0000(86)90006-1.

40 Sylvain Schmitz. The complexity of reachability in vector addition systems. *SIGLOG News*, 3(1):4–21, 2016. doi:10.1145/2893582.2893585.

41 Juliusz Straszyński. Complexity of the reachability problem for pushdown Petri nets. Master's thesis, University of Warsaw, Faculty of Mathematics, Informatics, and Mechanics, 2017. URL: https://apd.uw.edu.pl/diplomas/155747.

42 Leslie G. Valiant and Mike Paterson. Deterministic one-counter automata. *J. Comput. Syst. Sci.*, 10(3):340–350, 1975. doi:10.1016/S0022-0000(75)80005-5.

43 Wil M. P. van der Aalst. Business process management as the "killer app" for Petri nets. *Software and System Modeling*, 14(2):685–691, 2015. doi:10.1007/s10270-014-0424-2.

A Missing proofs

Proof of Theorem 8. The size of $\mathcal{P}$ is polynomial in n, k and m and it can be computed in time polynomial with respect to the size of the input: s_0, $S = \{s_1, \ldots, s_k\}$. $\mathcal{P}$ represents a flat VASS since its explicit **goto** commands form a directed acyclic graph, and **loop** macros are not nested. We prove that $\mathcal{P}$ has a halting run if, and only if, the instance $\{s_0\}, \{s_1, \ldots, s_k\}$ of the subset problem is positive.

($\Longleftarrow$) Fix a subset $R \subseteq S$ with $\sum_{s \in R} s = s_0$. We define a halting run ρ that starts (cf. Claim 10) by executing $\mathcal{I}'$ to compute $N(n)$ and $N(n) \cdot (k+1)$ in counters $\mathtt{e}$ and $\mathtt{f}$, respectively, and 0 in the remaining counters $\mathtt{x}, \mathtt{y}$ and $\mathtt{z}$. Then $\mathcal{R}^{+}_{s_0,\mathbf{true}}$ is executed, and finally for every $1 \leq i \leq k$, if $i \in R$ then ρ jumps to $\mathcal{R}^{-}_{s_i,\mathbf{true}}$, otherwise ρ jumps to $\mathcal{R}^{-}_{s_i,\mathbf{false}}$. Inside every component $\mathcal{R}^{*}_{s_i,p}$ the run ρ *iterates all loops maximally*, by which we mean:

- the value of $\mathtt{v}$ is 0 at the exit of the loops in lines 3–7 and in lines 10–13;
- the value of $\mathtt{v}'$ is 0 at the exit of the loop in lines 8–9;
- the value of $\mathtt{e}'$ is 0 at the exit of the loop in lines 14–15.

It remains to observe that by iterating all loops maximally, in every component $\mathcal{R}^{*}_{s_i,p}$, for $0 \leq i \leq k$, the counter $\mathtt{f}$ will be decremented by exactly $N(n)$, and thus the value of $\mathtt{f}$ at the end of ρ is zero. Moreover, $\mathcal{R}^{+}_{s_0,\mathbf{true}}$ sets the counter $\mathtt{u}$ to s_0, and for every $s_i \notin R$ the value

of counter u is preserved by $\mathcal{R}^-_{s_i,\mathbf{false}}$, and for every $s_i \in R$ the counter u is decremented by s_i in $\mathcal{R}^-_{s_i,\mathbf{true}}$. Thus the value of the counter u is 0 at the end of ρ, as well as the values of y and f, as required by the final **halt**.

($\Longrightarrow$) Consider a halting run ρ of $\mathcal{P}$, and recall that after $\mathcal{I}'$ the counter y is not modified any more, and zero-tested by the final **halt** command of $\mathcal{P}$. By Claim 10 the values of e and f after $\mathcal{I}'$ are $N(n)$ and $N(n) \cdot (k+1)$, respectively.

The sum of counters e and e' is invariantly equal $N(n)$ as decrement of one is always accompanied by increment of the other. Thus in every component $\mathcal{R}^*_{a,p}$ visited by ρ, the counter f is decreased by at most the initial value of e, hence by at most $N(n)$. Finally, by construction of $\mathcal{P}$ the run ρ passes through exactly $k+1$ components $\mathcal{R}^*_{a,p}$. Therefore, as f is zero-tested by the final **halt** command, we deduce.

▷ Claim 16. The run ρ decreases f by exactly $N(n)$ in every visited component $\mathcal{R}^*_{a,p}$.

In consequence, the initial values of component $\mathcal{R}^*_{s_i,p}$, for $0 \leq i \leq k$, satisfy:

$$\mathsf{e} = N(n) \qquad \mathsf{f} = N(n) \cdot (k+1-i) \qquad \mathsf{v} = \mathsf{v}' = \mathsf{e}' = 0.$$

Using Claim 16 we deduce.

▷ Claim 17. The run ρ iterates all loops maximally in every visited component $\mathcal{R}^*_{a,p}$, except possibly the last loop in line (15) in the last two components $\mathcal{R}^*_{s_k,p}$.

Possible non-maximal iteration of the last loop in $\mathcal{R}^*_{s_k,\mathbf{true}}$ and $\mathcal{R}^*_{s_k,\mathbf{false}}$ has no impact on the further analysis of the run ρ. As a direct corollary we deduce:

▷ Claim 18. The run ρ executes the command $\mathsf{u} \mathrel{*}= 1$ exactly a times in every visited component $\mathcal{R}^*_{a,\mathbf{true}}$.

Therefore, the value of u is incremented by s_0 in component $\mathcal{R}^+_{s_0,\mathbf{true}}$. Let $R \subseteq \{s_1, \ldots, s_k\}$ be the set of all s_i such that ρ passes through $\mathcal{R}^-_{s_i,\mathbf{true}}$. Again by Claim 18, for every $s_i \in R$ the value of u is decreased by s_i in component $\mathcal{R}^-_{s_i,\mathbf{true}}$, and for every $s_i \notin R$ the value of u is preserved in component $\mathcal{R}^-_{s_i,\mathbf{false}}$. Since u is zero-tested by the final **halt** command, the instance of the Subset Sum problem is necessarily positive. ◀

Bounded Reachability Problems Are Decidable in FIFO Machines

Benedikt Bollig
LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France
benedikt.bollig@ens-paris-saclay.fr

Alain Finkel
LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France
alain.finkel@ens-paris-saclay.fr

Amrita Suresh
LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France
amrita.suresh@ens-paris-saclay.fr

Abstract

The undecidability of basic decision problems for general FIFO machines such as reachability and unboundedness is well-known. In this paper, we provide an underapproximation for the general model by considering only runs that are input-bounded (i.e. the sequence of messages sent through a particular channel belongs to a given bounded language). We prove, by reducing this model to a counter machine with restricted zero tests, that the rational-reachability problem (and by extension, control-state reachability, unboundedness, deadlock, etc.) is decidable. This class of machines subsumes input-letter-bounded machines, flat machines, linear FIFO nets, and monogeneous machines, for which some of these problems were already shown to be decidable. These theoretical results can form the foundations to build a tool to verify general FIFO machines based on the analysis of input-bounded machines.

2012 ACM Subject Classification Theory of computation

Keywords and phrases FIFO machines, reachability, underapproximation, counter machines

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.49

Related Version A full version of the paper is available at `https://hal.archives-ouvertes.fr/hal-02900813`.

1 Introduction

Context. Asynchronous distributed processes communicating using First In First Out (FIFO) channels are being widely used for distributed and concurrent programming, and more recently, for web service choreographies. Since systems of processes communicating through (at least two) one-directional FIFO channels, or equivalently, machines having a unique control-structure with a single FIFO channel (acting as a buffer) simulate Turing machines, most properties, such as unboundedness of a channel, are undecidable for such systems [31, 6, 30].

Reachability in FIFO machines. If one restricts to runs with B-bounded channels (the number of messages in every channel does not exceed B), then reachability becomes decidable for existentially-bounded and universally-bounded FIFO systems [20]. When limiting the number of phases, the bounded-context reachability problem is in 2-EXPTIME, even for recursive FIFO systems [27, 24]. For non-confluent topology, reachability is in EXPTIME for recursive FIFO systems with 1-bounded channels [24]. The notion of k-synchronous computations was introduced in [5]. Reachability under this restriction and checking k-synchronizability are both PSPACE-complete [22]. Reachability is in PTIME in half-duplex

systems [7] with two processes (moreover, the reachability set is recognizable and effectively computable), but the natural extension to three processes leads to undecidability. Lossy FIFO systems (where the channels can lose messages) [1, 16] have been shown to be well-structured and have a decidable (but non-elementary) reachability problem [9]. In [28, 2], uniform criteria for decidability of reachability and model-checking questions are established for communicating recursive systems whose restricted architecture or communication mechanism gives rise to behaviours of bounded tree-width.

Input-bounded FIFO machines. Many papers, starting in the 80s until today, have studied FIFO machines in which the input-language of a channel (i.e. the set of words that record the messages entering a channel) is included in the set $\mathit{Pref}(w_1^* w_2^* \dots w_n^*)$ of prefixes of a bounded language $w_1^* w_2^* \dots w_n^*$. We call this class of FIFO machines *input-bounded*.

If the *set of letters* that may enter a channel c is reduced to a unique letter a_c, then the input-language of c is included in a_c^* and this subclass trivially reduces to VASS and Petri nets [32]. Also note that, in general, the behaviour of those FIFO machines does not have bounded tree-width. *Monogeneous* FIFO nets [15, 30, 19] (input-languages of channels c are included in $LF(u_c v_c^*)$ where u_c, v_c are two words associated with c) and *linear* FIFO nets [17] (input-languages are included in $\mathit{Pref}(a_1^* a_2^* \dots a_n^*)$ where each a_i is a letter and $a_i \neq a_j$ iff $i \neq j$) both generalize Petri nets with still a decidable reachability problem. A variant of the reachability problem, the deadlock problem, is shown decidable for input-*letter*-bounded FIFO systems in [23] by reducing to reachability for VASS, but the extension to general input-bounded machines was left open.

Flat machines are another subclass of input-bounded machines in which the language of their control-graph, considered as a finite automaton, is a bounded language. For flat FIFO machines, control-state reachability is NP-complete [14]; this result has recently been extended to reachability, channel unboundedness, and other classical properties [18].

To the best of our knowledge, the decidability status of control-state reachability, reachability, deadlock, and termination was not known for input-bounded FIFO machines, which strictly include all the classes discussed above such as flat, input-letter-bounded, monogeneous, and linear FIFO machines (the last three types contain VASS and they are all incomparable). The unboundedness problem of input-bounded FIFO machines was shown decidable in [26] by using the well-structured concepts but with no extension to decidability of reachability.

Our contributions:

- We solve a problem that was left open in [23], the decidability of the reachability problem for input-bounded FIFO machines. We present a simulation of input-bounded FIFO machines by counter machines with restricted zero tests. The main idea is to associate a counter with each word in the bounded language, and to ensure that the counters are incremented and decremented in a way that corresponds to the FIFO order. Since we can have repeated letters, and ambiguities in the FIFO machine, we first need to construct a normal form of the FIFO machine. Furthermore, we ensure that for every run in the FIFO machine, we can construct an equivalent run in the counter machine and vice-versa.
- As we actually solve the general rational-reachability problem, we can deduce the decidability of other verification properties like control-state reachability, deadlock, unboundedness, and termination.
- We unify various definitions from the literature, survey the (not well-known) results, and generalize them.

- Following the bounded verification paradigm, applied to FIFO machines (for instance in [14, 18]), we open the way to a methodology that would apply existing results on input-bounded FIFO machines to general FIFO machines.

Plan. In Section 2, we present counter and FIFO machines, with the connection-deconnection protocol as an example. Section 3 contains the main result, which states the decidability of rational-reachability for FIFO machines restricted to input-bounded languages. Section 4 considers variants of the reachability problem such as unboundedness and termination. Finally, in Section 5, we mention further results, state some open problems, and discuss a possible theory of boundable FIFO machines. Missing proofs can be found in the long version of the paper, available at: `https://hal.archives-ouvertes.fr/hal-02900813`

2 Preliminaries

Words and Languages. Let A be a finite alphabet. As usual, A^* is the set of finite words over A, and A^+ the set of non-empty finite words. We let $|w|$ denote the length of $w \in A^*$. For the empty word ε, we have $|\varepsilon| = 0$. Given $a \in A$, let $|w|_a$ denote the number of occurrences of a in w. With this, we let $\mathit{Alph}(w) = \{a \in A \mid |w|_a \geq 1\}$. The concatenation of two words $u, v \in A^*$ is denoted by $u \cdot v$ or $u.v$ or simply uv. The sets of prefixes, suffixes, and infixes of $w \in A^*$ are denoted by $\mathit{Pref}(w)$, $\mathit{Suf}(w)$, and $\mathit{Infix}(w)$, resp. Note that $\{\varepsilon, w\} \subseteq \mathit{Pref}(w) \cap \mathit{Suf}(w) \cap \mathit{Infix}(w)$. For a set X, any mapping $f : A^* \to 2^X$ can be extended to $f : 2^{A^*} \to 2^X$ letting, for $L \subseteq A^*$, $f(L) = \bigcup_{w \in L} f(w)$. In particular, Alph, Pref, Suf, and Infix are extended in that way.

▶ **Definition 1** ([21]). *Let $w_1, \ldots, w_n \in A^+$ be non-empty words where $n \geq 1$. A* bounded language *over $(w_1, \ldots, w_n)$ is a language $L \subseteq w_1^* \ldots w_n^*$.*

We always assume that a bounded language L is given together with its tuple $(w_1, \ldots, w_n)$ and that $\mathit{Alph}(L) = \mathit{Alph}(w_1 \ldots w_n)$. We say that L is *distinct-letter* if $|w_1 \ldots w_n|_a \leq 1$ for all $a \in A$. If $|w_1| = \ldots = |w_n| = 1$, i.e. $w_1, \ldots, w_n \in A$, then L is a *letter-bounded language*. Let us remark that the set of bounded languages is closed under Pref and Suf.

Semi-Linear Sets. A *linear* set X (of dimension $d \geq 1$) is defined as a subset of $\mathbb{N}^d$ for which there exist a basis $\mathbf{b} \in \mathbb{N}^d$ and a finite set of periods $\{\mathbf{p}_1, \ldots, \mathbf{p}_m\} \subseteq \mathbb{N}^d$ such that $X = \{\mathbf{b} + \sum_{i=1}^m \lambda_i \mathbf{p}_i \mid \lambda_1, \ldots, \lambda_m \in \mathbb{N}\}$. A *semi-linear* set is defined as a finite union of linear sets.

Transition Systems. A *labeled transition system* is a quadruple $\mathcal{T} = (S, A, \to, \mathit{init})$ where S is the (potentially infinite) set of *configurations*[1], A is a finite alphabet, $\mathit{init} \in S$ is the *initial configuration*, and $\to \subseteq S \times A \times S$ is the *transition relation*.

For $s, s' \in S$, let $s \to s'$ if $s \xrightarrow{a} s'$ for some $a \in A$. For $w \in A^*$, we write $s \xrightarrow{w} s'$ if there is a w-labeled path from s to s'. Formally, $s \xrightarrow{\varepsilon} s'$ if $s = s'$, and $s \xrightarrow{aw} s'$ if there is $t \in S$ such that $s \xrightarrow{a} t$ and $t \xrightarrow{w} s'$. We let $\mathit{Traces}(\mathcal{T}) = \{w \in A^* \mid \mathit{init} \xrightarrow{w} s \text{ for some } s \in S\}$.

Given $w \in A^*$, we let $\mathit{Reach}_{\mathcal{T}}(w) = \{s \in S \mid \mathit{init} \xrightarrow{w} s\}$. Moreover, for $L \subseteq A^*$, $\mathit{Reach}_{\mathcal{T}}(L) = \bigcup_{w \in L} \mathit{Reach}_{\mathcal{T}}(w)$ is the set of configurations that are reachable via a word from L. Finally, the *reachability set* of $\mathcal{T}$ is defined as $\mathit{Reach}_{\mathcal{T}} = \mathit{Reach}_{\mathcal{T}}(A^*)$. We call $\mathcal{T}$ *finite* if $\mathit{Reach}_{\mathcal{T}}$ is finite (and this is the case if S is finite). Otherwise, $\mathcal{T}$ is called *infinite*.

[1] We say *configurations* rather than *states* to distinguish them from the *control states* used in FIFO and counter machines.

FIFO Machines. We consider FIFO machines having a sequential control graph rather than systems of communicating processes that are distributed systems. It is clear that, given a distributed system, one may compute the Cartesian product of all processes to obtain a FIFO machine (the converse is not always true).

▶ **Definition 2.** *A* FIFO machine *is a tuple* $M = (Q, Ch, \Sigma, T, q_0)$ *where* Q *is a finite set of* control states*,* $q_0 \in Q$ *is an* initial control state*, and* Ch *is a finite set of* channels*. Moreover,* Σ *is a finite* message alphabet*. It is partitioned into* $\Sigma = \biguplus_{c \in Ch} \Sigma_c$ *where* Σ_c *contains the messages that can be sent through channel* c*. Finally,* $T \subseteq Q \times A_M \times Q$ *is a* transition relation *where* $A_M = \{\langle c!a \rangle \mid c \in Ch \text{ and } a \in \Sigma_c\} \cup \{\langle c?a \rangle \mid c \in Ch \text{ and } a \in \Sigma_c\}$ *is the set of send and receive actions.*

▶ **Example 3** (Connection-Deconnection Protocol)**.** A model for the (simplified) connection-deconnection protocol, CDP, between two processes is described as follows (see Figure 1): We model the protocol with two automata (representing the two processes) and two (infinite) channels. The first processes (on the left) can open a session (this is denoted by sending the message "a" through channel c_1 to the other process). Once a session is open, the first process can close it (by sending message "b" to the other process), or on the demand of the second process (if it receives the message "e"). This protocol has been studied in [25].

In the example, it is natural to have two separate processes. However, following Definition 2, we formalize this in terms of the Cartesian product of the two processes. That is, the CDP is modeled as the FIFO machine $M = (Q, Ch, \Sigma, T, q_0)$ where $Q = \{0, 1\} \times \{0, 1\}$ (the Cartesian product of the local state spaces) with initial state $q_0 = (0, 0)$, $Ch = \{c_1, c_2\}$, $\Sigma = \Sigma_{c_1} \uplus \Sigma_{c_2}$ with $\Sigma_{c_1} = \{a, b\}$ and $\Sigma_{c_2} = \{e\}$. Moreover, the transition relation T contains, amongst others, $((0, 0), \langle c_1!a \rangle, (1, 0))$ and $((1, 0), \langle c_1?a \rangle, (1, 1))$. ⌟

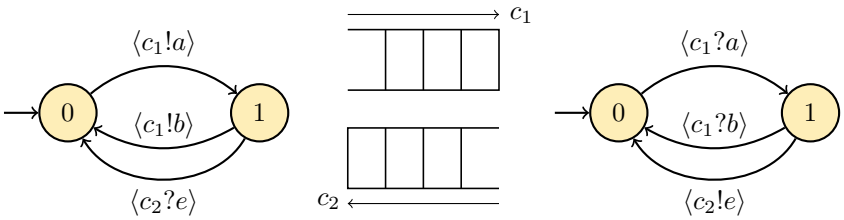


Figure 1 The model of the connection-deconnection protocol.

A FIFO machine $M = (Q, Ch, \Sigma, T, q_0)$ induces a (potentially infinite) transition system $\mathcal{T}_M = (S_M, A_M, \to_M, init_M)$. Its set of configurations is $S_M = Q \times \prod_{c \in Ch} \Sigma_c^*$. In $(q, \mathbf{w}) \in S_M$, the first component q denotes the current control state and $\mathbf{w} = (\mathbf{w}_c)_{c \in Ch}$ determines the contents $\mathbf{w}_c \in \Sigma_c^*$ for every channel $c \in Ch$. The initial configuration is $init_M = (q_0, \boldsymbol{\varepsilon})$ where $\boldsymbol{\varepsilon} = (\varepsilon, \ldots, \varepsilon)$, i.e., every channel is empty. The transitions are given as follows:

- $(q, \mathbf{w}) \xrightarrow{\langle c!a \rangle}_M (q', \mathbf{w}')$ if $(q, \langle c!a \rangle, q') \in T$, $\mathbf{w}'_c = \mathbf{w}_c \cdot a$, and $\mathbf{w}'_d = \mathbf{w}_d$ for all $d \in Ch \setminus \{c\}$;
- $(q, \mathbf{w}) \xrightarrow{\langle c?a \rangle}_M (q', \mathbf{w}')$ if $(q, \langle c?a \rangle, q') \in T$, $\mathbf{w}_c = a \cdot \mathbf{w}'_c$, and $\mathbf{w}'_d = \mathbf{w}_d$ for all $d \in Ch \setminus \{c\}$.

The index M may be omitted whenever M is clear from the context.

The *reachability set* of M is defined as the reachability set of $\mathcal{T}_M$, i.e., $Reach_M = Reach_{\mathcal{T}_M}$ and, for $L \subseteq A_M^*$, $Reach_M(L) = Reach_{\mathcal{T}_M}(L)$. Moreover, we let $Traces(M) = Traces(\mathcal{T}_M)$.

▶ **Example 4.** An example run of the FIFO machine M from Example 3 and Figure 1 is $((0, 0), (\varepsilon, \varepsilon)) \xrightarrow{\langle c_1!a \rangle} ((1, 0), (a, \varepsilon)) \xrightarrow{\langle c_1?a \rangle} ((1, 1), (\varepsilon, \varepsilon)) \xrightarrow{\langle c_2!e \rangle} ((1, 0), (\varepsilon, e))$. As for the reachability set, we have, e.g., $((1, 1), ((ba)^*, \varepsilon))) \subseteq Reach_M$ and $((0, 0), (b(ab)^*, e)) \subseteq Reach_M$.

Let us remark that CDP is not half-duplex because there are reachable configurations with both channels non-empty, e.g., $((0,0),(b,e))$; moreover, it is neither monogeneous, nor linear, nor input-letter-bounded. ⌟

Counter Machines. We next recall the notion of counter machines, where multiple counters can take non-negative integer values, be incremented and decremented, and be tested for zero (though in a restricted fashion).

▶ **Definition 5.** *A* counter machine *(with zero tests) is a tuple* $\mathcal{C} = (Q, Cnt, T, q_0)$*. Like in a FIFO machine,* Q *is the finite set of* control states *and* $q_0 \in Q$ *is the* initial control state*. Moreover,* Cnt *is a finite set of* counters *and* $T \subseteq Q \times A_{\mathcal{C}} \times Q$ *is the transition relation where* $A_{\mathcal{C}} = \{\mathsf{inc}(x), \mathsf{dec}(x) \mid x \in Cnt\} \times 2^{Cnt}$*.*

The counter machine $\mathcal{C}$ induces a transition system $\mathcal{T}_{\mathcal{C}} = (S_{\mathcal{C}}, A_{\mathcal{C}}, \to_{\mathcal{C}}, init_{\mathcal{C}})$ with set of configurations $S_{\mathcal{C}} = Q \times \mathbb{N}^{Cnt}$. In $(q, \mathbf{v}) \in S_{\mathcal{C}}$, q is the current control state and $\mathbf{v} = (\mathbf{v}_x)_{x \in Cnt}$ represents the counter values. The initial configuration is $init_{\mathcal{C}} = (q_0, \mathbf{0})$ where $\mathbf{0}$ maps all counters to 0. For $op \in \{\mathsf{inc}, \mathsf{dec}\}$, $x \in Cnt$, and $Z \subseteq Cnt$ (the counters tested for zero), there is a transition $(q, \mathbf{v}) \xrightarrow{(op(x),Z)}_{\mathcal{C}} (q', \mathbf{v}')$ if $(q, (op(x), Z), q') \in T$, $\mathbf{v}_y = 0$ for all $y \in Z$ (applies the zero tests), $\mathbf{v}'_x = \mathbf{v}_x + 1$ if $op = \mathsf{inc}$ and $\mathbf{v}'_x = \mathbf{v}_x - 1$ if $op = \mathsf{dec}$, and $\mathbf{v}'_y = \mathbf{v}_y$ for all $y \in Cnt \setminus \{x\}$.

The *reachability set* of $\mathcal{C}$ is defined as $Reach_{\mathcal{C}} = Reach_{\mathcal{T}_{\mathcal{C}}}$. For $L \subseteq A_{\mathcal{C}}^*$, we also let $Reach_{\mathcal{C}}(L) = Reach_{\mathcal{T}_{\mathcal{C}}}(L)$. Moreover, $Traces(\mathcal{C}) = Traces(\mathcal{T}_{\mathcal{C}})$. To get decidability of reachability in counter machines, we impose the restriction that, once a counter has been tested for zero, it cannot be incremented or decremented anymore. This is clearly an extension of VASS. To define this, let $L_{\mathcal{C}}^{\mathsf{zero}}$ be the set of words $(op_1(x_1), Z_1) \dots (op_n(x_n), Z_n) \in A_{\mathcal{C}}^*$ such that, for every two positions $1 \le i \le j \le n$, we have $x_j \notin Z_i$.

▶ **Theorem 6.** *The following problem is decidable (though inherently non-elementary): Given a counter machine* $\mathcal{C} = (Q, Cnt, T, q_0)$*, a regular language* $L \subseteq A_{\mathcal{C}}^*$*, a control state* $q \in Q$*, and a semi-linear set* $V \subseteq \mathbb{N}^{Cnt}$*, do we have* $(q, \mathbf{v}) \in Reach_{\mathcal{C}}(L_{\mathcal{C}}^{\mathsf{zero}} \cap L)$ *for some* $\mathbf{v} \in V$*?*

Proof sketch. Reachability in presence of a semi-linear target set and restricted zero tests straightforwardly reduces to configuration-reachability in counter machines without zero tests (i.e., VASS and Petri nets). The latter is decidable [29], though inherently non-elementary [11]. First, zero tests are postponed to the very end of an execution and, to this aim, stored in the control-state. Second, to check whether a counter valuation is contained in V, we can branch, whenever we are in the given control-state q, into a new component that decrements counters accordingly and eventually checks whether they are all zero. ◀

3 The Input-Bounded Rational-Reachability Problem

It is very well known that the following reachability problem is undecidable: Given a FIFO machine $M = (Q, Ch, \Sigma, T, q_0)$, a configuration $(q, \mathbf{w}) \in S_M$, and a regular language $L \subseteq A_M^*$, do we have $(q, \mathbf{w}) \in Reach_M(L)$? Of course, the problem is already undecidable when we impose $L = A_M^*$. Motivated by this negative result, we are looking for language classes $\mathfrak{C}$ that render the problem decidable under the restriction that $L \in \mathfrak{C}$.

We say that a FIFO machine $M = (Q, Ch, \Sigma, T, q_0)$ has a *bounded reachability set* if there is a tuple $(L_c)_{c \in Ch}$ of regular bounded languages $L_c \subseteq \Sigma_c^*$ such that, for all $(q, \mathbf{w}) \in Reach_M$, we have $\mathbf{w} \in \prod_{c \in Ch} L_c$. We observe that restricting the reachability set to be bounded is not sufficient to obtain a decidable reachability problem. We show this by simulating any two counter Minsky machine by a FIFO machine with fixed languages L_c.

▶ **Theorem 7.** *The reachability problem is undecidable for FIFO machines with a (given) bounded reachability set.*

We therefore consider a different restriction to obtain decidability. For a given FIFO machine $M = (Q, Ch, \Sigma, T, q_0)$, we are interested in $Reach_M(L)$ where $L \subseteq A_M^*$ is *input-bounded* in the following sense: For every channel c, the sequence of messages that are sent through channel c is from a given regular bounded language $L_c \subseteq \Sigma_c^*$.

Let us be more formal. For $c \in Ch$, we let $proj_{c!} : A_M^* \to \Sigma_c^*$ be the homomorphism defined by $proj_{c!}(\langle c!a\rangle) = a$ for all $a \in \Sigma_c$, and $proj_{c!}(\beta) = \varepsilon$ if $\beta \in A_M$ is not of the form $\langle c!a\rangle$ for some $a \in \Sigma_c$. We define $proj_{c?} : A_M^* \to \Sigma_c^*$ accordingly.

With this, given a tuple $\mathscr{L} = (L_c)_{c\in Ch}$ of bounded languages $L_c \subseteq \Sigma_c^*$, we set $\mathscr{L}_! = \{\sigma \in A_M^* \mid proj_{c!}(\sigma) \in L_c \text{ for all } c \in Ch\}$ and $\mathscr{L}_? = \{\sigma \in A_M^* \mid proj_{c?}(\sigma) \in L_c \text{ for all } c \in Ch\}$. We observe that, if all L_c are regular, then so are $\mathscr{L}_!$ and $\mathscr{L}_?$.

▶ **Definition 8.** *The* input-bounded (IB) reachability problem *asks whether a given configuration* $(q, \mathbf{w})$ *is reachable along a sequence of actions from* $\mathscr{L}_!$*, i.e., whether* $(q, \mathbf{w}) \in Reach_M(\mathscr{L}_!)$.

Note that, if $(q_0, \boldsymbol{\varepsilon}) \xrightarrow{\sigma}_M (q, \mathbf{w})$ and $\sigma \in \mathscr{L}_!$, then we also have $\sigma \in Pref(\mathscr{L}_?)$ due to the FIFO policy. Thus, $Reach_M(\mathscr{L}_!) = Reach_M(\mathscr{L}_! \cap Pref(\mathscr{L}_?))$ so that we can restrict to action sequences from $\mathscr{L}_! \cap Pref(\mathscr{L}_?)$. We will call $\mathscr{L}_! \cap Pref(\mathscr{L}_?)$ the set of *valid* words.

▶ **Example 9.** Let us come back to the protocol CDP M from Example 3 and Figure 1, which is neither monogeneous nor linear nor flat. Since the "input-languages" of the two channels (i.e. the languages of words that record the messages entering a channel) contain $\{a, ab\}^*$ and e^*, resp., and since $\{a, ab\}^*$ is not a bounded language, we have $Traces(M) \not\subseteq \mathscr{L}_!$ for every pair of bounded languages $\mathscr{L}$. In other words, M is not input-bounded. However, when we look at the reachability set obtained by considering the tuple of bounded languages $\mathscr{L} = (L_{c_1}, L_{c_2})$ where $L_{c_1} = (ab)^*(a + \varepsilon)(ab)^*$ is a bounded language over (ab, a, ab), and $L_{c_2} = e^*$ is a bounded language over (e), we still obtain the entire reachability set. That is, we have $Reach_M = Reach_M(\mathscr{L}_!)$. Hence, even though the input-languages of the system are not all bounded, we can still compute the reachability set by restricting our exploration to a tuple of (regular) bounded languages $\mathscr{L}$. ⌟

Actually, instead of reachability of a single configuration as stated in Definition 8, we study a more general problem, called the *input-bounded rational-reachability problem*. It asks whether a configuration $(q, \mathbf{w})$ is reachable for some channel contents $\mathbf{w}$ from a given *rational* relation. So let us define rational relations.

Rational and Recognizable Relations. Consider a relation $\mathscr{R} \subseteq \prod_{c\in Ch} \Sigma_c^*$. We say that $\mathscr{R}$ is *rational* if there is a regular word language $R \subseteq \Theta^*$ over the alphabet $\Theta = \prod_{c\in Ch}(\Sigma_c \cup \{\varepsilon\})$ such that $\mathscr{R} = \{(\mathbf{a}_c^1 \cdot \ldots \cdot \mathbf{a}_c^n)_{c\in Ch} \mid \mathbf{a}^1 \ldots \mathbf{a}^n \in R \text{ with } n \in \mathbb{N} \text{ and } \mathbf{a}^i = (\mathbf{a}_c^i)_{c\in Ch} \in \Theta \text{ for } i \in \{1, \ldots, n\}\}$. Here, $\mathbf{a}_c^1 \cdot \ldots \cdot \mathbf{a}_c^n \in \Sigma_c^*$ is the concatenation of all $\mathbf{a}_c^i \in \Sigma_c \cup \{\varepsilon\}$ while ignoring the neutral element ε. For example, in the presence of two channels, $\mathscr{R} = \{(a^m, b^n) \mid m \geq n\}$ is a rational relation, witnessed by $R = ((a, b) + (a, \varepsilon))^*$. In the following, we will always assume that a rational relation is given in terms of a finite automaton for the underlying regular language R.

A relation $\mathscr{R} \subseteq \prod_{c\in Ch} \Sigma_c^*$ is called *recognizable* if it is the finite union of relations of the form $\prod_{c\in Ch} R_c$ where all $R_c \subseteq \Sigma_c^*$ are regular languages. Note that every recognizable relation is rational while the converse is, in general, false.

We define the *Parikh image* of a relation $\mathscr{R} \subseteq \prod_{c \in Ch} \Sigma_c^*$ as $\mathrm{Parikh}(\mathscr{R}) = \{(\pi_a)_{a \in \Sigma} \in \mathbb{N}^\Sigma \mid \exists \mathbf{w} = (\mathbf{w}_c)_{c \in Ch} \in \mathscr{R} : \pi_a = |\mathbf{w}_c|_a \text{ for all } c \in Ch \text{ and } a \in \Sigma_c\}$. It is well known that, if $\mathscr{R}$ is rational, then $\mathrm{Parikh}(\mathscr{R})$ is semi-linear.

For more background on rational relations and their subclasses, we refer to [4, 10].

The IB Rational-Reachability Problem. We are now prepared to define the input-bounded (IB) rational-reachability problem and to state its decidability:

▶ **Definition 10.** *The* IB rational-reachability problem *is defined as follows: Given a FIFO machine $M = (Q, Ch, \Sigma, T, q_0)$, a tuple $\mathscr{L} = (L_c)_{c \in Ch}$ of non-empty regular bounded languages $L_c \subseteq \Sigma_c^*$ (each given in terms of a finite automaton), a control state $q \in Q$, and a rational relation $\mathscr{R} \subseteq \prod_{c \in Ch} \Sigma_c^*$. Do we have $(q, \mathbf{w}) \in Reach_M(\mathscr{L}_!)$ for some $\mathbf{w} \in \mathscr{R}$?*

▶ **Theorem 11.** *IB rational-reachability is decidable for FIFO machines.*

The remainder of this section is devoted to the proof of Theorem 11.

Let $M = (Q, Ch, \Sigma, T, q_0)$ and let $\mathscr{L} = (L_c)_{c \in Ch}$ be a tuple of non-empty regular bounded languages $L_c \subseteq \Sigma_c^*$ over $(w_{c,1}, \ldots, w_{c,n_c})$. We proceed by reduction to counter machines. The rough idea is to represent the contents of channel c in terms of several counters, one for every component $w_{c,i}$. To have a faithful simulation, we rely on a normal form of M and its bounded languages, which can be achieved at the expense of an exponential blow-up of the FIFO machine.

▶ **Definition 12.** *We say that M and $\mathscr{L}$ are in* normal form *if the following hold:*

1. *For all $c \in Ch$, $\Sigma_c \subseteq Alph(L_c)$ and L_c is distinct-letter.*
2. *We have $Traces((Q, A_M, T, q_0)) \subseteq Pref(\mathcal{V})$ where $\mathcal{V} = \mathscr{L}_! \cap Pref(\mathscr{L}_?)$. Note that (Q, A_M, T, q_0) is the finite transition system induced by the control graph of M.*

Given a FIFO machine $\hat{M} = (\hat{Q}, Ch, \hat{\Sigma}, \hat{T}, \hat{q}_0)$ and the tuple $\hat{\mathscr{L}} = (\hat{L}_c)_{c \in Ch}$ of non-empty regular bounded languages $\hat{L}_c \subseteq \hat{\Sigma}_c^*$, we now construct $M = (Q, Ch, \Sigma, T, q_0)$ and $\mathscr{L} = (L_c)_{c \in Ch}$ in normal form such that a reachability query in the former can be transformed into a reachability query in the latter (made precise in Lemma 15 below).

Distinct-Letter Property. Consider the bounded language $\hat{L}_c$ over $(\hat{w}_{c,1}, \ldots, \hat{w}_{c,n_c})$. For $i \in \{1, \ldots, n_c\}$, let $m_i = |\hat{w}_{c,1}| + \ldots + |\hat{w}_{c,i}|$ be the number of letters in the first i words. Moreover, $m = m_{n_c}$. Let Σ_c denote the alphabet $\{a_1^c, \ldots, a_m^c\}$. It contains the "distinct" letters for the bounded language L_c over $(w_{c,1}, \ldots, w_{c,n_c})$, where we let $w_{c,1} = a_1^c \ldots a_{m_1}^c$ and $w_{c,i} = a_{m_{i-1}+1}^c \ldots a_{m_i}^c$ for $i \geq 2$. In other words, the letters in $(w_{c,1}, \ldots, w_{c,n_c})$ are numbered consecutively. In order to obtain the language L_c, we first consider the homomorphism $h_c : \Sigma_c^* \to \hat{\Sigma}_c^*$ where $h_c(a_i^c)$ is the i-th letter in the word $\hat{w}_{c,1} \ldots \hat{w}_{c,n_c}$. We obtain L_c as $h_c^{-1}(\hat{L}_c) \cap (w_{c,1})^* \ldots (w_{c,n_c})^*$, hence preserving regularity and boundedness. We then remove those words from $(w_{c,1}, \ldots, w_{c,n_c})$ (and their letters from Σ_c) whose letters do not occur in L_c. We have $\Sigma_c \subseteq Alph(L_c)$.

▶ **Example 13.** For example, suppose we have one channel c and $\hat{L}_c = (ab)^*bb^*$ over (ab, b). We determine the language L_c over (a_1a_2, a_3) (omitting the superscript c in the letters). The homomorphism $h_c : \{a_1, a_2, a_3\}^* \to \{a, b\}^*$ is given by $h_c(a_1) = a$ and $h_c(a_2) = h_c(a_3) = b$. We have $h_c^{-1}(\hat{L}_c) = (a_1(a_2 + a_3))^*(a_2 + a_3)(a_2 + a_3)^*$, which we intersect with $(a_1a_2)^*a_3^*$. We thus get the regular bounded language $L_c = (a_1a_2)^*a_3a_3^*$ over (a_1a_2, a_3). All letters from $\{a_1, a_2, a_3\}$ occur in L_c so that we are done.

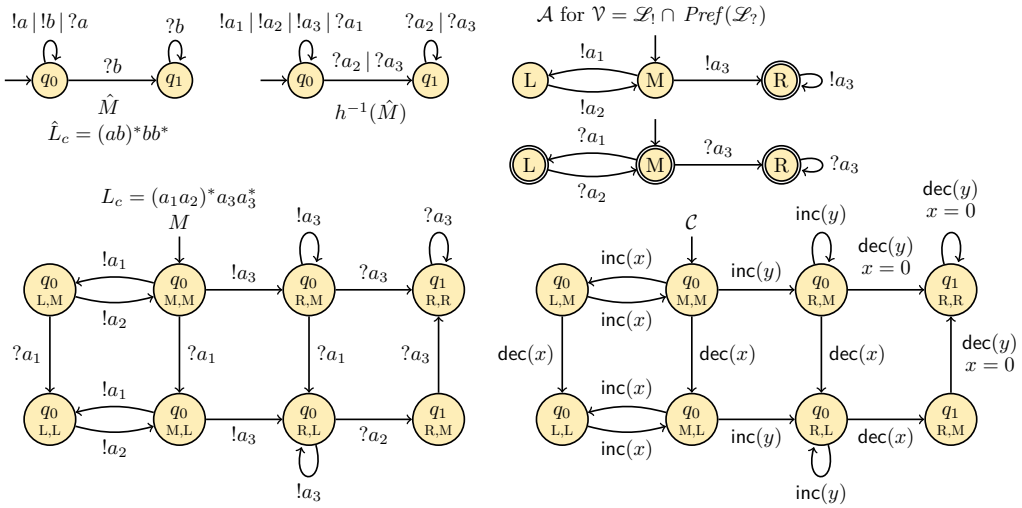


Figure 2 For a FIFO machine $\hat{M}$ with a single channel c and the bounded language $\hat{L}_c = (ab)^*bb^*$ over (ab, b) (top leftmost), we construct a FIFO machine M (bottom left), together with $L_c = (a_1a_2)^*a_3a_3^*$, in normal form as the product of $h^{-1}(\hat{M})$ and an automaton for $\mathcal{V}$ (top right). From M, we then obtain the counter machine $\mathcal{C}$ (bottom right).

Trace Property. In the next step, we build the FIFO machine $M = (Q, Ch, \Sigma, T, q_0)$ such that $Traces((Q, A_M, T, q_0)) \subseteq Pref(\mathcal{V})$ with $\mathcal{V} = \mathscr{L}_! \cap Pref(\mathscr{L}_?)$. First, to take care of the homomorphisms h_c, we define the transition relation $h^{-1}(\hat{T}) = \{(q, \langle c!e\rangle, q') \mid (q, \langle c!a\rangle, q') \in \hat{T}$ and $e \in h_c^{-1}(a)\} \cup \{(q, \langle c?e\rangle, q') \mid (q, \langle c?a\rangle, q') \in \hat{T}$ and $e \in h_c^{-1}(a)\}$. Thus, the set of actions of M will be $A_M = \{\langle c!e\rangle \mid c \in Ch$ and $e \in \Sigma_c\} \cup \{\langle c?e\rangle \mid c \in Ch$ and $e \in \Sigma_c\}$.

To continue our above example, a transition $(q, \langle c!b\rangle, q')$ would be replaced with the two transitions $(q, \langle c!a_2\rangle, q')$ and $(q, \langle c!a_3\rangle, q')$, and similarly for $(q, \langle c?b\rangle, q')$.

To guarantee trace inclusion in $Pref(\mathcal{V})$, we will consider a deterministic (not necessarily complete) finite automaton $\mathcal{A} = (Q_\mathcal{A}, A_M, T_\mathcal{A}, q^0_\mathcal{A}, F_\mathcal{A})$, with set of final states $F_\mathcal{A} \subseteq Q_\mathcal{A}$, whose language is $L(\mathcal{A}) = \mathcal{V}$ and where, from every state, a final state is reachable in the finite graph $(Q_\mathcal{A}, T_\mathcal{A})$. With this, we define M as the product of the FIFO machine $h^{-1}(\hat{M}) = (\hat{Q}, Ch, \Sigma, h^{-1}(\hat{T}), \hat{q}_0)$ and $\mathcal{A}$ in the expected manner. In particular, the set of control states of M is $\hat{Q} \times Q_\mathcal{A}$, and its initial state is the pair $(\hat{q}_0, q^0_\mathcal{A})$.

▶ **Example 14.** Figure 2 illustrates the result of the normalization procedure for a FIFO machine $\hat{M}$ with one single channel c (which is therefore omitted) and its bounded language $\hat{L}_c = (ab)^*bb^*$ over (ab, b). Recall from Example 13 that the corresponding homomorphism h_c maps a_1 to a and both a_2 and a_3 to b, and that we obtain $L_c = (a_1a_2)^*a_3a_3^*$. Moreover, M is the product of $h^{-1}(\hat{M})$ (depicted in the top center) and a finite automaton $\mathcal{A}$ for $\mathcal{V} = \mathscr{L}_! \cap Pref(\mathscr{L}_?)$ (obtained as the shuffle of the two finite automata on the top right). The state names in M reflect the states of $\hat{M}$ and $\mathcal{A}$ they originate from. We depict only accessible states of M from which we can still complete the word read so far to a word in $\mathcal{V}$. For example, $(q_1, \mathrm{M}, \mathrm{L})$ and $(q_1, \mathrm{L}, \mathrm{R})$ would no longer allow us to reach the final state R of the $\mathscr{L}_!$-component. ⌟

Now suppose we are given a reachability query for $\hat{M}$ in terms of $\hat{q} \in \hat{Q}$ and a rational relation $\hat{\mathscr{R}} \subseteq \prod_{c \in Ch} \hat{\Sigma}_c^*$. The lemma below shows how to reduce it to a reachability query in M. Here, for $\mathbf{w} = (\mathbf{w}_c)_{c \in Ch} \in \prod_{c \in Ch} \Sigma_c^*$, we define $h(\mathbf{w}) = (h_c(\mathbf{w}_c))_{c \in Ch} \in \prod_{c \in Ch} \hat{\Sigma}_c^*$. Note that $h^{-1}(\hat{\mathscr{R}})$ is rational.

▶ **Lemma 15.** *We have* $(\hat{q}, \hat{\mathbf{w}}) \in Reach_{\hat{M}}(\hat{\mathscr{L}}_!)$ *for some* $\hat{\mathbf{w}} \in \hat{\mathscr{R}}$ *iff* $((\hat{q}, q_{\mathcal{A}}), \mathbf{w}) \in Reach_M(\mathscr{L}_!)$ *for some* $q_{\mathcal{A}} \in Q_{\mathcal{A}}$ *and* $\mathbf{w} \in h^{-1}(\hat{\mathscr{R}})$.

Reduction of Normal Form to Counter Machine

Henceforth, we suppose that $M = (Q, Ch, \Sigma, T, q_0)$ and $\mathscr{L} = (L_c)_{c \in Ch}$ are in normal form, where L_c is a bounded language over $(w_{c,1}, \ldots, w_{c,n_c})$. In particular, for every letter $a \in \Sigma_c$, there is a unique index $i \in \{1, \ldots, n_c\}$ such that $a \in \Sigma_{c,i}$ where $\Sigma_{c,i} = Alph(w_{c,i})$. We denote this index i by i_a.

We build a counter machine $\mathcal{C}$ such that the IB rational-reachability problem for M can be solved by answering a reachability query in $\mathcal{C}$, using Theorem 6. Each run in $\mathcal{C}$ will simulate a run in M. In particular, we want a configuration of $\mathcal{C}$ to allow us to draw conclusions about the simulated configuration in M. The difficulty here is that counter values are just natural numbers and a priori store less information than channel contents with their messages. To overcome this, the idea is to represent each word $w_{c,i}$ of a tuple $(w_{c,1}, \ldots, w_{c,n_c})$ as a counter $x_{(c,i)}$. Since the set of possible action sequences is "guided" by a bounded language, we can replace send actions with increments and receive actions with decrements. More precisely, $\langle c!a \rangle$ becomes $(\mathsf{inc}(x_{(c,i_a)}), \emptyset)$, thus incrementing the counter associated with the unique word $w_{c,i}$ in which a occurs. Similarly, $\langle c?a \rangle$ translates to $(\mathsf{dec}(x_{(c,i_a)}), Z)$ (for suitable Z).

This alone does not put us in a position yet where, from a counter valuation, we can infer a unique channel contents. However, when we additionally keep track of the last messages that have been sent for each channel, we can reconstruct a unique channel contents.

There is one more thing to consider here. While the counters $x_{(c,i)}$ for a given channel c are kind of independent, the FIFO policy would not allow us to receive a letter from $w_{c,j}$ while a letter from $w_{c,i}$ with $i < j$ is in transit. Translated to the counter setting, this means that performing $\mathsf{dec}(x_{(c,j)})$ should require all counters $x_{(c,i)}$ with $i < j$ to be 0, so this is where zero tests come into play. As the L_c are bounded languages and thanks to the normal form, however, a counter that has been tested for zero does not need to be modified anymore.

We can directly implement these ideas formally and define $\mathcal{C} = (Q, Cnt, T', q_0)$ as follows (note that Q and q_0 remain unchanged):

- The set of counters is $Cnt = \{x_{(c,i)} \mid c \in Ch \text{ and } i \in \{1, \ldots, n_c\}\}$.
- For every $(q, \langle c!a \rangle, q') \in T$, we have $(q, (\mathsf{inc}(x_{(c,i_a)}), \emptyset), q') \in T'$.
- For every $(q, \langle c?a \rangle, q') \in T$, we have $(q, (\mathsf{dec}(x_{(c,i_a)}), Z), q') \in T'$ where the set of counters to be tested for zero is $Z = \{x_{(c,j)} \mid j < i_a\}$.

▶ **Example 16.** Figure 2 illustrates the construction of $\mathcal{C}$ from a FIFO machine M in normal form (cf. Example 14). Recall that we have one channel c and the bounded language $L_c = (a_1 a_2)^* a_3 a_3^*$ over $(a_1 a_2\,, a_3)$. Thus, $\mathcal{C}$ will have two counters, say x for $a_1 a_2$ and y for a_3. Note that performing $\mathsf{dec}(y)$ indeed comes with a test of x for zero.

Let us first observe that it is actually important that the FIFO machine satisfies the trace property. Suppose that, rather than from M, we constructed the counter machine directly from $h^{-1}(\hat{M})$. Then, configuration $(q_1, (1, 0))$ would be reachable in the counter machine via $\mathsf{inc}(x)\mathsf{inc}(x)\mathsf{dec}(x)$, which arises from $\langle c!a_1 \rangle \langle c!a_2 \rangle \langle c?a_2 \rangle$. However the only corresponding trace from $Pref(\mathcal{V})$ is $\langle c!a_1 \rangle \langle c!a_2 \rangle \langle c?a_1 \rangle$, which in the FIFO machine $h^{-1}(\hat{M})$ leads to q_0.

So consider M and its counter machine $\mathcal{C}$. A channel contents $\mathbf{w} \in \Sigma_c^*$ (here, we have one channel) has a natural counter analogue $\langle\!\langle \mathbf{w} \rangle\!\rangle = (|\mathbf{w}|_{a_1} + |\mathbf{w}|_{a_2}, |\mathbf{w}|_{a_3})$. In fact, if $(\overline{q}, \mathbf{w})$ is reachable in M, then following the corresponding transitions in $\mathcal{C}$ will lead us to $(\overline{q}, \langle\!\langle \mathbf{w} \rangle\!\rangle)$. For example, $((q_0, \mathrm{R}, \mathrm{L}), a_2 a_3 a_3)$ is reachable in M along the trace $\langle c!a_1 \rangle \langle c!a_2 \rangle \langle c?a_1 \rangle \langle c!a_3 \rangle \langle c!a_3 \rangle$, and so is $((q_0, \mathrm{R}, \mathrm{L}), (1, 2))$ in $\mathcal{C}$ along $\mathsf{inc}(x)\mathsf{inc}(x)\mathsf{dec}(x)\mathsf{inc}(y)\mathsf{inc}(y)$ (all zero tests are empty).

But how about the converse? In general, one may associate with a counter valuation such as $(4,0)$ several channel contents. Actually, both $a_1a_2a_1a_2$ and $a_2a_1a_2a_1$ seem suitable. However, if we know the most recent message that has been sent, say a_1, then this leaves only one option, namely $a_2a_1a_2a_1$. In this way, we can associate with each counter valuation $\mathbf{v}$ and message $a_i \in \Sigma_c$ a unique (if it exists at all) possible channel contents $[\![\mathbf{v}]\!]_{a_i}$. Suppose that τ is a trace in $\mathcal{C}$ arising from a trace σ in M whose last sent message is a_i. If $(\overline{q}, \mathbf{v})$ is reachable in $\mathcal{C}$ via τ, then $(\overline{q}, [\![\mathbf{v}]\!]_{a_i})$ is reachable in M via σ. For example, $\tau = \mathsf{inc}(x)\mathsf{inc}(x)\mathsf{dec}(x)$ allows us to go to configuration $((q_0, \mathrm{M}, \mathrm{L}), (1,0))$. It arises from $\sigma = \langle c!a_1\rangle\langle c!a_2\rangle\langle c?a_1\rangle \in \mathit{Pref}(\mathcal{V})$, whose last sent message is a_2. We have $[\![(1,0)]\!]_{a_2} = a_2$. Indeed, σ leads to $((q_0, \mathrm{M}, \mathrm{L}), a_2)$. ⌟

Relation between FIFO Machine and Counter Machine

Recall that the FIFO machine $M = (Q, \mathit{Ch}, \Sigma, T, q_0)$ and $\mathscr{L} = (L_c)_{c\in\mathit{Ch}}$ are in normal form, where L_c is a bounded language over $(w_{c,1}, \ldots, w_{c,n_c})$. Let $\mathcal{C} = (Q, \mathit{Cnt}, T', q_0)$ be the associated counter machine. We will now formalize the tight forth-and-back correspondence that allows us to solve reachability queries in M in terms of reachability queries in $\mathcal{C}$.

We start with a simple observation concerning the traces of M and $\mathcal{C}$.

▶ **Lemma 17.** *We have* $\mathit{Traces}(M) \subseteq \mathit{Pref}(\mathcal{V})$ *and* $\mathit{Traces}(\mathcal{C}) \subseteq L_{\mathcal{C}}^{\mathsf{zero}}$.

With every channel contents $\mathbf{w} \in \prod_{c\in\mathit{Ch}} \Sigma_c^*$ of the FIFO machine M, we associate a counter valuation $\langle\!\langle \mathbf{w} \rangle\!\rangle = \mathbf{v} \in \mathbb{N}^{\mathit{Cnt}}$ where, for each counter $x_{(c,i)}$, we let $\mathbf{v}_{x_{(c,i)}} = \sum_{a\in\Sigma_{c,i}} |\mathbf{w}_c|_a$. Furthermore, abusing notation, we define a homomorphism $\langle\!\langle\, . \,\rangle\!\rangle : A_M^* \to A_{\mathcal{C}}^*$ which maps a sequence of actions of M to a sequence of actions of $\mathcal{C}$. It is defined by $\langle\!\langle \langle c!a\rangle \rangle\!\rangle = (\mathsf{inc}(x_{(c,i_a)}), \emptyset)$ and $\langle\!\langle \langle c?a\rangle \rangle\!\rangle = (\mathsf{dec}(x_{(c,i_a)}), Z)$ where $Z = \{x_{(c,j)} \mid j < i_a\}$.

Conversely, we will associate, with counter values and traces of $\mathcal{C}$ the corresponding objects in the FIFO machine. Because of the inherent ambiguity, this is, however, less straightforward. First, we define a partial mapping $[\![\, . \,]\!] : A_{\mathcal{C}}^* \to A_M^*$ (that is not a homomorphism). For $\tau \in A_{\mathcal{C}}^*$, we let $[\![\tau]\!]$ be the unique (if it exists) word $\sigma \in \mathit{Pref}(\mathcal{V})$ such that $\langle\!\langle \sigma \rangle\!\rangle = \tau$.

Next, we associate with a counter valuation a corresponding channel contents. As explained above, there is no unique choice unless we make an assumption on the last messages that have been sent. For $c \in \mathit{Ch}$, we set $\Sigma_c^\perp = \Sigma_c \uplus \{\perp\}$. Let $a \in \Sigma_c^\perp$ and $w \in \Sigma_c^*$. We say that a is *good* for w if $w \in \mathit{Infix}(L_c)$ and either $w = \varepsilon$ or $w = u.a$ for some $u \in \Sigma_c^*$. Intuitively, it may be possible to obtain contents w in channel c when a is the last message sent (no message was sent yet through c if $a = \perp$). Note that the set of words $w \in \Sigma_c^*$ such that a is good for w is a regular language. Moreover, with $\mathbf{w} \in \prod_{c\in\mathit{Ch}} \Sigma_c^*$, we associate the finite set $G(\mathbf{w}) \subseteq \prod_{c\in\mathit{Ch}} \Sigma_c^\perp$ of tuples $\mathbf{a} = (\mathbf{a}_c)_{c\in\mathit{Ch}}$ such that, for all $c \in \mathit{Ch}$, $\mathbf{a}_c$ is good for $\mathbf{w}_c$.

Let $\mathbf{v} \in \mathbb{N}^{\mathit{Cnt}}$ and $\mathbf{a} \in \prod_{c\in\mathit{Ch}} \Sigma_c^\perp$. Abusing notation, we will associate with $\mathbf{v}$ and $\mathbf{a}$ the channel contents $[\![\mathbf{v}]\!]_{\mathbf{a}} \in \prod_{c\in\mathit{Ch}} \Sigma_c^*$ (if it exists). We let $[\![\mathbf{v}]\!]_{\mathbf{a}} = \mathbf{w}$ if $\langle\!\langle \mathbf{w} \rangle\!\rangle = \mathbf{v}$ and $\mathbf{a} \in G(\mathbf{w})$. There is at most one such $\mathbf{w}$ so that this is well-defined. Note that $\langle\!\langle [\![\mathbf{v}]\!]_{\mathbf{a}} \rangle\!\rangle = \mathbf{v}$.

▶ **Example 18.** If we have one channel c and our bounded language is $L_c = (a_1a_2a_3)^*(a_4)^*$, then $[\![(4,0)]\!]_{a_2} = a_2a_3a_1a_2$ and $[\![(2,1)]\!]_{a_4} = a_2a_3a_4$, whereas $[\![(3,1)]\!]_{a_3}$ is undefined. Moreover, $G(a_2a_3) = \{a_3\}$ and $G(\varepsilon) = \{a_1, a_2, a_3, a_4, \perp\}$. ⌟

Given $\mathbf{v}$ and $\mathbf{a}$, we can easily compute $[\![\mathbf{v}]\!]_{\mathbf{a}}$ since there are only finitely many words $\mathbf{w}$ for a given $\mathbf{v}$ such that $\langle\!\langle \mathbf{w} \rangle\!\rangle = \mathbf{v}$. Furthermore, we can also compute $G(\mathbf{w})$ for a given $\mathbf{w}$ as we have finitely many possibilities of $\mathbf{a}$.

Finally, for $\mathbf{a} \in \prod_{c\in\mathit{Ch}} \Sigma_c^\perp$, we let $L_{\mathbf{a}}^{\mathsf{last}} \subseteq A_M^*$ be the set of words σ such that, for all $c \in \mathit{Ch}$, $\mathbf{a}_c$ is the last message sent to c in σ (no message was sent if $\mathbf{a}_c = \perp$). We are now ready to state that runs in the FIFO machine are faithfully simulated by runs in the counter machine (the proof is by induction on the length of the trace):

▶ **Proposition 19.** *Let $\sigma \in A_M^*$. For all $(q, \mathbf{w}) \in S_M$ and $\mathbf{a} \in \prod_{c \in Ch} \Sigma_c^\perp$ such that $\sigma \in L_{\mathbf{a}}^{\mathsf{last}}$, we have: $(q_0, \varepsilon) \xrightarrow{\sigma}_M (q, \mathbf{w}) \implies \big((q_0, \mathbf{0}) \xrightarrow{\langle\!\langle \sigma \rangle\!\rangle}_{\mathcal{C}} (q, \langle\!\langle \mathbf{w} \rangle\!\rangle)$ and $\mathbf{a} \in G(\mathbf{w})\big)$.*

Conversely, we can show that runs of the counter machine can be retrieved in the FIFO machine (again, the proof proceeds by induction on the length of the trace):

▶ **Proposition 20.** *Let $\tau \in A_{\mathcal{C}}^*$. For all $(q, \mathbf{v}) \in S_{\mathcal{C}}$ and $\mathbf{a} \in \prod_{c \in Ch} \Sigma_c^\perp$ such that $\tau \in \langle\!\langle Pref(\mathcal{V}) \cap L_{\mathbf{a}}^{\mathsf{last}} \rangle\!\rangle$, we have: $(q_0, \mathbf{0}) \xrightarrow{\tau}_{\mathcal{C}} (q, \mathbf{v}) \implies (q_0, \varepsilon) \xrightarrow{[\![\tau]\!]}_M (q, [\![\mathbf{v}]\!]_{\mathbf{a}})$.*

From Propositions 19 and 20 and Lemma 17, we obtain the following corollary.

▶ **Corollary 21.** *For all $(q, \mathbf{w}) \in S_M$, we have: $(q, \mathbf{w}) \in Reach_M(\mathscr{L}_!) \iff (q, \langle\!\langle \mathbf{w} \rangle\!\rangle) \in Reach_{\mathcal{C}}(L_{\mathcal{C}}^{\mathsf{zero}} \cap \langle\!\langle \mathcal{V} \cap \bigcup_{\mathbf{a} \in G(\mathbf{w})} L_{\mathbf{a}}^{\mathsf{last}} \rangle\!\rangle)$.*

From Theorem 6, we know that verifying whether $(q, \langle\!\langle \mathbf{w} \rangle\!\rangle) \in Reach_{\mathcal{C}}(L_{\mathcal{C}}^{\mathsf{zero}} \cap L)$ where $L = \langle\!\langle \mathcal{V} \cap \bigcup_{\mathbf{a} \in G(\mathbf{w})} L_{\mathbf{a}}^{\mathsf{last}} \rangle\!\rangle$ is decidable. Hence, we can already deduce decidability of the (configuration-)reachability problem. In fact, using Propositions 19 and 20, we can solve the more general IB rational-reachability problem. For this, it is actually enough to check, in the counter machine, the reachability of a counter value that belongs to a semi-linear set. For $\mathbf{a} \in \prod_{c \in Ch} \Sigma_c^\perp$ and a rational relation $\mathscr{R} \subseteq \prod_{c \in Ch} \Sigma_c^*$, let $V_{\mathbf{a}}(\mathscr{R}) = \{\mathbf{v} \in \mathbb{N}^{Cnt} \mid [\![\mathbf{v}]\!]_{\mathbf{a}} \in \mathscr{R}\}$.

▶ **Lemma 22.** *The set $V_{\mathbf{a}}(\mathscr{R})$ is effectively semi-linear.*

Using this property, we finally reduce the IB rational-reachability problem to a reachability problem in counter machines:

▶ **Corollary 23.** *For every $q \in Q$, we have: $(q, \mathbf{w}) \in Reach_M(\mathscr{L}_!)$ for some $\mathbf{w} \in \mathscr{R} \iff (q, \mathbf{v}) \in Reach_{\mathcal{C}}(L_{\mathcal{C}}^{\mathsf{zero}} \cap \langle\!\langle \mathcal{V} \cap L_{\mathbf{a}}^{\mathsf{last}} \rangle\!\rangle)$ for some $\mathbf{a} \in \prod_{c \in Ch} \Sigma_c^\perp$ and $\mathbf{v} \in V_{\mathbf{a}}(\mathscr{R})$.*

By Theorem 6, we can now deduce Theorem 11, i.e., decidability of IB rational-reachability.

4 Reachability, Deadlock, Unboundedness, and Termination

We now address some other commonly studied reachability problems, which, as it turns out, can be reduced to the IB rational-reachability problem studied in the previous section.

A configuration $(q, \mathbf{w})$ of a FIFO machine M is a *deadlock* if there is no $(q', \mathbf{w}')$ such that $(q, \mathbf{w}) \to_M (q', \mathbf{w}')$.

▶ **Definition 24** (IB decision problems). *Given a FIFO machine $M = (Q, Ch, \Sigma, T, q_0)$, a control-state $q \in Q$, a configuration $s \in S_M$, and a tuple $\mathscr{L} = (L_c)_{c \in Ch}$ of non-empty regular bounded languages $L_c \subseteq \Sigma_c^*$.*

- IB reachability*: Do we have $s \in Reach_M(\mathscr{L}_!)$?*
- IB control-state reachability*: Do we have $(q, \mathbf{w}) \in Reach_M(\mathscr{L}_!)$ for some $\mathbf{w}$?*
- IB deadlock*: Does $Reach_M(\mathscr{L}_!)$ contain a deadlock ?*
- IB unboundedness*: Is $Reach_M(Pref(\mathscr{L}_!))$ infinite?*
- IB termination*: Is there no infinite execution of the form $init_M \xrightarrow{\beta_1} s_1 \xrightarrow{\beta_2} s_2 \xrightarrow{\beta_3} \dots$ such that, for all $i \in \mathbb{N}$, we have $s_i \in S_M$, $\beta_i \in A_M$, and $\beta_1 \dots \beta_i \in Pref(\mathscr{L}_!)$?*

Reachability and Deadlock

In [18], it was shown that reachability reduces to control-state reachability for flat FIFO machines but the converse is not true. However, using the same reductions as in [18], we obtain the following results:

▶ **Proposition 25.** *IB reachability is*
(a) *recursively equivalent to IB control-state reachability for FIFO machines, and*
(b) *recursively reducible to IB deadlock for FIFO machines.*

If, for a given $q \in Q$, we set $\mathscr{R}$ to be the universal relation $\prod_{c \in Ch} \Sigma_c^*$, IB rational-reachability captures IB control-state reachability, and if we set $\mathscr{R} = \{\mathbf{w}\}$, we can decide if the configuration $(q, \mathbf{w})$ is reachable. In order to reduce IB deadlock to IB rational-reachability, we first explore the control states in order to find the set of states $Q' \subseteq Q$ which allow only receptions (no control states with sends can be part of a deadlock). This set can easily be computed from the set of transitions of the machine. Then, for each $q \in Q'$, we can see if there exists a reachable configuration $(q, \mathbf{w})$ such that, for all c, we have $\mathbf{w}_c \in K_c = \{\varepsilon\} \cup \{a.u \mid u \in \Sigma_c^* \text{ and } a \in \Sigma_c \text{ such that there is no transition } (q, \langle c?a \rangle, q') \text{ in } M\}$. Note that $\mathscr{R}_q = \prod_{c \in Ch} K_c$ is recognizable and, therefore, rational. Furthermore, if there exists such a reachable $(q, \mathbf{w})$ with $\mathbf{w} \in \mathscr{R}_q$, then it is a deadlock. Hence, using the fact that generalized IB rational-reachability is decidable (Theorem 11), we immediately deduce the following corollary:

▶ **Corollary 26.** *The problems IB reachability, IB control-state reachability, and IB deadlock are decidable for FIFO machines.*

▶ Remark. Since input-bounded FIFO machines subsume VASS (a VASS can be seen as an input-bounded FIFO machine with an alphabet reduced to a unique letter), the complexity of IB reachability is not elementary, which is inherited from the lower bound for VASS [11].

Unboundedness and Termination

IB unboundedness in FIFO machines reduces to an equivalent problem in counter machines. Given a FIFO machine $\hat{M}$ and $\hat{\mathscr{L}}$, the associated FIFO machine M in normal form (with the corresponding tuple $\mathscr{L}$ of distinct-letter languages), as well as the associated counter machine $\mathcal{C}$, the following result can be derived.

▶ **Proposition 27.** *$Reach_{\hat{M}}(\hat{\mathscr{L}}_!)$ is infinite iff $Reach_M(\mathscr{L}_!)$ is infinite iff $Reach_{\mathcal{C}}(L_{\mathcal{C}}^{\mathsf{zero}} \cap \langle\!\langle \mathcal{V} \rangle\!\rangle)$ is infinite.*

This statement also applies to prefix-closed languages so we have $Reach_{\hat{M}}(Pref(\hat{\mathscr{L}}_!))$, $Reach_M(Pref(\mathscr{L}_!))$, and $Reach_{\mathcal{C}}(L_{\mathcal{C}}^{\mathsf{zero}} \cap \langle\!\langle Pref(\mathcal{V}) \rangle\!\rangle)$ are either all infinite or all finite. The latter-most is decidable as we establish in the following. Recall that, by the construction of $\mathcal{C}$, we have $Reach_{\mathcal{C}} = Reach_{\mathcal{C}}(L_{\mathcal{C}}^{\mathsf{zero}} \cap \langle\!\langle Pref(\mathcal{V}) \rangle\!\rangle)$.

The main idea that follows is the reduction of unboundedness of the counter machine to reachability in a modified counter machine. It is not immediate that we can use the results in the literature (for example [13]) which reduce boundedness to reachability in Petri nets/VASS. This is because the property of monotonicity does not extend to zero tests; if one can execute a zero test at $(q, \mathbf{v})$, it is not necessarily the case that it can be executed at $(q, \mathbf{v}')$ with $\mathbf{v} \leq \mathbf{v}'$, where we let $\mathbf{v} \leq \mathbf{v}'$ if $\mathbf{v}_x \leq \mathbf{v}'_x$ for all $x \in Cnt$. However, we show that, for the counter machine that we construct from the FIFO machine, this property does hold. If we are able to show this, the constructions used in the case of VASS can be adapted.

▶ **Lemma 28.** *For every execution in $\mathcal{C}$ of the form $(q_0, \mathbf{0}) \xrightarrow{\sigma} (q, \mathbf{v}) \xrightarrow{\sigma'} (q, \mathbf{v}')$ such that $\mathbf{v} \leq \mathbf{v}'$, the following holds: The only counters that are tested to zero during σ' already evaluate to zero at $(q, \mathbf{v})$, and do not change their value throughout the execution of σ'.*

Proof. Let us assume to the contrary that there is at least one counter which is incremented or decremented during σ' and also tested to zero during the execution. Without loss of generality, let us consider $x_{(c,i)}$ to be the first counter along the execution σ' that is tested to zero during σ' and also incremented/decremented before it was tested to zero.

- Case (1): It has a non-zero value at $(q, \mathbf{v})$, and is then either decremented, or first incremented and then decremented, and finally tested to zero. Since we know that $\sigma.\sigma' \in L_{\mathcal{C}}^{\mathsf{zero}}$, no counter tested to zero can then be incremented. Hence, its value will remain zero. But this is a contradiction to our assumption that $\mathbf{v} \leq \mathbf{v}'$. Hence, all the counters with non-zero values at $(q, \mathbf{v})$ cannot be tested to zero during σ'.
- Case (2): It has value zero in $(q, \mathbf{v})$, and is incremented, then decremented, then tested to zero during σ'. This implies that it first has to be incremented. Consider now some sub-execution $\sigma'' \in \mathit{Pref}(\sigma')$ where $(q, \mathbf{v}) \xrightarrow{\sigma''} (q_1, \mathbf{v}_1)$ such that the value of $x_{(c,i)}$ in the configuration $(q_1, \mathbf{v}_1)$ is non-zero. Since there are no "new" zero-tests along the execution σ'' (by our assumption), we can execute σ'' from $(q, \mathbf{v}')$ (by the monotonicity and trace property). However, we cannot increment the counter $x_{(c,i)}$ along σ'', because it was tested to zero during the run $(q_0, \mathbf{0}) \xrightarrow{\sigma.\sigma'} (q, \mathbf{v}')$. Hence, we once again have a contradiction. ◀

Now, we can use results from [19] to show the following:

▶ **Proposition 29.** *The set $\mathit{Reach}_{\mathcal{C}}$ is infinite iff there exist $\sigma, \sigma' \in A_{\mathcal{C}}^*$, $q \in Q$, and $\mathbf{v}, \mathbf{v}' \in \mathbb{N}^{Cnt}$ such that $(q_0, \mathbf{0}) \xrightarrow{\sigma} (q, \mathbf{v}) \xrightarrow{\sigma'} (q, \mathbf{v}')$ and $\mathbf{v} < \mathbf{v}'$ (i.e., $\mathbf{v} \leq \mathbf{v}'$ and $\mathbf{v} \neq \mathbf{v}'$).*

Construction of modified counter machine. We modify the counter machine $\mathcal{C}$ and construct a new counter machine $\mathcal{C}'$ such that $\mathit{Reach}_{\mathcal{C}}$ is infinite iff a configuration belonging to a finite set is reachable in $\mathcal{C}'$. The construction is loosely based on the reduction of boundedness to reachability for Petri Nets in [13]. Since we do not know the values of $\mathbf{v}$ and $\mathbf{v}'$ a priori, we will try to characterize the general condition. The difference $\mathbf{v}' - \mathbf{v}$ is a non negative vector, with at least one strictly positive component. We add a duplicate set of counters for every counter in the system. The intuition is that the counter machine non-deterministically moves from operating on both sets to a configuration from where it only operates on this second set. The first set will remain unchanged (with the value $\mathbf{v}$), and the second set will keep track of the values (until it reaches $\mathbf{v}'$). From this configuration (which represents $(q, \mathbf{v}')$), we move to a new control state, q_{reach}. Here, we check for the condition $\mathbf{v}' - \mathbf{v} > \mathbf{0}$ by first decrementing each counter in the first set which has a non-zero value in tandem with the corresponding counter in the second set. We do this until all the counters in the first set are equal to zero. If $\mathbf{v}' - \mathbf{v} > \mathbf{0}$, then there is at least one counter in the second set with a non-zero counter value. We non-deterministically decrement all the counters in the second set until we reach a configuration that has some counter c in the second set with a value of 1, and all other counters evaluate to zero. Since there are finitely many such configurations, we can just check every case.

Note that we can extend all these results for the case of termination as well. The only difference is that we now consider configurations $(q, \mathbf{v})$ and $(q, \mathbf{v}')$ such that $\mathbf{v} \leq \mathbf{v}'$. Once again, we can follow a similar argument to reduce the termination to the reachability of a configuration in this same modified counter machine.

Hence, we obtain the following theorem:

▶ **Theorem 30.** *IB unboundedness and IB termination are decidable for FIFO machines.*

▶ Remark. Gouda et al, stated that unboundedness is in EXPSPACE for letter-bounded systems [23]. However, they only give an idea of the proof, stating that it can be done in a similar fashion as for the deadlock problem. In the construction for solving the deadlock problem, they reduce the input language to *tally* letter-bounded languages (tally means that the input-language is included in a^* where a is a letter). They add as many channels as letters in the original letter-bounded-language. Furthermore, in order to ensure that every channel is empty before the next channel is read, they ensure that in all control states where a later channel is being read, there are reception transitions of previous channel contents which lead to a sink state (where there is never a deadlock). Notice that it is still possible to leave a channel non-empty before the next channel is read. But one never reaches a deadlock in such an "incorrect" run, since there is always the option of reading the unread channel contents of the previous channels and reach the sink state.

However, when we consider this model for unboundedness, there may exist unbounded "incorrect" runs since we can leave a channel non-empty and proceed to the next and may have an unbounded run. Hence, it seems that one still needs some reachability test to check if the runs are correct as we cannot ensure that some channels are zero in an unbounded run.

5 Conclusion and Perspectives

We extend recent results of the *bounded verification* of communicating finite-state machines (equivalently FIFO machines) [14] and of *flat* FIFO machines [18] by using bounded languages for controlling the input-languages of FIFO channels (and not for controlling the runs of the machine). We extend old and recent results about input-bounded FIFO machines (see Table 1). In particular, we introduce the rational-reachability problem, which subsumes most of the well-known variants of reachability problems like: the (classical) reachability problem, the control-state reachability problem, and the deadlock problem. We also unify the terminology to facilitate the comparison between results. Moreover, note that, for most problems (except general/rational reachability), we can reduce *output-bounded* reachability to an equivalent input-bounded problem. There are still many open problems and challenges:

- What is the precise complexity of the five problems for input-bounded FIFO machines with a fixed number of channels?
- What is the precise complexity of control-state reachability, deadlock, unboundedness, and termination for input-bounded FIFO machines?
- The size of the counter machine associated with a FIFO machine and a tuple of bounded languages is exponential, but only polynomial when we start from a normal form. It will be interesting to see whether the use of existing tools for counter machines is feasible for the verification of FIFO machines from case studies. Case studies shall also reveal how many FIFO machines/systems are actually boundable and/or flattable.

Towards a theory of boundable FIFO machines. In Example 9, we have seen that all configurations that are reachable in the CDP protocol are already reachable in presence of a suitable collection $\mathscr{L}$ of bounded input-languages. By analogy with the well-established theory of *flattable* machines [3, 12, 8], we propose the following definition.

Table 1 Summary of key results; results for all other extensions are subsumed by these results (D stands for decidable).

	Flat	Letter-bounded	Bounded
UNBOUND	NP-C ([18])	D ([23])	D ([26])
TERM	NP-C ([18])	D	**D**
REACH	NP-C ([18])	D	**D, not ELEM**
CS-REACH	NP-C ([14, 18])	D	D
DEADLOCK	D	D ([23])	**D**

▶ **Definition 31.** *Let M be a FIFO machine and let $\mathscr{L}$ be a tuple of regular bounded languages. We say that M is* $\mathscr{L}$-boundable *if $Reach_M = Reach_M(\mathscr{L}_!)$. We say that M is* boundable *if there exists a tuple $\mathscr{L}$ of regular bounded languages such that M is* $\mathscr{L}$-boundable.

Hence, we deduce that reachability is decidable for $\mathscr{L}$-boundable FIFO machines, which is a *strictly larger* class than input-bounded machines. CDP is not input-bounded but it is $\mathscr{L}_{CDP}$-boundable with $\mathscr{L}_{CDP} = ((ab)^*(a+\varepsilon)(ab)^*, e^*)$. Let us also remark that CDP is flattable by using the bounded set of runs $(!a!b)^*!a!e?e(!a!b)^* + (!a!b)^*$ (where we omit channel information for readability), because it covers the reachability set which is equal to $(ab)^*(a+\varepsilon)(ab)^*$ on control-state $(0,0)$. It is not clear whether reachability is decidable for boundable machines. A strategy that would fairly enumerate *all* regular bounded families $\mathscr{L}_1, \mathscr{L}_2, \ldots, \mathscr{L}_n, \ldots$ will necessarily find the good one, if M is boundable, but this is not sufficient because we must be able to *recognize* $Reach_M$. Observe that boundable machines are more robust than flat machines. Consider a system $\mathcal{S} = (\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_n)$ of n flat finite automata $\mathcal{A}_i$ communicating peer to peer (P2P) through one-directional FIFO channels. Let $M_{\mathcal{S}}$ denote FIFO the machine obtained as the Cartesian product of all automata $\mathcal{A}_i$ of $\mathcal{S}$; there is no reason to assume that $M_{\mathcal{S}}$ is flattable but it is input-bounded and thus $M_{\mathcal{S}}$ is $\mathscr{L}$-boundable where $\mathscr{L}$ is computable from $\mathcal{S}$.

References

1. Parosh Aziz Abdulla, Aurore Collomb-Annichini, Ahmed Bouajjani, and Bengt Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Formal Methods Syst. Des.*, 25(1):39–65, 2004. doi:10.1023/B:FORM.0000033962.51898.1a.
2. C. Aiswarya, Paul Gastin, and K. Narayan Kumar. Verifying communicating multi-pushdown systems via split-width. In *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014*, volume 8837 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2014.
3. Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Laure Petrucci. FAST: Acceleration from theory to practice. *International Journal on Software Tools for Technology Transfer*, 10(5):401–424, October 2008. doi:10.1007/s10009-008-0064-3.
4. Jean Berstel. *Transductions and context-free languages*, volume 38 of *Teubner Studienbücher : Informatik*. Teubner, 1979.
5. Ahmed Bouajjani, Constantin Enea, Kailiang Ji, and Shaz Qadeer. On the completeness of verifying message passing programs under bounded asynchrony. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Proceedings, Part II*, volume 10982 of *Lecture Notes in Computer Science*, pages 372–391. Springer, 2018. doi:10.1007/978-3-319-96142-2_23.
6. Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983. doi:10.1145/322374.322380.

7 Gérard Cécé and Alain Finkel. Verification of programs with half-duplex communication. *Inf. Comput.*, 202(2):166–190, 2005. doi:10.1016/j.ic.2005.05.006.

8 Pierre Chambart, Alain Finkel, and Sylvain Schmitz. Forward analysis and model checking for trace bounded WSTS. In Lars M. Kristensen and Laure Petrucci, editors, *Proceedings of the 32nd International Conference on Applications and Theory of Petri Nets (PETRI NETS'11)*, volume 6709 of *Lecture Notes in Computer Science*. Springer, 2011. doi:10.1007/978-3-642-21834-7_4.

9 Pierre Chambart and Philippe Schnoebelen. The ordinal recursive complexity of lossy channel systems. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008*, pages 205–216. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.47.

10 Christian Choffrut. Relations over words and logic: A chronology. *Bulletin of the EATCS*, 89:159–163, 2006.

11 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for petri nets is not elementary. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019*, pages 24–33. ACM, 2019.

12 Stéphane Demri, Alain Finkel, Valentin Goranko, and Govert van Drimmelen. Model-checking CTL* over flat Presburger counter systems. *Journal of Applied Non-Classical Logics*, 20(4):313–344, 2010. doi:10.3166/jancl.20.313-344.

13 Catherine Dufourd and Alain Finkel. Polynomial-Time Many-One Reductions for Petri Nets. In S. Ramesh and G. Sivakumar, editors, *Foundations of Software Technology and Theoretical Computer Science, 17th Conference*, volume 1346 of *Lecture Notes in Computer Science*, pages 312–326. Springer, 1997. doi:10.1007/BFb0058039.

14 Javier Esparza, Pierre Ganty, and Rupak Majumdar. A perfect model for bounded verification. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012*, pages 285–294. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.39.

15 Alain Finkel. About monogeneous fifo Petri nets. In *Proceedings of the 3rd International Conference on Applications and Theory of Petri Nets (APN'82)*, Varenna, Italy, September 1982.

16 Alain Finkel. Decidability of the termination problem for completely specified protocols. *Distributed Computing*, 7(3):129–135, 1994. doi:10.1007/BF02277857.

17 Alain Finkel and Annie Choquet. Simulation of linear fifo nets by Petri nets having a structured set of terminal markings. In *Proceedings of the 8th International Conference on Applications and Theory of Petri Nets (APN'87)*, Zaragoza, Spain, June 1987.

18 Alain Finkel and M. Praveen. Verification of flat FIFO systems. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019*, volume 140 of *LIPIcs*, pages 12:1–12:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.12.

19 Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001. doi:10.1016/S0304-3975(00)00102-X.

20 Blaise Genest, Dietrich Kuske, and Anca Muscholl. On communicating automata with bounded channels. *Fundam. Inform.*, 80(1-3):147–167, 2007. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi80-1-3-09.

21 Seymour Ginsburg and Edwin H. Spanier. Bounded Algol-Like Languages. *Transactions of the American Mathematical Society*, 113(2):333–368, 1964. URL: http://www.jstor.org/stable/1994067.

22 Cinzia Di Giusto, Laetitia Laversa, and Étienne Lozes. On the k-synchronizability of systems. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020*, volume 12077 of *Lecture Notes in Computer Science*, pages 157–176. Springer, 2020. doi:10.1007/978-3-030-45231-5_9.

23 M. G. Gouda, E. M. Gurari, T. H. Lai, and L. E. Rosier. On deadlock detection in systems of communicating finite state machines. *Comput. Artif. Intell.*, 6(3):209–228, July 1987.

24 Alexander Heußner, Jérôme Leroux, Anca Muscholl, and Grégoire Sutre. Reachability analysis of communicating pushdown systems. In C.-H. Luke Ong, editor, *Foundations of Software Science and Computational Structures, 13th International Conference, FOSSACS 2010*, volume 6014 of *Lecture Notes in Computer Science*, pages 267–281. Springer, 2010. doi:10.1007/978-3-642-12032-9_19.

25 Thierry Jéron. Testing for unboundedness of FIFO channels. In Christian Choffrut and Matthias Jantzen, editors, *STACS 91, 8th Annual Symposium on Theoretical Aspects of Computer Science*, volume 480 of *Lecture Notes in Computer Science*, pages 322–333. Springer, 1991. doi:10.1007/BFb0020809.

26 Thierry Jéron and Claude Jard. Testing for unboundedness of FIFO channels. *Theor. Comput. Sci.*, 113(1):93–117, 1993. doi:10.1016/0304-3975(93)90212-C.

27 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Context-bounded analysis of concurrent queue systems. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008*, volume 4963 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 2008. doi:10.1007/978-3-540-78800-3_21.

28 P. Madhusudan and Gennaro Parlato. The tree width of auxiliary storage. In *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011*, pages 283–294. ACM, 2011.

29 Ernst W. Mayr. An algorithm for the general petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.

30 Gérard Memmi and Alain Finkel. An introduction to fifo nets-monogeneous nets: A subclass of fifo nets. *Theor. Comput. Sci.*, 35:191–214, 1985. doi:10.1016/0304-3975(85)90014-3.

31 Bernard Vauquelin and Paul Franchi-Zannettacci. Automates a file. *Theor. Comput. Sci.*, 11:221–225, 1980. doi:10.1016/0304-3975(80)90047-X.

32 Yao-Tin Yu and Mohamed G. Gouda. Unboundedness detection for a class of communicating finite-state machines. *Inf. Process. Lett.*, 17(5):235–240, 1983. doi:10.1016/0020-0190(83)90105-9.

Propositional Dynamic Logic for Hyperproperties

Jens Oliver Gutsfeld
Institut für Informatik, Westfälische Wilhelms-Universität Münster, Germany
jens.gutsfeld@uni-muenster.de

Markus Müller-Olm
Institut für Informatik, Westfälische Wilhelms-Universität Münster, Germany
markus.mueller-olm@uni-muenster.de

Christoph Ohrem
Institut für Informatik, Westfälische Wilhelms-Universität Münster, Germany
christoph.ohrem@uni-muenster.de

Abstract

Information security properties of reactive systems like non-interference often require relating different executions of the system to each other and following them simultaneously. Such *hyperproperties* can also be useful in other contexts, e.g., when analysing properties of distributed systems like linearizability. Since common logics like LTL, CTL, or the modal μ-calculus cannot express hyperproperties, the hyperlogics HyperLTL and HyperCTL* were developed to cure this defect. However, these logics are not able to express arbitrary ω-regular properties. In this paper, we introduce HyperPDL-Δ, an adaptation of the Propositional Dynamic Logic of Fischer and Ladner for hyperproperties, in order to remove this limitation. Using an elegant automata-theoretic framework, we show that HyperPDL-Δ model checking is asymptotically not more expensive than HyperCTL* model checking, despite its vastly increased expressive power. We further investigate fragments of HyperPDL-Δ with regard to satisfiability checking.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Verification by model checking; Theory of computation → Logic and verification; Theory of computation → Automata over infinite objects

Keywords and phrases Hyperlogics, Hyperproperties, Model Checking, Automata

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2020.50

Related Version An extended version of this paper is available at `https://arxiv.org/abs/1910.10546`.

Funding This work was partially funded by DFG project Model-Checking of Navigation Logics (MoNaLog) (MU 1508/3).

Acknowledgements We thank the reviewers for their helpful comments.

1 Introduction

Temporal logics like LTL, CTL or CTL* have been used successfully in verification. These logics consider paths of a structure (in linear time logics) or paths and their possible extensions (in branching time logics). Notably, since they cannot refer to multiple paths at once, they cannot express *hyperproperties* that relate multiple paths to each other. Examples of hyperproperties include information security properties like non-interference [6] or properties of distributed systems like linearizablity [2]. In order to develop a dedicated logic for these properties, Clarkson et. al. [5, 12] introduced HyperLTL and HyperCTL*, which extend LTL and CTL* by path variables. However, just like LTL and CTL [22], they cannot express arbitrary ω-regular properties of traces [20], a desirable property of specification logics [1, 16]. Logics like Propositional Dynamic Logic (PDL) [13, 18] and Linear Dynamic Logic (LDL) [8, 10] are able to do so for single traces. As we seek to extend this ability to

hyperproperties, we introduce a variant of PDL for hyperproperties called HyperPDL-Δ in this paper. HyperPDL-Δ properly extends logics like HyperLTL, HyperCTL*, LDL and PDL-Δ and can express all ω-regular properties over hypertraces in a handy formalism based on regular expressions over programs. We develop a model checking algorithm for HyperPDL-Δ inspired by one for HyperCTL* [12] and show that the model checking problem for HyperPDL-Δ is decidable at no higher asymptotic cost than the corresponding problem for HyperCTL* despite the vastly increased expressive power. Our algorithm non-trivially differs from the algorithm in [12] in two ways: first of all, it handles more general, regular modalities that subsume the modalities of HyperCTL* and require different constructions. Then, we use a different notion of alternation depth (called *criticality*) which conservatively extends their notion, but requires handling structurally different operators and regular expressions. We also show that for fragments of HyperPDL-Δ similar to the fragments of HyperLTL considered in [11], the satisfiability problem is decidable.

This paper is structured as follows: Section 2 introduces Kripke Transition Systems and (alternating) Büchi automata. In Section 3, we define our new logic HyperPDL-Δ and describe properties expressible in it. Afterwards, in Section 4, we outline a model checking algorithm for HyperPDL-Δ and show that it is asymptotically optimal by providing a precise complexity classification. In Section 5, we consider fragments of HyperPDL-Δ for which the satisfiability problem is decidable. Then, in Section 6, we show that HyperPDL-Δ can express all ω-regular properties over sets of traces and compare it to existing hyperlogics with regard to expressivity. Finally, in Section 7, we provide a summary of this paper. The appendices contain two constructions only sketched in the main body of this paper. Due to lack of space, some proofs can be found in the appendix of the extended version only.

Related Work. Hyperproperties were systematically analysed in [6] and dedicated temporal logics for hyperproperties, HyperLTL and HyperCTL*, were introduced in [5]. An overview of temporal hyperlogics and discussion of their expressive power can be found in [7]. Efficient model checking algorithms for these logics were introduced in [12] by Finkbeiner et al. and our model checking algorithm for HyperPDL-Δ builds on ideas from their construction. Our satisfiability algorithm, on the other hand, is inspired by the corresponding algorithm for HyperLTL [11]. Recently, Bonakdapour et. al. proposed *regular hyperlanguages* and a corresponding automata model [3]. In contrast to our work, their model is concerned with hyperproperties over finite instead of infinite words and does not concern branching-time properties. Moreover, they study automata-theoretic questions while our focus here is on verification of hyperproperties specified by logical means. A different line of research for properties involving multiple traces at once is given by *epistemic temporal logics* [14]. An attempt to unify epistemic temporal logics and hyperlogics is given by Bozzelli et. al in [4] via a variant of HyperCTL* with past modalities. PDL was originally introduced in [13] by Fischer and Ladner and has been extended in multiple ways [18]. There are several attempts to extend temporal logic by regular properties: a variant of PDL for linear time properties of finite traces, LDL_f, was introduced in [8] and was extended upon for the infinite setting in [10] by introducing parametrised operators. Other regular extensions of temporal logics were studied e.g. by Wolper [22] or by Kupferman et. al [16]. However, all these extensions do not concern hyperproperties.

2 Preliminaries

Let AP be a finite set of atomic propositions and Σ a finite set of atomic programs. A *Kripke Transition System* (KTS) is a tuple $\mathcal{T} = (S, s_0, \{\delta_\sigma \mid \sigma \in \Sigma\}, L)$ where S is a finite set of states, $s_0 \in S$ is an initial state, $\delta_\sigma \subseteq S \times S$ is a transition relation for each $\sigma \in \Sigma$ and

$L : S \to 2^{AP}$ is a labeling function. We assume that there are no states without outgoing edges, that is for each $s \in S$, there is $s' \in S$ with $(s, s') \in \delta_\sigma$ for some $\sigma \in \Sigma$. A KTS is a combination of a Kripke Structure and a labeled transition system (LTS) where a Kripke Structure K is a KTS for $|\Sigma| = 1$ and an LTS T is a KTS with the labelling function $s \mapsto \emptyset$ for all $s \in S$. A path in a KTS $\mathcal{T}$ is an infinite alternating sequence $s_0\sigma_0 s_1\sigma_1 ... \in (S\Sigma)^\omega$ where s_0 is the initial state of $\mathcal{T}$ and $(s_i, s_{i+1}) \in \delta_{\sigma_i}$ for all $i \geq 0$. We denote by $Paths(\mathcal{T}, s)$ the set of paths in $\mathcal{T}$ starting in s and by $Paths^*(\mathcal{T}, s)$ the set of corresponding path suffixes $\{p[i, \infty] \mid p \in Paths(\mathcal{T}, s), i \in \mathbb{N}_0\}$ where $p[i, \infty]$ is the path suffix of p starting at index i. A trace is an alternating infinite sequence $t \in (2^{AP}\Sigma)^\omega$. For a path $\pi = s_0\sigma_0 s_1\sigma_1 ...$, the induced trace is given by $L(s_0)\sigma_0 L(s_1)\sigma_1 ....$ For a KTS $\mathcal{T}$ and a state $s \in S$, we write $Traces(\mathcal{T}, s)$ to denote the traces induced by paths of $\mathcal{T}$ starting in s.

An alternating Büchi automaton (ABA) is a tuple $\mathcal{A} = (Q, q_0, \Sigma, \rho, F)$ where Q is a finite set of states, $q_0 \in Q$ is an initial state, Σ is a finite alphabet, $\rho : Q \times \Sigma \to \mathbb{B}^+(Q)$ is a transition function mapping each pair of state and input symbol to a non-empty positive boolean combination of successor states and $F \subseteq Q$ is a set of accepting states. We assume that every ABA has two distinct states $true \in F$ and $false \in Q \setminus F$ with $\rho(true, \sigma) = true$ and $\rho(false, \sigma) = false$ for all $\sigma \in \Sigma$. Thus, all maximal paths in an ABA are infinite. A tree T is a subset of $\mathbb{N}^*$ such that for every node $t \in \mathbb{N}^*$ and every positive integer $n \in \mathbb{N}$: $t \cdot n \in T$ implies (i) $t \in T$ (we then call $t \cdot n$ a child of t), and (ii) for every $0 < m < n$, $t \cdot m \in T$. We assume every node has at least one child. A path in a tree T is a sequence of nodes $t_0 t_1 ...$ such that $t_0 = \varepsilon$ and t_{i+1} is a child of t_i for all $i \in \mathbb{N}_0$. A run of an ABA $\mathcal{A}$ on an infinite word $w \in \Sigma^\omega$ is defined as a Q-labeled tree (T, r) where $r : T \to Q$ is a labelling function such that $r(\varepsilon) = q_0$ and for every node $t \in T$ with children $t_1, ..., t_k$, we have $1 \leq k \leq |Q|$ and the valuation assigning true to the states $r(t_1), ..., r(t_k)$ and false to all other states satisfies $\rho(r(t), w(|t|))$. A run (T, r) is an accepting run iff for every path $t_1 t_2 ...$ in T, there are infinitely many i with $r(t_i) \in F$. A word w is accepted by $\mathcal{A}$ iff there is an accepting run of $\mathcal{A}$ on w. The set of infinite words accepted by $\mathcal{A}$ is denoted by $\mathcal{L}(\mathcal{A})$. A nondeterministic Büchi automaton is an ABA in which every transition rule consists only of disjunctions.

We will make use of two well-known theorems about ABA:

▶ **Proposition 1** ([19]). *For every ABA $\mathcal{A}$ with n states, there is a nondeterministic Büchi automaton $MH(\mathcal{A})$ with $2^{\mathcal{O}(n)}$ states that accepts the same language.*

▶ **Proposition 2** ([19, 17]). *For every ABA $\mathcal{A}$ with n states, there is an ABA $\overline{\mathcal{A}}$ with $\mathcal{O}(n^2)$ states that accepts the complement language, i.e., $\mathcal{L}(\overline{\mathcal{A}}) = \overline{\mathcal{L}(\mathcal{A})}$.*

3 Propositional Dynamic Logic for Hyperproperties

In this section, we define our new logic, HyperPDL-Δ. Structurally, it consists of formulas φ referring to state labels and programs α referring to transition labels. We use the syntax of HyperCTL* as a basis for formulas but replace the modalities $\bigcirc, \mathcal{U}$ and $\mathcal{R}$ by PDL-like expressions $\langle\alpha\rangle\varphi, [\alpha]\varphi$ and $\Delta\alpha$ constructed from programs. These programs α are regular expressions over tuples of atomic programs τ capturing the transition behaviour on the considered paths. Additionally, we allow test-operators $\varphi?$ in α in order to enable constructions like conditional branching.

▶ **Definition 3** (Syntax of HyperPDL-Δ). *Let $N = \{\epsilon, \pi_1, \pi_2 \ldots\}$ be a set of path variables with a special path variable $\epsilon \in N$. A formula φ is a HyperPDL-Δ formula if it is built from the following context-free grammar:*

$$\varphi ::= \exists \pi.\varphi \mid \forall \pi.\varphi \mid a_\pi \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle\alpha\rangle\varphi \mid [\alpha]\varphi \mid \Delta\alpha$$
$$\alpha ::= \tau \mid \varepsilon \mid \alpha + \alpha \mid \alpha \cdot \alpha \mid \alpha^* \mid \varphi?$$

where $\pi \in N \setminus \{\epsilon\}$, $a \in AP$ and $\tau \in (\Sigma \cup \{\cdot\})^n$ for the number $n > 0$ of path quantifiers that α is in scope of. The constructs $\langle\alpha\rangle\varphi, [\alpha]\varphi$ and $\Delta\alpha$ are only allowed in scope of at least one quantifier.

We call a HyperPDL-Δ formula φ *closed* iff all occurrences of path variables π in φ (as indices of atomic propositions or in atomic programs) are bound by a quantifier. In this paper, we only consider closed formulas φ. In programs α, each component of tuples $\tau \in (\Sigma \cup \{\cdot\})^n$ corresponds to one of the path variables bound by a quantifier α is in scope of. We assume that a quantifier that is in scope of $i-1$ other quantifiers quantifies path variable π_i.

Connectives inherited from HyperCTL* are interpreted analogously: quantifiers $\exists$ and $\forall$ should be read as "along some path" and "along all paths". Using different path variables π enables us to refer to multiple paths at the same time. For example, with $\forall\pi_1.\exists\pi_2.\exists\pi_3.\varphi$, one can express that for all paths π_1, there are paths π_2 and π_3 such that φ holds along these three paths. Boolean connectives are defined in the usual way. Atomic propositions $a \in AP$ express information about a state and have to be indexed by a path variable π to express on which path we expect a to hold.

Intuitively, a program α explores all paths it is in scope of synchronously and thus allows us to pose a regular constraint on the sequence of atomic programs visited on path prefixes of equal length. In addition, properties of infinite suffixes can be required at certain points during the exploration using tests $\varphi?$. In formulas φ, atomic propositions a_π on single paths suffice to relate behavior on different paths because boolean connectives are available. Referring to occurences of atomic programs on single paths in programs α in a similar way, however, would reduce expressivity since neither negation nor conjunction are present in α. As a remedy, we use tuples $\tau \in (\Sigma \cup \{\cdot\})^n$ to refer to atomic programs on paths $\pi_1, ..., \pi_n$, where σ in position i means that on path π_i, we expect a use of σ to reach the next state. A wildcard symbol $\cdot$ expresses that any atomic program is allowed on the corresponding path.

The constructs using α can be interpreted as follows: the diamond operator $\langle\alpha\rangle\varphi$ means that α matches a prefix of the current paths after which φ holds. The box operator $[\alpha]\varphi$ is the dual of $\langle\alpha\rangle\varphi$, meaning φ holds at the end of all prefixes matching α. The last construct $\Delta\alpha$ is of a different kind and expresses ω-regular rather than regular properties. It says that α occurs repeatedly, i.e., the currently quantified paths can be divided into infinitely many segments matching α. $\Delta\alpha$ expresses a variant of a Büchi condition. Instead of moving from accepting states to accepting states in a Büchi automaton, one moves from initial states to accepting states repeatedly.

Using our logic, common hyperproperties can be expressed easily and intuitively. Let us consider two examples: the first, *observational determinism* [6], states that if two executions of a system receive equal low security inputs, they are indistinguishable for a low security observer all the time. It can be expressed by $\forall\pi_1.\forall\pi_2.(\bigwedge_{a\in L}(a_{\pi_1} \leftrightarrow a_{\pi_2})) \rightarrow [\bullet^*]\bigwedge_{a\in L}(a_{\pi_1} \leftrightarrow a_{\pi_2})$, where low security observable behaviour is modelled by the atomic propositions in L. Here, we use the common boolean abbreviations $\rightarrow$ for implication and $\leftrightarrow$ for equivalence as

well as $\bullet$ as an abbreviation the program $\tau = (\cdot, ..., \cdot)$. The second example, *generalized noninterference* [6], states that high security injections do not interfere with low security observable behaviour. It can be expressed by stating that for all pairs of executions π_1, π_2 there is a third execution π_3 agreeing with π_1 on high security injections and is indistinguishable from π_2 for a low security observer: $\forall\pi_1.\forall\pi_2.\exists\pi_3.[\bullet^*]\bigwedge_{a\in H}(a_{\pi_1} \leftrightarrow a_{\pi_3}) \wedge \bigwedge_{a\in L}(a_{\pi_2} \leftrightarrow a_{\pi_3})$. Since these hyperproperties can already be expressed in HyperLTL [5], which is subsumed by our logic by encoding $\varphi_1 \mathcal{U} \varphi_2$ formulas with $\langle(\varphi_1? \cdot \bullet)^*\rangle\varphi_2$, it is no surprise, that they can be expressed in HyperPDL-Δ as well.

The ability to express arbitrary ω-regular properties however, allows a much more fine-grained analysis of a system than HyperLTL. For example, by replacing the program $\bullet^*$ with $(\bullet \cdot \bullet)^* \cdot \bullet \cdot (\sigma_1, \sigma_1)$ in the observational determinism formula, we can restrict the requirement on low security outputs to only apply for every other state with the additional constraint that some specific program σ_1 was last executed in both π_1 and π_2. As was argued in [1] for linear time specification logics, the ability to express ω-regular properties can indeed become a practical issue when in an assume-guarantee setting detailed information about the behaviour of the context has to be taken into account in order to prove properties of interest. This indicates that availability of ω-regular properties is not just a theoretical issue in specification logics.

While there are hyperlogics with the ability to express all ω-regular languages [7], these lack properties desirable for verification: HyperQPTL obtains the ability to express ω-regular properties from the addition of propositional quantification, which complicates its use for specification purposes since the user has to keep track of heterogeneous types of quantifiers. The simple property that all executions π_1 and π_2 agree on propositions from a set P in every other state for example is expressed by the HyperQPTL formula $\forall\pi_1.\forall\pi_2.\exists t : t \wedge \square(\bigcirc t \leftrightarrow \neg t) \wedge \square(t \rightarrow \bigwedge_{a\in P} a_{\pi_1} \leftrightarrow a_{\pi_2})$ [15]. The specification of this property in HyperPDL-Δ is much more direct: $\forall\pi_1.\forall\pi_2.[(\bullet \cdot \bullet)^*]\bigwedge_{a\in P} a_{\pi_1} \leftrightarrow a_{\pi_2}$. This example also illustrates another problem: due to the additional quantifier alternation, the only known model checking algorithm for HyperQPTL [20] is exponentially more expensive than that of HyperPDL-Δ for such formulas. S1S[E] on the other hand, while being even more expressive than HyperPDL-Δ, has an undecidable model checking problem [7].

Before formally defining our logic's semantics, we introduce some notation. We call a partial function $\Pi : N \rightsquigarrow \mathit{Paths}^*(\mathcal{T}, s_0)$ with $\mathit{dom}(\Pi) = \{\epsilon, \pi_1, ..., \pi_n\}$ a path assignment and denote by PA the set of all path assignments. In the context of a subformula φ, $\mathit{dom}(\Pi)$ contains exactly the variables it is in scope of as well as ϵ. The path variable ϵ refers to the most recently *assigned* path in a path assignment and is used to ensure that paths induced by quantifiers branch from the most recently quantified path. We use $\{\epsilon \rightarrow p\}$ for a path p to denote the path assignment Π with $\mathit{dom}(\Pi) = \{\epsilon\}$ and $\Pi(\epsilon) = p$. We introduce $\Pi[i, \infty]$ as a notation to manipulate path assignments Π such that $\Pi[i, \infty](\pi) = \Pi(\pi)[i, \infty]$ holds for all $\pi \in \mathit{dom}(\Pi)$. Also, $\Pi[\pi_i \rightarrow p]$ is a notation for a path assignment Π' where $\Pi'(\pi_i) = p$ and $\Pi'(\pi_j) = \Pi(\pi_j)$ for all $j \neq i$. As a convention, we do not count ϵ when determining $|\mathit{dom}(\Pi)|$. For a tuple $\tau = (\sigma_1, ..., \sigma_n)$, we write $\tau|_i$ to refer to σ_i.

We write $\Pi \models_{\mathcal{T}} \varphi$ to denote that in the context of a KTS $\mathcal{T}$, a path assignment Π fulfills a formula φ. We also write $(\Pi, i, k) \in R(\alpha)$ for a path assignment Π and two even numbers $i \leq k$ to denote that the transition labels on the paths in Π between i and k match α. Formally, we define these two relations as follows:

▶ **Definition 4** (Semantics of HyperPDL-Δ). *Given a KTS* $\mathcal{T} = (S, s_0, \{\delta_\sigma \mid \sigma \in \Sigma\}, L)$*, we inductively define both satisfaction of formulas* φ *and programs* α *on path assignments* Π*.*

$$
\begin{array}{ll}
\Pi \models_{\mathcal{T}} \exists \pi.\varphi & \text{iff there is } p \in \mathit{Paths}(\mathcal{T}, \Pi(\epsilon)(0)) \text{ s.t. } \Pi[\pi \to p, \epsilon \to p] \models_{\mathcal{T}} \varphi \\
\Pi \models_{\mathcal{T}} \forall \pi.\varphi & \text{iff for all } p \in \mathit{Paths}(\mathcal{T}, \Pi(\epsilon)(0)) : \Pi[\pi \to p, \epsilon \to p] \models_{\mathcal{T}} \varphi \\
\Pi \models_{\mathcal{T}} a_\pi & \text{iff } a \in L(\Pi(\pi)(0)) \\
\Pi \models_{\mathcal{T}} \neg\varphi & \text{iff } \Pi \not\models_{\mathcal{T}} \varphi \\
\Pi \models_{\mathcal{T}} \varphi_1 \wedge \varphi_2 & \text{iff } \Pi \models_{\mathcal{T}} \varphi_1 \text{ and } \Pi \models_{\mathcal{T}} \varphi_2 \\
\Pi \models_{\mathcal{T}} \varphi_1 \vee \varphi_2 & \text{iff } \Pi \models_{\mathcal{T}} \varphi_1 \text{ or } \Pi \models_{\mathcal{T}} \varphi_2 \\
\Pi \models_{\mathcal{T}} \langle\alpha\rangle\varphi & \text{iff there is } i \geq 0 \text{ s.t. } \Pi[i, \infty] \models_{\mathcal{T}} \varphi \text{ and } (\Pi, 0, i) \in R(\alpha) \\
\Pi \models_{\mathcal{T}} [\alpha]\varphi & \text{iff for all } i \geq 0 \text{ with } (\Pi, 0, i) \in R(\alpha) : \Pi[i, \infty] \models_{\mathcal{T}} \varphi \\
\Pi \models_{\mathcal{T}} \Delta\alpha & \text{iff there are } 0 = k_1 \leq k_2 \leq \ldots \text{ s.t. for all } i \geq 1 : \\
 & \quad (\Pi, k_i, k_{i+1}) \in R(\alpha)
\end{array}
$$

$$
\begin{array}{ll}
(\Pi, i, k) \in R(\tau) & \text{iff } k = i + 2 \text{ and for all } 1 \leq l \leq |\mathit{dom}(\Pi)| : \\
 & \quad \tau|_l = \cdot \text{ or } \Pi(\pi_l)(i+1) = \tau|_l \\
(\Pi, i, k) \in R(\varepsilon) & \text{iff } i = k \\
(\Pi, i, k) \in R(\alpha_1 + \alpha_2) & \text{iff } (\Pi, i, k) \in R(\alpha_1) \text{ or } (\Pi, i, k) \in R(\alpha_2) \\
(\Pi, i, k) \in R(\alpha_1 \cdot \alpha_2) & \text{iff there is } j \text{ s.t. } i \leq j \leq k, (\Pi, i, j) \in R(\alpha_1) \text{ and} \\
 & \quad (\Pi, j, k) \in R(\alpha_2) \\
(\Pi, i, k) \in R(\alpha^*) & \text{iff there are } l \geq 0, i = j_0 \leq j_1 \leq \ldots \leq j_l = k \text{ s.t.} \\
 & \quad \text{for all } 0 \leq m < l : (\Pi, j_m, j_{m+1}) \in R(\alpha) \\
(\Pi, i, k) \in R(\varphi?) & \text{iff } i = k \text{ and } \Pi[i, \infty] \models_{\mathcal{T}} \varphi
\end{array}
$$

A KTS $\mathcal{T}$ satisfies a formula φ, denoted by $\mathcal{T} \models \varphi$, iff $\{\epsilon \to p\} \models_{\mathcal{T}} \varphi$ holds for an arbitrary $p \in \mathit{Paths}(\mathcal{T}, s_0)$. Note that the choice of p ensures that the outermost quantified paths in a formula always branch from the starting state of $\mathcal{K}$, i.e. s_0.

4 Model Checking HyperPDL-Δ

In order to tackle the model checking problem for HyperPDL-Δ, that is to check whether $\mathcal{T} \models \varphi$ holds for arbitrarily given KTS $\mathcal{T}$ and closed HyperPDL-Δ formulas φ, we develop a new algorithm inspired by the HyperCTL* model checking algorithm from [12]. A crucial idea is to represent a path assignment Π with $\mathit{dom}(\Pi) = \{\epsilon, \pi_1, ..., \pi_n\}$ by an ω-word over $(S^n \times \Sigma^n)$. Formally, we define a translation function $\nu : PA \to \bigcup_{n \in \mathbb{N}_0} (S^n \times \Sigma^n)^\omega$ such that a path assignment Π with $\Pi(\pi_i) = s_i^0 \sigma_i^0 s_i^1 \sigma_i^1 ...$ is mapped to $\nu(\Pi) = ((s_0^0, ..., s_n^0), (\sigma_0^0, ..., \sigma_n^0))((s_0^1, ..., s_n^1), (\sigma_0^1, ..., \sigma_n^1))... \in (S^n \times \Sigma^n)^\omega$ for $n = |\mathit{dom}(\Pi)|$. Note that ϵ need not be encoded separately in $\nu(\Pi)$ since $\Pi(\pi_n) = \Pi(\epsilon)$ always holds. Similar to the notation for τ used before, we use the notation s to refer to tuples $(s_1, ..., s_n)$ and write $s|_i$ to refer to s_i. Then, given a formula φ and a KTS $\mathcal{T}$, we construct an ABA $\mathcal{A}_\varphi$ recognising $\nu(\Pi)$ iff $\Pi \models_{\mathcal{T}} \varphi$ holds.

First, we transform all formulas into a variation of negation normal form, where only existential quantifiers are allowed and negation can only occur in front of existential quantifiers, atomic propositions or Δs. This form differs from conventional NNF in that negated existential instead of universal quantifiers are used, because they can be handled more efficiently in our

setup. The transformation is done by driving all negations in the formula inwards using De Morgan's laws and the duality $\neg\langle\alpha\rangle\varphi \equiv [\alpha]\neg\varphi$ while also replacing universal quantifiers $\forall$ with $\neg\exists\neg$ and cancelling double negations successively. For example, the formula $\forall\pi.[\alpha]\neg a_\pi$ is transformed into $\neg\exists\pi.\neg[\alpha]\neg a_\pi$, $\neg\exists\pi.\langle\alpha\rangle\neg\neg a_\pi$ and then $\neg\exists\pi.\langle\alpha\rangle a_\pi$ successively.

Then, as another preprocessing step, all programs α appearing in the formula are inductively translated to an intermediate automaton representation $M_\alpha = (Q, q_0, \Sigma^n, \rho, q_f, \Psi)$. The automaton M_α can be seen as a nondeterministic finite automaton (NFA) with access to oracles for the tests in α which recognises all prefixes up to index j of the ω-word $\nu(\Pi[i,\infty])$ such that $(\Pi, i, j) \in R(\alpha)$. This is formalised in Lemma 5. The only differences in syntax when compared to a conventional NFA are (i) there is exactly one final state q_f instead of a set F, (ii) ε-edges are not eliminated and (iii) we have a state marking function Ψ mapping every state $q \in Q$ to a singleton or empty set of formulas $\Psi(q)$. State markings $\Psi(q)$ are introduced to tackle tests $\psi?$ and are later replaced by transitions to automata $\mathcal{A}_\psi$, which we define in the construction for formulas. These state markings make the standard elimination of ε-edges impossible, which is why we delay the elimination until the markings are eliminated as well. This approach is similar to the one used in [10] to transform LDL formulas into automata. However, we apply the construction in a hyperlogic context and make it more succinct by offering an alternative approach to constructing the transition function, thus avoiding an exponential blowup.

Construction of M_α. We now describe the construction of M_α. When an expression α has one or more subexpressions α_i, we assume that automata $M_{\alpha_i} = (Q_i, q_{0,i}, \Sigma^n, \rho_i, q_{f,i}, \Psi_i)$ are already constructed. Intuitively, most cases are analogous to the translation from regular expressions to NFA. For the case of a test $\psi?$, a state marked with ψ is introduced in between starting and final state. Detailed constructions are shown in Figure 1.

Let $\overset{\varepsilon}{\Rightarrow}_X \subseteq Q \times Q$ for a set of formulas X be the smallest relation such that (i) $q \overset{\varepsilon}{\Rightarrow}_{\Psi(q)} q$ and (ii) $q' \overset{\varepsilon}{\Rightarrow}_X q''$ and $q' \in \rho(q, \varepsilon)$ imply $q \overset{\varepsilon}{\Rightarrow}_{X\cup\Psi(q)} q''$. Then, for $\tau \in \Sigma^n$, let $\overset{\tau}{\Rightarrow}_X$ be the smallest relation such that $q \overset{\tau}{\Rightarrow}_X q''$ iff there is a $q' \in Q$ such that $q \overset{\varepsilon}{\Rightarrow}_X q'$ and $q'' \in \rho(q', \tau)$. These relations capture the encountered markings along ε-paths in M_α in the following way: $q \overset{\varepsilon}{\Rightarrow}_X q'$ holds if there is an ε-path from q to q' in M_α that encounters exactly the state markings in the set X. $q \overset{\tau}{\Rightarrow}_X q'$ is used to describe the same behaviour, but requires an additional τ-step at the end.

We obtain the following Lemma:

▶ **Lemma 5.** *Let Π be a path assignment with $\nu(\Pi) = (s_0\tau_1)(s_2\tau_3)...$ and α a program, then $(\Pi, i, k) \in R(\alpha)$ iff there is a state sequence $q_0q_1...q_m$ with $m = \frac{k-i}{2}$ in M_α and sets of formulas $X_0, ..., X_m$ such that*

- **(i)** *q_0 is the initial state of M_α,*
- **(ii)** *$q_m \overset{\varepsilon}{\Rightarrow}_{X_m} q_f$ for the final state q_f of M_α,*
- **(iii)** *$q_l \overset{\tau_{i+2l+1}}{\Longrightarrow}_{X_l} q_{l+1}$ for all $l < m$ and*
- **(iv)** *$\psi \in X_l$ implies $\Pi[i+2l, \infty] \models_\mathcal{T} \psi$ for all $l \leq m$.*

As mentioned, we transform φ into an ABA $\mathcal{A}_\varphi$ recognising $\nu(\Pi)$ iff $\Pi \models_\mathcal{T} \varphi$ holds. Formally, a language $\mathcal{L} \subseteq (S^n \times \Sigma^n)^\omega$ is called $\mathcal{T}$-equivalent to a formula φ, if for each Π the statements $\nu(\Pi) \in \mathcal{L}(\mathcal{A}_\varphi)$ and $\Pi \models_\mathcal{T} \varphi$ are equivalent; we say that an ABA $\mathcal{A}$ is $\mathcal{T}$-equivalent to φ iff its language $\mathcal{L}(\mathcal{A})$ is $\mathcal{T}$-equivalent to φ. As a closed formula φ is a boolean combination of quantified subformulas, the model checking problem can be solved by performing separate nonemptiness checks on $\mathcal{A}_\psi$ for all maximal quantified subformulas ψ and combining the results in accordance with the global structure of φ. For example, if φ is given as $\psi_1 \wedge \neg\psi_2$ for quantified formulas ψ_1 and ψ_2, one performs emptiness tests on $\mathcal{A}_{\psi_1}$ as well as $\mathcal{A}_{\psi_2}$ and accepts iff the first test is positive and the second test is negative.

$\boldsymbol{\tau}$ $M_\alpha = (\{q_0, q_1\}, q_0, \Sigma^n, \rho, q_1, \Psi)$, $\Psi(q_i) = \emptyset$

$$\rho(q, (\sigma_1, \ldots, \sigma_n)) = \begin{cases} \{q_1\} & \text{if } \forall i. \tau|_i = \cdot \vee \tau|_i = \sigma_i \text{ and } q = q_0 \\ \emptyset & \text{else} \end{cases}$$

$\rho(q, \varepsilon) = \emptyset$

$\boldsymbol{\varepsilon}$ $M_\alpha = (\{q_0, q_1\}, q_0, \Sigma^n, \rho, q_1, \Psi)$, $\Psi(q_i) = \emptyset$

$\rho(q_i, \tau) = \emptyset$

$$\rho(q_i, \varepsilon) = \begin{cases} \{q_{i+1}\} & \text{if } i = 0 \\ \emptyset & \text{else} \end{cases}$$

$\boldsymbol{\alpha_1 + \alpha_2}$ $M_\alpha = (Q_1 \dot\cup Q_2 \dot\cup \{q_0, q_f\}, q_0, \Sigma^n, \rho, q_f, \Psi)$,

$\Psi(q_0) = \Psi(q_f) = \emptyset$, $\Psi(q) = \Psi_i(q)$ for $q \in Q_i$

$$\rho(q, \tau) = \begin{cases} \rho_i(q, \tau) & \text{if } q \in Q_i \\ \emptyset & \text{else} \end{cases}$$

$$\rho(q, \varepsilon) = \begin{cases} \{q_{0,1}, q_{0,2}\} & \text{if } q = q_0 \\ \rho_i(q, \varepsilon) & \text{if } q \in Q_i \setminus \{q_{f,i}\} \\ \rho_i(q, \varepsilon) \cup \{q_f\} & \text{if } q = q_{f,i} \end{cases}$$

$\boldsymbol{\alpha_1 \cdot \alpha_2}$ $M_\alpha = (Q_1 \dot\cup Q_2, q_{0,1}, \Sigma^n, \rho, q_{f,2}, \Psi)$, $\Psi(q) = \Psi_i(q)$ for $q \in Q_i$

$\rho(q, \tau) = \rho_i(q, \tau)$ for $q \in Q_i$

$$\rho(q, \varepsilon) = \begin{cases} \rho_i(q, \varepsilon) & \text{if } q \in Q_i, q \neq \{q_{f,1}\} \\ \rho_1(q, \varepsilon) \cup \{q_{0,2}\} & \text{if } q = q_{f,1} \end{cases}$$

$\boldsymbol{(\alpha_1)^*}$ $M_\alpha = (Q_1 \dot\cup \{q_0, q_f\}, q_0, \Sigma^n, \rho, q_f, \Psi)$,$\Psi(q_0) = \Psi(q_f) = \emptyset$, $\Psi(q) = \Psi_1(q)$, for $q \in Q_1$

$$\rho(q, \tau) = \begin{cases} \rho_1(q, \tau) & \text{if } q \in Q_1 \\ \emptyset & \text{else} \end{cases}$$

$$\rho(q, \varepsilon) = \begin{cases} \rho_1(q, \varepsilon) & \text{if } q \notin \{q_0, q_f, q_{f,1}\} \\ \{q_{0,1}, q_f\} & \text{if } q = q_0 \\ \{q_0\} & \text{if } q = q_f \\ \rho_1(q, \varepsilon) \cup \{q_f\} & \text{if } q = q_{f,1} \end{cases}$$

$\boldsymbol{\psi?}$ $M_\alpha = (\{q_0, q_1, q_2\}, q_0, \Sigma^n, \rho, q_2, \Psi)$,$\Psi(q_0) = \Psi(q_2) = \emptyset$, $\Psi(q_1) = \{\psi\}$

$\rho(q, \tau) = \emptyset$

$$\rho(q_i, \varepsilon) = \begin{cases} \{q_{i+1}\} & \text{if } i = 0, 1 \\ \emptyset & \text{else} \end{cases}$$

Figure 1 Construction of M_α.

Construction of $\mathcal{A}_\varphi$. We construct the ABA $\mathcal{A}_\varphi$ inductively. The alphabet of $\mathcal{A}_\varphi$ is given as $\Sigma_\varphi = S^n \times \Sigma^n$ where n is the number of path quantifiers the formula φ is in scope of. When constructing an automaton for φ with subformulas φ_i, we assume that the automata $\mathcal{A}_{\varphi_i} = (Q_{\varphi_i}, q_{0,\varphi_i}, \Sigma_{\varphi_i}, \rho_{\varphi_i}, F_{\varphi_i})$ are already constructed. Similarly, when a formula contains an expression α, we assume that not only $M_\alpha = (Q_\alpha, q_{0,\alpha}, \Sigma^n, \rho_\alpha, q_{f,\alpha}, \Psi)$ but also $\mathcal{A}_{\psi_i}$ in the case of $\langle\alpha\rangle\varphi$ and $\Delta\alpha$ or $\mathcal{A}_{\bar\psi_i}$ in case of $[\alpha]\varphi$ for each construct $\psi_i?$ in α are already constructed. Here, $\bar\psi_i$ is the negation normal form of $\neg\psi_i$. Recall that states *true* and *false* are always part of an ABA in our definition, so we will not mention them explicitly. Furthermore, let $\mathcal{T} = (S, s_0, \{\delta_\sigma \mid \sigma \in \Sigma\}, L)$ be the KTS to be checked.

The cases of the constructions shown in Figure 2 are straightforward: for a_{π_k} (resp. $\neg a_{\pi_k}$), only those words are accepted where the state first read for path π_k is labelled with a (resp. not labelled with a). For the connectives $\wedge$ and $\vee$, one checks whether both automata $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$ accept (resp. at least one automaton accepts) the word.

$$
\begin{array}{ll}
\boldsymbol{a_{\pi_k}} & \mathcal{A}_\varphi = (\{q_0\}, q_0, \Sigma_\varphi, \rho, \emptyset) \\
& \rho(q_0, (s,\tau)) = \begin{cases} \mathit{true} & \text{if } a \in L(s|_k) \\ \mathit{false} & \text{else} \end{cases} \\
\boldsymbol{\neg a_{\pi_k}} & \mathcal{A}_\varphi = (\{q_0\}, q_0, \Sigma_\varphi, \rho, \emptyset) \\
& \rho(q_0, (s,\tau)) = \begin{cases} \mathit{false} & \text{if } a \in L(s|_k) \\ \mathit{true} & \text{else} \end{cases} \\
\boldsymbol{\varphi_1 \wedge \varphi_2} & \mathcal{A}_\varphi = (Q_1 \dot\cup Q_2 \dot\cup \{q_0\}, q_0, \Sigma_\varphi, \rho, F_1 \dot\cup F_2) \\
& \rho(q, (s,\tau)) = \begin{cases} \rho_1(q_{0,1}, (s,\tau)) \wedge \rho_2(q_{0,2}, (s,\tau)) & \text{if } q = q_0 \\ \rho_i(q, (s,\tau)) & \text{if } q \in Q_i, i \in \{1,2\} \end{cases} \\
\boldsymbol{\varphi_1 \vee \varphi_2} & \mathcal{A}_\varphi = (Q_1 \dot\cup Q_2 \dot\cup \{q_0\}, q_0, \Sigma_\varphi, \rho, F_1 \dot\cup F_2) \\
& \rho(q, (s,\tau)) = \begin{cases} \rho_1(q_{0,1}, (s,\tau)) \vee \rho_2(q_{0,2}, (s,\tau)) & \text{if } q = q_0 \\ \rho_i(q, (s,\tau)) & \text{if } q \in Q_i, i \in \{1,2\} \end{cases}
\end{array}
$$

Figure 2 Construction of $\mathcal{A}_\varphi$ in basic cases.

We now discuss how to handle the PDL-like modalities $\langle\alpha\rangle\varphi$, $[\alpha]\varphi$ and $\Delta\alpha$. In the correctness statement for the automata M_α constructed for this purpose, we rely on oracle requests for tests $\psi?$ (Lemma 5 (iv)). As mentioned, we eliminate these oracle requests by transitioning into the automaton $\mathcal{A}_\psi$ whenever we reach a state marked with $\psi?$.

In the construction for $\langle\alpha\rangle\varphi_1$, we want to recognise a single path where after a prefix satisfying α, φ_1 holds. This is achieved by enabling a move into $\mathcal{A}_{\varphi_1}$ whenever the final state $q_{f,\alpha}$ of M_α is reached. Since none of the states of M_α are declared final in $\mathcal{A}_\varphi$, an accepting run in $\mathcal{A}_\varphi$ cannot stay in M_α forever and thus eventually has to move into $\mathcal{A}_{\varphi_1}$ in this way. Moves into $\mathcal{A}_\psi$ for tests $\psi?$ are made conjunctively since all tests on an accepting run have to be successful. For the dual case $[\alpha]\varphi_1$, we want φ_1 to hold after all prefixes satisfying α. Thus, dual to the previous construction, whenever reaching the state $q_{f,\alpha}$, we are not only given the possibility to, but have to move into $\mathcal{A}_{\varphi_1}$ as well. Since all transitions in this construction are combined with $\wedge$ to ensure that all prefixes satisfying α are considered, we have to take care of paths in $\mathcal{A}_\varphi$ that never leave M_α by declaring all states of M_α accepting. On the other hand, paths not satisfying α due to a violation of a test $\psi?$ are ruled out by a disjunctive test for the negation of ψ. An illustration of these two cases can be found in Figure 4. To handle formulas of the form $\Delta\alpha$, we transform M_α into a Büchi automaton with a distinguished new initial state q_0 which ensures that in between two visits of q_0, α is satisfied. The state q_0 acts like the initial state $q_{0,\alpha}$ for outgoing, and like the final state $q_{f,\alpha}$ for incoming transitions and is the only accepting state (apart from those in the $\mathcal{A}_{\psi_i}$ automata). Moves into test-automtata are handled just as in the $\langle\alpha\rangle\varphi_1$ case. This ensures that an accepted input word consists of repeated segments matched by α. Negated $\Delta\alpha$ constructions can be handled by complementation using Proposition 2.

Note that when we translate state markings in M_α into transitions in $\mathcal{A}_\varphi$, $q \overset{\varepsilon}{\Rightarrow}_X q'$ may hold for exponentially many sets X, resulting in disjunctions of exponential size (in $|\alpha|$) in the transition functions for $\langle\cdot\rangle$ and Δ, and conjunctions of exponential size for $[\cdot]$. This can, however, be avoided by constructing formulas equivalent to these disjunctions and conjunctions during the inductive construction of M_α, the size of which is not larger than $3 \cdot |\alpha| + 2$. We discuss the case of disjunctions; the other case can be handled in a dual way. Two observations are exploited. First of all, the formula need not be written in a disjunctive form but can mix disjunctions and conjunctions freely. Thus, one source of exponential increase can be avoided by constructing the formulas for nested sums and concatenations

$$
\begin{array}{ll}
\langle\boldsymbol{\alpha}\rangle\boldsymbol{\varphi_1} & \mathcal{A}_\varphi = (Q_1 \dot\cup Q_\alpha \dot\cup \bigcup_i Q_{\psi_i}, q_{0,\alpha}, \Sigma_\varphi, \rho, F_1 \dot\cup \bigcup_i F_{\psi_i}) \\
& \rho(q,(s,\tau)) = \begin{cases}
\rho_1(q,(s,\tau)) & \text{if } q \in Q_1 \\
\rho_{\psi_i}(q,(s,\tau)) & \text{if } q \in Q_{\psi_i} \\
\bigvee\{q' \wedge \bigwedge_{\psi_i \in X} \rho_{\psi_i}(q_{0,\psi_i},(s,\tau)) \mid q \overset{\tau}{\Rightarrow}_X q'\} \cup & \\
\quad \{\rho(q_{0,1},(s,\tau)) \wedge & \\
\qquad \bigwedge_{\psi_i \in X} \rho_{\psi_i}(q_{0,\psi_i},(s,\tau)) \mid q \overset{\varepsilon}{\Rightarrow}_X q_{f,\alpha}\} & \text{if } q \in Q_\alpha
\end{cases} \\
[\boldsymbol{\alpha}]\boldsymbol{\varphi_1} & \mathcal{A}_\varphi = (Q_1 \dot\cup Q_\alpha \dot\cup \bigcup_i Q_{\bar\psi_i}, q_{0,\alpha}, \Sigma_\varphi, \rho, F_1 \dot\cup Q_\alpha \dot\cup \bigcup_i F_{\bar\psi_i}) \\
& \rho(q,(s,\tau)) = \begin{cases}
\rho_1(q,(s,\tau)) & \text{if } q \in Q_1 \\
\rho_{\bar\psi_i}(q,(s,\tau)) & \text{if } q \in Q_{\bar\psi_i} \\
\bigwedge\{q' \vee \bigvee_{\psi_i \in X} \rho_{\bar\psi_i}(q_{0,\bar\psi_i},(s,\tau)) \mid q \overset{\tau}{\Rightarrow}_X q'\} \cup & \\
\quad \{\rho(q_{0,1},(s,\tau)) \vee & \\
\qquad \bigvee_{\psi_i \in X} \rho_{\bar\psi_i}(q_{0,\bar\psi_i},(s,\tau)) \mid q \overset{\varepsilon}{\Rightarrow}_X q_{f,\alpha}\} & \text{if } q \in Q_\alpha
\end{cases} \\
\boldsymbol{\Delta\alpha} & \mathcal{A}_\varphi = (Q_\alpha \dot\cup \{q_0\} \dot\cup \bigcup_i Q_{\psi_i}, q_0, \Sigma_\varphi, \rho, \{q_0\} \dot\cup \bigcup_i F_{\psi_i}) \\
& \rho(q,(s,\tau)) = \begin{cases}
\rho_{\psi_i}(q,(s,\tau)) & \text{if } q \in Q_{\psi_i} \\
\bigvee\{q' \wedge \bigwedge_{\psi_i \in X} \rho_{\psi_i}(q_{0,\psi_i},(s,\tau)) \mid q_{0,\alpha} \overset{\tau}{\Rightarrow}_X q'\} \cup & \\
\quad \{\bigwedge_{\psi_i \in X} \rho_{\psi_i}(q_{0,\psi_i},(s,\tau)) \mid q_{0,\alpha} \overset{\varepsilon}{\Rightarrow}_X q_{f,\alpha}\} & \text{if } q = q_0 \\
\bigvee\{q' \wedge \bigwedge_{\psi_i \in X} \rho_{\psi_i}(q_{0,\psi_i},(s,\tau)) \mid q \overset{\tau}{\Rightarrow}_X q'\} \cup & \\
\quad \{q_0 \wedge \bigwedge_{\psi_i \in X} \rho_{\psi_i}(q_{0,\psi_i},(s,\tau)) \wedge & \\
\qquad \bigwedge_{\psi_i \in Y} q_{0,\psi_i} \mid q \overset{\tau}{\Rightarrow}_X q' \overset{\varepsilon}{\Rightarrow}_Y q_{f,\alpha}\} & \text{if } q \in Q_\alpha
\end{cases} \\
\neg\boldsymbol{\Delta\alpha} & \mathcal{A}_\varphi = \overline{\mathcal{A}_{\Delta\alpha}}
\end{array}
$$

Figure 3 Construction of $\mathcal{A}_\varphi$ for α formulas.

inductively using that their contribution is directly captured by disjunction and conjunction, respectively. The second observation is that the conjunction resulting from a subpath of a path subsumes the conjunction for that path. This can be exploited to show that only paths using backwards edges (i.e., edges from q_f to q_0 in *-constructions) at most once and using only particular backward edges have to be considered for treating iteration. We refer the interested reader to Appendix B for a more detailed look at this alternative construction.

In the constructions for path quantifiers (Figure 5), we eliminate one component of the alphabet $\Sigma_{\varphi_1} = S^{n+1} \times \Sigma^{n+1}$ and switch to $\Sigma_\varphi = S^n \times \Sigma^n$. The eliminated component is now simulated by the state space of $\mathcal{A}_\varphi$, which ensures that said component is indeed a path in $\mathcal{T}$ by using the additional components from S and Σ. The rule for the initial state guarantees that this path from $\mathcal{T}$ starts in the state it is branching from. Negated existential quantifiers that by our definition can occur in formulas in negation normal form are handled straightforwardly by complementation.

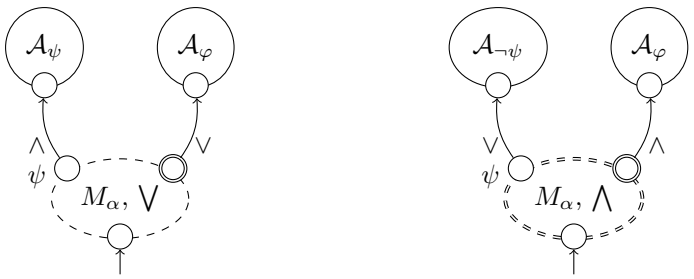


Figure 4 Illustration of the constructions for $\langle\alpha\rangle\varphi$ and $[\alpha]\varphi$. In automata shown with a single dashed line all states are non-final. In automata shown with a double dashed line all states are final.

$\exists \pi.\varphi_1$ $\quad \mathcal{A}_{\varphi_1}$ dealternised: $MH(\mathcal{A}_{\varphi_1}) = (Q_1, q_{0,1}, \Sigma_{\varphi_1}, \rho_1, F_1)$
$\mathcal{A}_\varphi = (Q_1 \times S \times \Sigma \dot\cup \{q_0\}, q_0, \Sigma_\varphi, \rho, F_1 \times S \times \Sigma)$
$\rho(q_0, (s,\tau)) = \{(q', s', \sigma') \mid q' \in \rho_1(q_{0,1}, s + s|_n, \tau + \sigma), s' \in \delta_\sigma(s|_n), \sigma, \sigma' \in \Sigma\}$
$\rho((q,s,\sigma),(s,\tau)) = \{(q', s', \sigma') \mid q' \in \rho_1(q, s + s, \tau + \sigma), s' \in \delta_\sigma(s), \sigma' \in \Sigma\}$

$\neg\exists \pi.\varphi_1$ $\quad \mathcal{A}_\varphi = \overline{\mathcal{A}_{\exists \pi.\varphi_1}}$

Figure 5 Construction of $\mathcal{A}_\varphi$ for quantifier formulas.

For the construction, we obtain the following theorem via induction:

▶ **Theorem 6.** *The automaton $\mathcal{A}_\varphi$ is $\mathcal{T}$-equivalent to φ.*

For the complexity analysis, we introduce a notion of criticality.

▶ **Definition 7** (Criticality). *The* criticality *of a HyperPDL-Δ formula φ in negation normal form equals the highest number of critical quantifiers along any path in the formula's syntax tree. A quantifier is called* critical *iff it is a non-outermost quantifier, fulfills at least one of the following three conditions:*

(i) *it is a negated quantifier,*
(ii) *it is an outermost quantifier in a test $\varphi?$ in some program α,*
(iii) *it is an outermost quantifier in the subformula φ of $[\alpha]\varphi$,*

and is not a negated outermost quantifier in a test $\varphi?$ occuring in a modality $[\alpha]$ where the automaton M_α is deterministic.
We call a HyperPDL-Δ formula with criticality 0 uncritical.

This definition is designed carefully in order to ensure that the alternation depth of HyperCTL* formulas coincides with the criticality of their direct translation to HyperPDL-Δ. Intuitively, an *uncritical* quantifier is one where the next dealternation construction $MH(\mathcal{A})$ does not cause another exponential blowup on this part of the automaton. Accordingly, a critical quantifier is one where this exponential blowup for the next dealternation construction cannot be avoided in general. Thus, the criticality of a formula accounts for the number of times an exponential blowup may happen during the construction.

Since exponential blowups occur in a nested manner, we can only bound the size of the resulting automaton by an exponential tower. As argued in the last paragraph, its height is determined by criticality rather than quantifier depth. Formally, we define a function g as $g_{p,c}(0,n) = p(n)$ and $g_{p,c}(k+1,n) = c^{g(k,n)}$ for a constant $c > 1$ and a polynomial p. We use $\mathcal{O}(g(k,n))$ as an abbreviation for $\mathcal{O}(g_{p,c}(k,n))$ for some $c > 1$ and polynomial p.

Two remarks are in order about the next Lemma. Firstly, the statement refers to the number of states of the construction, disregarding the number of transitions. However, this is harmless for our complexity analysis: while the alphabet size increases exponentially with the nesting depth of quantifiers, alphabets need not be represented explicitly and the number of transitions of the automata can be kept polynomial by delaying the substitution of the wildcard symbol $\cdot$ by concrete programs until the intersection with the system $\mathcal{T}$. We refrain from explicating this in our construction in order to increase readability. Secondly, the construction's size can also increase exponentially in the nesting level of negated $\Delta\alpha$ constructions. Since we expect that negated $\Delta\alpha$ constructions are rarely nested in formulas, we assume for the remainder of this paper a bound on this nesting level in order to simplify our complexity statement. Indeed, in Section 6, we will show that a bound of 1 suffices to express ω-regular properties, thus making this a reasonable constraint. The dependency from the nesting depth is reflected in the proof of Lemma 8 in Appendix A.

▶ **Lemma 8.** *The automaton $\mathcal{A}_\varphi$ has size $\mathcal{O}(g(k+1, |\varphi| + log(|\mathcal{T}|)))$ for formulas φ with criticality k.*

Proof (Sketch). By induction on the criticality k. In the base case, the dealternation constructions from Proposition 1 overall cause a single exponential blowup at most since in all constructions not increasing the criticality, one has to keep track of only a single state of each already dealternised subautomaton in further dealternations. Since the only dealternation is done before the system $\mathcal{T}$ is folded around the automaton, the automaton's size is only exponential in the size of φ, but not in the size of $\mathcal{T}$.

In the inductive step, the construction for the outermost critical quantifier increases the size of the automaton exponentially. For all further constructions, it can be argued that the automaton's size is asymptotically not further increased, just like in the base case. ◀

▶ **Theorem 9.** *The problem to decide whether $\mathcal{T} \models \varphi$ holds for KTS $\mathcal{T}$ and HyperPDL-Δ formulas φ with criticality k is in NSPACE$(g(k, |\varphi| + log(|\mathcal{T}|))$.*

Proof. The formula arising from the transformation of φ to our variant of negation normal form is a boolean combination of subformulas ψ with an outermost existential quantifier. Due to dealternation for the existential quantifier, the automata $\mathcal{A}_\psi$ are non-deterministic Büchi automata. We perform nonemptiness checks on these automata separately. It is well-known that the nonemptiness check for Büchi automata is possible in NLOGSPACE in the size of the automaton [9]. We then combine the results in accordance with the structure of φ. This does not add to the complexity. Thus, by Lemma 8, we obtain an NSPACE$(g(k, |\varphi| + log(|\mathcal{T}|))$ model checking algorithm for criticality k HyperPDL-Δ formulas. ◀

Since we can easily translate HyperCTL* formulas to HyperPDL-Δ while preserving the alternation depth as criticality, we can use known hardness results for HyperCTL* model checking [20] to obtain the following Theorem:

▶ **Theorem 10.** *Given a KTS $\mathcal{T}$ and a HyperPDL-Δ formula φ with criticality k, the model checking Problem for HyperPDL-Δ is hard for NSPACE$(g(k, |\varphi|))$ and NSPACE$(g(k-1, |\mathcal{T}|))$.*[1]

5 Satisfiability

While model checking of temporal logics is an essential technique for verification, it requires meaningful specifications in order to be useful. For example, if a formula is fulfilled by every or no structure, it is useless for specification purposes. To evaluate whether this is the case, satisfiability testing can be employed as a sanity check [21]. Since satisfiability checking is already undecidable for HyperLTL [11] via a reduction from the Post Correspondence Problem (PCP) and HyperLTL can be embedded into HyperPDL-Δ, we consider only restrictions of a fragment of HyperPDL-Δ where quantified paths can be traversed linearly for the purpose of this section.

[1] Note that we have not defined $g(-1, n)$ in this paper. For $k = 0$ and a fixed size formula φ, we use the definition from [12], where NSPACE$(g(-1, n))$ was defined as NLOGSPACE. Since one can see that our algorithm has NLOGSPACE complexity in this instance, the lower and upper bounds match in all cases. For $k = 0$ and a fixed size structure $\mathcal{T}$ or $k > 1$, we can use Savitch's Theorem to see that the problems are actually complete for the deterministic space classes.

▶ **Definition 11.** *A linear HyperPDL-Δ formula is of the form* $Q_1\pi_1 \dots Q_n\pi_n.\psi$ *for* $Q_i \in \{\exists, \forall\}$ *and* ψ *a quantifier-free formula. A linear HyperPDL-Δ formula* φ *is an* $\forall^*$*-formula if* $Q_i = \forall$ *for all* $1 \leq i \leq n$*. The* $\exists^*$*-fragment and the* $\exists^*\forall^*$*-fragment are defined analogously.*

For linear HyperPDL-Δ, we generalise the semantics to arbitrary sets of traces T instead of only those induced by KTS. More concretely, for a fixed set of traces T, we let the assignment functions Π map variables to traces in T instead of paths of a structure. Such a function Π is called a *trace assignment*; the set of all trace assignments is called TA. Furthermore, we let all quantifiers range over T instead of $Paths(\mathcal{T}, \Pi(\epsilon)(0))$. We further define $T \models \varphi$ to hold for a linear HyperPDL-Δ formula φ iff $\{\} \models \varphi$ holds for the trace assignment $\{\}$ with empty domain over T. For linear HyperPDL-Δ formulas and using $T = Traces(\mathcal{T}, s_0)$, this definition coincides with the definition in Section 3. We then call a linear HyperPDL-Δ formula φ *satisfiable* iff there is a non-empty set of traces T such that $T \models \varphi$ holds. The satisfiability problem for linear HyperPDL-Δ is to check whether a linear HyperPDL-Δ formula φ is satisfiable. For full linear HyperPDL-Δ, we obtain undecidability via a reduction from HyperLTL satisfiability [11]:

▶ **Theorem 12.** *The satisfiability problem for linear HyperPDL-Δ is undecidable.*

However, since the encoding relies on the availability of arbitrary combinations of quantifiers, a natural question is whether fragments of linear HyperPDL-Δ with restricted quantifier combinations yield a decidable satisfiability problem. For HyperLTL, several such restrictions were considered [11]. In [11], the satisfiability problem for the restricted fragments in HyperLTL is solved via the transformation of a HyperLTL formula to an equisatisfiable LTL formula and solving the satisfiability problem for the latter. Since there is no apparent connection of this form between HyperPDL-Δ and LTL, we use a similar type of translation, but instead translate our formulas into suitable ABA and check those for emptiness. In all cases, the lower complexity bound can be obtained via reduction from the satisfiability problem of the corresponding fragment of HyperLTL.

▶ **Theorem 13.** *The satisfiability problem for the* $\forall^*$*-fragment of HyperPDL-Δ is PSPACE-complete.*

Proof (Sketch). Let φ be a $\forall^*$-fragment formula, i.e. $\varphi \equiv \forall\pi_1 \dots \forall\pi_n.\psi$ for a quantifier-free formula ψ. We manipulate ψ by substituting $\pi_1, ..., \pi_n$ with a single fresh variable π and obtain a formula ψ'. This is done by (i) replacing every occurence of an atomic proposition a_{π_i} with a_π and (ii) compressing each tuple of atomic programs τ into a program α with only 1-tuples. The compression discriminates three cases. If τ only consists of wildcard programs, i.e. $\tau = \bullet$, then it is compressed to $(\cdot)$. If τ is composed from the set $\{\sigma, \cdot\}$ for some atomic program σ, then it is compressed to (σ). Otherwise, i.e. if τ is composed of distinct non-wildcard atomic programs, it is compressed to $\mathit{false}? \cdot (\cdot)$. For example, for $\varphi \equiv \forall\pi_1\forall\pi_2\langle(\cdot, \sigma)\rangle a_{\pi_1} \wedge [\bullet + (\sigma, \sigma')]\neg a_{\pi_2}$, ψ' is given by $\langle(\sigma)\rangle a_\pi \wedge [(\cdot) + \mathit{false}? \cdot (\cdot)]\neg a_\pi$. Let $\mathcal{A}$ be the ABA for ψ' as described in Section 4. We can test $\mathcal{A}$ for emptiness in PSPACE [9]. $\mathcal{A}$ is non-empty iff φ is satisfiable: any word w accepted by $\mathcal{A}$ gives rise to the trace set $\{w\}$ satisfying φ. Analogously, a non-empty trace set T with $T \models \varphi$ can be used to obtain a word accepted by $\mathcal{A}$ since $\mathcal{A}$ accepts all traces satisfying ψ' and we can thus pick any trace t in T as a witness.

The lower bound directly follows by a reduction from the satisfiability problem for the $\exists^*\forall^*$ fragment of HyperLTL [11]. ◀

Similarly, we can show that the $\exists^*$ fragment is also PSPACE-complete.

▶ **Theorem 14.** *The satisfiability problem for the $\exists^*$-fragment of HyperPDL-Δ is PSPACE-complete.*

Proof (Sketch). Let φ be a $\exists^*$-fragment formula, i.e. $\varphi \equiv \exists \pi_1 \dots \exists \pi_n . \psi$ for a quantifier-free formula ψ. We construct the alternating Büchi automaton $\mathcal{A}$ for ψ. Non-emptiness of $\mathcal{A}$ and satisfiability of φ are equivalent: every trace set T fulfilling φ contains traces $t_1 \dots t_n$ fulfilling ψ and these give rise to a word accepted by $\mathcal{A}$. On the other hand, if $\mathcal{L}(\mathcal{A})$ is non-empty, the trace set T induced by an arbitrary $w \in \mathcal{L}(\mathcal{A})$ fulfills φ.

The lower bound follows straightforwardly by a reduction from the satisfiability problem for the $\exists^*$ fragment of HyperLTL [11]. ◀

For the $\exists^*\forall^*$-fragment, we eliminate the universal quantifiers by taking the variables bound by existential quantifiers and replacing the variables bound by universal quantifiers by all possible combinations of them. Unlike the previous two fragments, this increases the complexity beyond PSPACE.

▶ **Theorem 15.** *The satisfiability problem is EXPSPACE-complete for the $\exists^*\forall^*$-fragment of HyperPDL-Δ.*

Proof (Sketch). Let φ be a $\exists^*\forall^*$-formula, i.e. $\varphi \equiv \exists \pi_1 \dots \exists \pi_n \forall \pi'_1 \dots \forall \pi'_m . \psi$ for a quantifier-free formula ψ. For a formula ψ and path variables π, π', we define the substitution $\psi[\pi/\pi']$ to be the variant of ψ in which all occurences of π' have been replaced by π similar to the substitution in the proof of Theorem 13. The main difference here is that only two instead of n path variables are compressed into one. Thus only two instead of all atomic programs have to be considered when determining the replacement of a tuple. Let $\varphi' \equiv \exists \pi_1 \dots \exists \pi_n . \bigwedge_{j_1=1}^{n} \cdots \bigwedge_{j_m=1}^{n} \psi[\pi_{j_1}/\pi'_1] \dots [\pi_{j_m}/\pi'_m]$. For example, for $\varphi \equiv \exists \pi_1 . \exists \pi_2 . \forall \pi'_1 . \langle (\cdot, \cdot, \sigma) \rangle a_{\pi_1} \wedge \neg a_{\pi_2} \wedge a_{\pi'_1}$, $\varphi' = \exists \pi_1 . \exists \pi_2 . (\langle (\sigma, \cdot) \rangle a_{\pi_1} \wedge \neg a_{\pi_2} \wedge a_{\pi_1}) \wedge (\langle (\cdot, \sigma) \rangle a_{\pi_1} \wedge \neg a_{\pi_2} \wedge a_{\pi_2})$. φ' is an $\exists^*$ formula and is equisatisfiable to φ: any trace assignment satisfying φ naturally induces a model of φ'. For the reverse direction, assume $\Pi \models \varphi'$. φ' contains all possible combinations of assignments for the variables $\pi'_1 \dots \pi'_m$ with traces chosen for the existentially quantified variables $\pi_1 \dots \pi_n$. Then $T = \{\Pi(\pi_i) \mid 1 \leq i \leq n\} \models \varphi$. φ' is constructible in EXPTIME. Therefore the satisfiability check is possible in EXPSPACE due to Theorem 14.

The lower bound easily follows by a reduction from the satisfiability problem for the $\exists^*\forall^*$ fragment of HyperLTL [11]. ◀

As in [11], from Theorem 15, we obtain that it is an EXPSPACE-complete problem to decide whether one uncritical HyperPDL-Δ formula is implied by another. In particular, the uncritical fragment includes properties like variants of observational determinism enriched by regular predicates.

6 Expressivity Results

As mentioned in the introduction, a desirable property of temporal logics is the ability to specify arbitrary ω-regular properties. We show that HyperPDL-Δ indeed has this property.

▶ **Theorem 16.** *Let Π be a trace assignment and $\pi_1 \dots \pi_n$ be the variables bound by Π. Let $\nu_{AP} : TA \to ((2^{AP})^n \times \Sigma^n)^\omega$ be the analog of ν for trace assignments. For a given ω-regular language $\mathcal{L}$ over $(2^{AP})^n \times \Sigma^n$, there is a quantifier-free HyperPDL$-\Delta$ formula φ with path variables $\pi_1 \dots \pi_n$ such that $\Pi \models \varphi$ iff $\nu_{AP}(\Pi) \in \mathcal{L}$.*

Proof. Let $\mathcal{L}$ be an ω-regular language over $(2^{AP})^n \times \Sigma^n$. It is well-known [9] that $\mathcal{L} = \bigcup_{i=1}^{k} \mathcal{L}_{i,0}\mathcal{L}_{i,1}^{\omega}$ holds for some regular languages $\mathcal{L}_{i,0}$, $\mathcal{L}_{i,1}$. Let $r_{i,j}$ be a regular expression for $\mathcal{L}_{i,j}$. Every symbol in $r_{i,j}$ has the form $((P_1, ..., P_n), \tau)$ for $P_k \subseteq AP$ and $\tau \in \Sigma^n$. Let $\alpha_{i,j}$ be the regular expression obtained by replacing each such symbol in $r_{i,j}$ by $(\bigwedge_{l=1}^{n} \bigwedge_{a \in P_l} a_{\pi_l} \wedge \bigwedge_{a \notin P_l} \neg a_{\pi_l})? \cdot \tau$. Then $\varphi \equiv \bigvee_{i=1}^{k} \langle \alpha_{i,0} \rangle \Delta \alpha_{i,1}$ yields the desired formula. ◀

It follows from this theorem that HyperPDL-Δ can express an infinitary version of the regular hyperlanguages recently proposed in [3].

We now compare our logic to other hyperlogics. For this purpose we introduce a logic that adds to HyperCTL* the ability to quantify over atomic propositions.

▶ **Definition 17** ([7]). *The logic HyperQCTL* is obtained by adding to the syntax of HyperCTL* the rules $\varphi ::= q \mid \exists q.\varphi$ and to the semantics the rules*

$$\Pi \models_{\mathcal{T}} q \quad \text{iff} \quad q \in \Pi(\pi_q)(0)$$
$$\Pi \models_{\mathcal{T}} \exists q.\varphi \quad \text{iff} \quad \exists t \in (2^{\{q\}})^{\omega}.\Pi[\pi_q \to t] \models_{\mathcal{T}} \varphi$$

The sub-logic HyperQPTL consists of the HyperQCTL* formulas where both path quantifiers $Q\pi.\varphi$ and propositional quantifiers $Qq.\varphi$ only occur at the front of the formula.

▶ **Theorem 18.**
1. *HyperCTL* $<$ HyperPDL-Δ $\leq$ HyperQCTL**
2. *HyperLTL $<$ Linear HyperPDL-Δ $\leq$ HyperQPTL*

Proof. Part one of both claims is straightforward: Embedding HyperLTL and HyperCTL* into (linear) HyperPDL-Δ works as described in Section 3. An embedding in the other direction is impossible due to the inability of HyperLTL and HyperCTL* to express arbitrary ω-regular properties [20].

For the second part of the first claim, we observe that the semantics of HyperPDL-Δ can straightforwardly be encoded in MSO[E], which is equally expressive as HyperQCTL* by [7]. The last claim can be shown by a direct translation via Büchi automata: A quantifier-free HyperPDL-Δ formula can be translated into a Büchi Automaton as described in Section 4. By [15], there is a QPTL formula for that automaton, where the quantifiers can be reattached to yield the desired formula. ◀

By the results of [7], we obtain that linear HyperPDL-Δ is stricly less expressive than S1S[E], which, as mentioned, has an undecidable model checking problem. We leave a more precise localisation of linear and unrestricted HyperPDL-Δ in the hierarchies of hyperlogics from [7] for future work. This includes comparisons with FO[<,E] and MPL[E] and an answer to the question if the second inequalities from the claims in Theorem 18 are indeed strict.

7 Conclusion

We introduced the logic HyperPDL-Δ as a variant of Propositional Dynamic Logic for hyperproperties that can express all ω-regular properties. Our model checking algorithm has the same complexity as model checking HyperCTL*, despite the increased expressive power. Finally, we showed that satisfiability checking for certain fragments has the same complexity as for structurally similar, but less expressive fragments of HyperLTL.

Future work includes implementing a model checker for HyperPDL-Δ. It would also be interesting to explore alternative model checking and satisfiability testing algorithms for subfragments of HyperPDL-Δ, possibly by exploiting classical techniques for PDL.

References

1 Roy Armoni, Limor Fix, Alon Flaisher, Rob Gerth, Boris Ginsburg, Tomer Kanza, Avner Landver, Sela Mador-Haim, Eli Singerman, Andreas Tiemeyer, et al. The ForSpec temporal logic: A new temporal property-specification language. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 296–311. Springer, 2002. doi:10.1007/3-540-46002-0_21.

2 Borzoo Bonakdarpour, Cesar Sanchez, and Gerardo Schneider. Monitoring hyperproperties by combining static analysis and runtime verification. In *International Symposium on Leveraging Applications of Formal Methods*, pages 8–27. Springer, 2018. doi:10.1007/978-3-030-03421-4_2.

3 Borzoo Bonakdarpour and Sarai Sheinvald. Automata for hyperlanguages, 2020. arXiv:2002.09877.

4 Laura Bozzelli, Bastien Maubert, and Sophie Pinchinat. Unifying hyper and epistemic temporal logics. In *FoSSaCS*, volume 9034 of *Lecture Notes in Computer Science*, pages 167–182. Springer, 2015. doi:10.1007/978-3-662-46678-0_11.

5 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *POST 2014*, pages 265–284, 2014. doi:10.1007/978-3-642-54792-8_15.

6 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.

7 Norine Coenen, Bernd Finkbeiner, Christopher Hahn, and Jana Hofmann. The hierarchy of hyperlogics. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13, 2019. doi:10.1109/LICS.2019.8785713.

8 Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013. URL: http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6997.

9 Stéphane Demri, Valentin Goranko, and Martin Lange. *Temporal Logics in Computer Science: Finite-State Systems*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016. doi:10.1017/CBO9781139236119.

10 Peter Faymonville and Martin Zimmermann. Parametric linear dynamic logic. *Information and Computation*, 253:237–256, 2017. GandALF 2014. doi:10.1016/j.ic.2016.07.009.

11 Bernd Finkbeiner and Christopher Hahn. Deciding hyperproperties. In *CONCUR 2016*, pages 13:1–13:14, 2016. doi:10.4230/LIPIcs.CONCUR.2016.13.

12 Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking HyperLTL and HyperCTL*. In *CAV 2015*, pages 30–48, 2015. doi:10.1007/978-3-319-21690-4_3.

13 Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. Comput. Syst. Sci.*, 18(2):194–211, 1979. doi:10.1016/0022-0000(79)90046-1.

14 Joseph Y. Halpern, Ron van der Meyden, and Moshe Y. Vardi. Complete axiomatizations for reasoning about knowledge and time. *SIAM J. Comput.*, 33(3):674–703, 2004. doi:10.1137/S0097539797320906.

15 Yonit Kesten and Amir Pnueli. Complete proof system for QPTL. *J. Log. Comput.*, 12(5):701–745, 2002. doi:10.1093/logcom/12.5.701.

16 Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. Extended temporal logic revisited. In *CONCUR 2001*, pages 519–535, 2001. doi:10.1007/3-540-44685-0_35.

17 Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Logic*, 2(3):408–429, July 2001. doi:10.1145/377978.377993.

18 Martin Lange. Model checking propositional dynamic logic with all extras. *J. Applied Logic*, 4(1):39–49, 2006. doi:10.1016/j.jal.2005.08.002.

19 Satoru Miyano and Takeshi Hayashi. Alternating finite automata on omega-words. *Theoretical Computer Science*, 32(3):321–330, 1984. doi:10.1016/0304-3975(84)90049-5.

20 Markus N. Rabe. *A temporal logic approach to Information-flow control*. PhD thesis, Saarland University, 2016. doi:10.22028/D291-26650.

21 Kristin Y. Rozier and Moshe Y. Vardi. LTL satisfiability checking. In *SPIN 2007*, pages 149–167, 2007. doi:10.1007/978-3-540-73370-6_11.

22 Pierre Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1/2):72–99, 1983. doi:10.1016/S0019-9958(83)80051-5.

A Detailed Complexity Analysis

Proof of Lemma 8. By induction on the criticality k.

Base case: φ has criticality 0. We show inductively that $\mathcal{A}_\varphi$ has size $2^{\mathcal{O}(p(n)+p'(log(m)))}$ in the size n of φ and the size m of $\mathcal{T}$ for some polynomials p, p'. First, notice that $|M_\alpha|$ is linear in the size of α. This can easily be shown by a structural induction, where each construction adds a constant number of states to its subautomata only.

Basic constructions $\mathcal{A}_{a_\pi}$ and $\mathcal{A}_{\neg a_\pi}$ have constant size. For boolean connectives as well as $\langle\alpha\rangle\varphi, [\alpha]\varphi$ and $\Delta\alpha$, the construction of $\mathcal{A}_\varphi$ again just adds a constant number of states to the automata for the subformulas. The construction for $\neg\Delta\alpha$ introduces a quadratic increase by Proposition 2. Throughout the whole construction, this results in an exponential increase in the nesting depth of negated $\Delta\alpha$ constructs at most. More precisely, when bounding this nesting depth to a constant d, the polynomial p on top of the exponential tower has degree at most $2d$. Existential quantifiers increase the size of the automaton exponentially in the size of the formula φ and add a factor polynomial in the size of the structure $\mathcal{T}$. Using logarithmic laws, this translates to the form above. Note that the factor depending on $|\mathcal{T}|$ is added after the exponential blowup from the dealternation construction $MH(\mathcal{A})$.

It remains to show that the dealternation construction $MH(\mathcal{A})$ introduces an exponential blowup of the structure's size at most once for formulas of criticality 0, regardless of how many quantifiers the formula contains. In order to do this, we have to look closer at the proof of Proposition 1. We show that once the dealternation construction $MH(\mathcal{A})$ is done for the innermost quantifiers, at most one state of each dealternised automaton has to be tracked in further dealternations. Thus, the exponential size of the subautomaton is added as a factor rather than in an exponent when determining the size of the state space of the full automaton.

Our claim can be shown by an induction over the number of constructions on top of the dealternised automaton. In the base case, no construction is done on top of a dealternised automaton $\mathcal{A}_\varphi$. Since $\mathcal{A}_\varphi$ is a Büchi automaton, a run of the resulting automaton is a path rather than a tree on every word. Thus, only one state has to be tracked. In the inductive step, we discriminate cases for the outermost construction. By the induction hypothesis, at most one state of each dealternised automaton has to be tracked in each subautomaton. For the construction $\varphi_1 \vee \varphi_2$, a run tree moving into $\mathcal{A}_{\varphi_1}$ or $\mathcal{A}_{\varphi_2}$ never returns to the initial state. Thus, since $\mathcal{A}_{\varphi_1}$ and $\mathcal{A}_{\varphi_2}$ are unconnected, we track states of only one of the automata. Then, the claim is implied by the induction hypothesis. The construction for $\mathcal{A}_{\varphi_1 \wedge \varphi_2}$ works similarly, with the difference that we have to track states of both subautomata when a run moves into this automaton over the initial state. This does, however, not lead to an increase in states of each dealternised automaton that have to be tracked, since these subautomata are unconnected. The next construction we have to consider is $\mathcal{A}_{\langle\alpha\rangle\varphi}$. Here, a run has the property that at most one state of M_α has to be tracked, which can be replaced by states of $\mathcal{A}_\varphi$ at some point. Additionally, arbitrarily many states of $\mathcal{A}_\psi$ for subformulas ψ of α can be tracked. However, this is no contradiction to our claim, since α may not

contain any quantified subformulas and thus $\mathcal{A}_\psi$ may not contain dealternised automata in uncritical formulas. Thus, since the induction hypothesis states that at most one state of each dealternised automaton of $\mathcal{A}_\varphi$ has to be tracked at any point, this shows our claim. As another case, we consider the construction for $\exists\pi.\varphi$. Here, since only a disjunctive transition is added on top of $\mathcal{A}_\varphi$, we can argue similar as in the case for $\varphi_1 \vee \varphi_2$ with the difference that we consider only a single subautomaton. Due to the exemption rule in the definition of criticality, there is an additional construction to be considered: $[\alpha]\varphi$, where α is deterministic and the outermost quantifier inside α is negated.[2] Since tests ψ in α are handled by disjunctively transitioning into the automaton for $\neg\psi$ in the $[\alpha]\varphi$ construction, this cancels out the negation of the quantifier. Therefore, no critical negation construction has to be performed on a dealternised subautomaton during the construction of $\mathcal{A}_{\neg\psi}$. Then, since due to the fact that M_α is deterministic, conjunctions of transitions in M_α behave the same as disjunctions and we can argue just as in the case for $\mathcal{A}_{\langle\alpha\rangle\varphi}$. Finally, observe that we do not have to consider constructions for $\Delta\alpha$, $\neg\Delta\alpha$, or general $[\alpha]\varphi$, since the resulting formula is not uncritical when any of these contain a quantified subformula.

Inductive step: φ has criticality $k+1$. On the path in φ's syntax tree inducing the criticality, we inspect the outermost critical quantifier. Its subformulas φ_i have criticality at most k. Using the induction hypothesis on all subformulas, we obtain automata of size at most $\mathcal{O}(g(k+1, |\varphi_i|+log(|\mathcal{T}|)))$. The next dealternation will result in an automaton of size $2^{\mathcal{O}(|\mathcal{A}_{\varphi_i}|)}$ (by Proposition 1) which can be bounded by $2^{\mathcal{O}(g(k+1,|\varphi_i|+log(|\mathcal{T}|)))} = \mathcal{O}(g(k+2, |\varphi|+log(|\mathcal{T}|)))$. Since we inspected the outermost critical quantifier on the path inducing the criticality of the formula, any of the subsequent constructions will not cause a further exponential blowup of the automaton's size, as argued in the base case. ◀

B Alternative Construction for the Transition Function of $\mathcal{A}_\varphi$

In the main body of the paper, we have argued that the transition function in $\mathcal{A}_\varphi$ for φ containing α can be exponential in the size of α. Consider the automaton in Figure 6 where states annotated with $\psi_{i,j}$ are marked with the corresponding formula and each unannotated edge stands for an ε-transition. Such an automaton can occur when α has the form $(\psi_{0,1}? + \psi_{0,2}?)(\psi_{1,1}? + \psi_{1,2}?)...(\psi_{n,1}? + \psi_{n,2}?)\tau$ and should serve as an illustrative example. We consider the case where this α is used in a $\langle . \rangle$ formula. For ease of presentation, we assume that $\psi_{i,j}$ can be tested by moving into a state $p_{i,j}$. It is possible to reach q_f from q_0 with an exponential number of ε-paths, each with a different combination of markings. Therefore we have $q_0 \overset{\tau}{\Rightarrow}_X q$ for exponentially many X, transferring into the size of the transition function when constructing $\rho(q_0, \tau) \equiv \bigvee\{q \wedge \bigwedge_{\psi_{i,j}\in X} p_{i,j} \mid q_0 \overset{\tau}{\Rightarrow}_X q\}$.

For this example, it is easy to see that these exponentially many different combinations of transitions could equally be represented by a formula of a much smaller size, namely $(p_{0,1} \vee p_{0,2}) \wedge (p_{1,1} \vee p_{1,2}) \wedge ... \wedge (p_{n,1} \vee p_{n,2})$. This is due to the fact that conjunction and disjunction closely resemble the behaviour of concatenation and sum constructions in M_α when considering ε-paths for the construction of ρ. We will show here, that using these ideas it is possible to construct such a formula of size not greater than $3 \cdot |\alpha| + 2$ for every α.

[2] There are additional forms of α, where explosion through a negated quantifier inside the modality $[\alpha]$ can be avoided. This includes all forms where in any run in M_α, a test for $\neg\psi$ occurs only in a situation where all states occurring at the same level of the run can transition into $\mathcal{A}_{\neg\psi}$. Then, when a transition into $\mathcal{A}_{\neg\psi}$ can be taken in one of the states, it can be taken in all of the states. Since they are on the same level, the same continuation in $\mathcal{A}_{\neg\psi}$ can be used for all these branches, such that only a single state of each dealternised subautomaton of $\mathcal{A}_{\neg\psi}$ needs to be tracked.

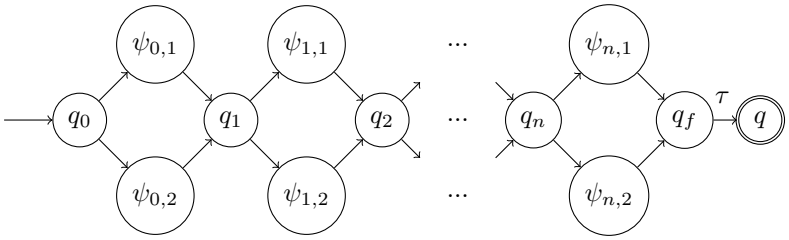


Figure 6 Automaton M_α with an exponential number of test combinations.

We proceed by constructing a function εp such that $\varepsilon p(q, q', \alpha) = \vartheta$ for a formula ϑ equivalent to $\bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\}$ for $\overset{\varepsilon}{\Rightarrow}_X$ constructed from M_α. Here we use variables v_i as placeholders for formulas ψ_i to be later replaced by a transition into the corresponding automaton $\mathcal{A}_{\psi_i}$. For τ-transitions, this can straightforwardly be extended to a function τp which can then be used to construct the τ transition function for a state q in a more succinct way. For a $\langle\alpha\rangle\varphi$ formula and some $q \in Q_\alpha$ we then have $\rho(q, (s, \tau)) = \bigvee\{q' \wedge \tau p(q, q', \alpha)[\rho_{\psi_i}(q_{0,\psi_i}, (s, \tau))/v_i] \mid q' \in Q\}$. Some remarks are in order for this transition function construction: (i) in this construction, opposed to the one used before, each $q' \in Q$ can occur at most once in the disjunction, (ii) for $q' \in Q$ such that there is no X with $q \overset{\tau}{\Rightarrow}_X q'$, i.e. q' is not reachable from q with τ, we have $\tau p(q, q', \alpha) \equiv \mathit{false}$ and thus the state can be eliminated from the disjunction and (iii) for [.] formulas, τp and ρ can similarly be constructed in a dual way.

Since there is no direct way to create the desired formula for $\alpha = \alpha_1^*$ while meeting the size constraints, we cannot do a direct inductive construction for εp. Instead we perform an inductive construction dp that is similar to εp but does not take *backwards edges* (q_f, q_0) originating from α^*-constructions into account. Then, εp can be constructed from dp by only considering a single backwards edge for each pair of states. The idea behind this is that each path p_* considering more than one backwards edge is *subsumed* by some path p_1 considering only one backwards edge in the sense that if p_* visits the set X_* of markings and p_1 visits the set X_1 of markings, then $X_1 \subseteq X_*$. Then, the conjunction over X_* is implied by the conjunction over X_1. Since in the construction of ρ, we perform a disjunction over all paths with a conjunction over all seen markings inside, we can then omit p_* from the disjunction.

Construction of dp and εp. First, we construct dp inductively.

$$\alpha = \tau \quad dp(q, q', \alpha) = \begin{cases} \mathit{false} & \text{if } q \neq q' \\ \mathit{true} & \text{else} \end{cases}$$

$$\alpha = \varepsilon \quad dp(q, q', \alpha) = \begin{cases} \mathit{false} & \text{if } q = q_1 \text{ and } q' = q_0 \\ \mathit{true} & \text{else} \end{cases}$$

$$\alpha = \alpha_1 + \alpha_2 \quad dp(q, q', \alpha) = \begin{cases} dp(q, q', \alpha_i) & \text{if } q, q' \in Q_i \\ dp(q_{0,i}, q', \alpha_i) & \text{if } q = q_0 \text{ and } q' \in Q_i \\ dp(q, q_{f,i}, \alpha_i) & \text{if } q \in Q_i \text{ and } q' = q_f \\ dp(q_{0,1}, q_{f,1}, \alpha_1) \vee dp(q_{0,2}, q_{f,2}, \alpha_2) & \text{if } q = q_0 \text{ and } q' = q_f \\ \mathit{false} & \text{else} \end{cases}$$

$$\alpha = \alpha_1 \cdot \alpha_2 \quad dp(q,q',\alpha) = \begin{cases} dp(q,q',\alpha_i) & \text{if } q,q' \in Q_i \\ dp(q,q_{f,1},\alpha_1) \wedge dp(q_{0,2},q',\alpha_2) & \text{if } q \in Q_1 \text{ and } q' \in Q_2 \\ \mathit{false} & \text{else} \end{cases}$$

$$\alpha = (\alpha_1)^* \quad dp(q,q',\alpha) = \begin{cases} dp(q,q',\alpha_1) & \text{if } q,q' \in Q_1 \\ \mathit{true} & \text{if } q = q_0 \text{ and } q' = q_f \\ dp(q_{0,1},q',\alpha_1) & \text{if } q = q_0 \text{ and } q' \in Q_1 \\ dp(q,q_{f,1},\alpha_1) & \text{if } q \in Q_1 \text{ and } q' = q_f \\ \mathit{false} & \text{else} \end{cases}$$

$$\alpha = \psi_k? \quad dp(q,q',\alpha) = \begin{cases} \mathit{true} & \text{if } q = q' \neq q_1 \\ \mathit{false} & \text{if } q = q_i, q' = q_j, i > j \\ v_k & \text{else} \end{cases}$$

Using dp, we are now able to construct εp directly for all q, q' and α.

$$\varepsilon p(q,q',\alpha) = dp(q,q',\alpha) \vee (dp(q,q_{f,\bar{\alpha}},\alpha) \wedge dp(q_{0,\bar{\alpha}},q',\alpha))$$

Here $\bar{\alpha}$ is the innermost $*$-construction that contains both q and q'. In case no such $\bar{\alpha}$ exists, both $dp(q,q_{f,\bar{\alpha}},\alpha)$ and $dp(q_{0,\bar{\alpha}},q',\alpha)$ are given by *false* instead.

Theoretical justification. In order to use this succinct alternative in our construction, we have to argue that it indeed has the desired properties. Therefore we establish a number of theorems:

▶ **Theorem 19.** *$|dp(q,q',\alpha)| \leq |\alpha|$ and $|\varepsilon p(q,q',\alpha)| \leq 3 \cdot |\alpha| + 2$ for all q, q' and α.*

Proof. The first claim can be established by a straightforward structural induction on α. It is easy to see that in each case of the construction, at most one operator is added to $dp(q,q',\alpha)$ and each partial term is used at most once.

The second claim follows directly from the first claim and the definition of εp. ◀

▶ **Lemma 20.** *We have $dp(q,q',\alpha) \equiv \bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\}$ for $\overset{\varepsilon}{\Rightarrow}_X$ constructed from M_α where all backwards edges from $*$-constructions are removed.*

Proof. We show this claim by a structural induction on α.

Case $\alpha = \tau$: There are two unmarked states q_0, q_1 in M_α with a τ-transition connecting them. There are no ε-transitions. Thus, we have $q \overset{\varepsilon}{\Rightarrow}_X q'$ iff $q = q'$ and $X = \emptyset$. Therefore we have $\bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\} \equiv \mathit{false}$ for $q \neq q'$ and $\bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\} \equiv \mathit{true}$ for $q = q'$, establishing the claim.

Case $\alpha = \varepsilon$: There are two unmarked states q_0, q_1 in M_α with an ε-transition connecting q_0 to q_1. Thus, we have $q \overset{\varepsilon}{\Rightarrow}_X q'$ iff $X = \emptyset$ and either $q \neq q_1$ or $q' \neq q_0$. Therefore we have $\bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\} \equiv \mathit{false}$ for $q = q_1$,$q' = q_0$ and $\bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\} \equiv \mathit{true}$ else, establishing the claim.

Case $\alpha = \alpha_1 + \alpha_2$: By induction hypothesis, the claim holds for α_1 and α_2. To obtain M_α from M_{α_1} and M_{α_2}, a new starting and final state are added with ε-transitions to the old starting states and from the old final states, respectively. We consider different cases how a path inducing $q \overset{\varepsilon}{\Rightarrow}_X q'$ could have been constructed. In the first case, where both q and q' are in the same automaton M_{α_i}, no additional paths could have been introduced by the new transitions. Thus, the claim follows immediately from the induction hypothesis. In the second case, where $q = q_0$ and q' is in M_{α_i} a path must take the ε-transition to $q_{0,i}$ and then take a path between $q_{0,i}$ and q'. Since q_0 is not

marked, we have $q_0 \overset{\varepsilon}{\Rightarrow}_X q'$ iff $q_{0,i} \overset{\varepsilon}{\Rightarrow}_X q'$, establishing the claim by induction hypothesis. The third case, where q is in M_{α_i} and $q' = q_f$ is analogous to the second one. Another case is $q = q_0$ and $q' = q_f$. Since M_{α_1} and M_{α_2} are not connected, we have $q \overset{\varepsilon}{\Rightarrow}_X q'$ iff $q_{0,1} \overset{\varepsilon}{\Rightarrow}_X q_{f,1}$ or $q_{0,2} \overset{\varepsilon}{\Rightarrow}_X q_{f,2}$ with the same argument as used in cases two and three. Since no paths are added from $q_{0,i}$ to $q_{f,i}$ when going over from M_{α_i} to M_α, $q_{0,i} \overset{\varepsilon}{\Rightarrow}_X q_{f,i}$ holds for $\overset{\varepsilon}{\Rightarrow}_X$ constructed from M_{α_i} iff it holds for $\overset{\varepsilon}{\Rightarrow}_X$ constructed from M_α. Therefore we have $\bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\} \equiv \bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q_{0,1} \overset{\varepsilon}{\Rightarrow}_X q_{f,1}\} \vee \bigvee\{\bigwedge_{\psi_i \in X} v_i | q_{0,2} \overset{\varepsilon}{\Rightarrow}_X q_{f,2}\} \equiv dp(q_{0,1}, q_{f,1}, \alpha_1) \vee dp(q_{0,2}, q_{f,2}, \alpha_2) = dp(q, q', \alpha)$ using the induction hypothesis. The remaining cases include $q = q_f$ with $q' = q_0$ and q being in M_{α_i} with q' being in $M_{\alpha_{1-i}}$. In both cases, q' is not reachable from q using only ε-transitions. Thus, the claim is established with similar arguments as in previous cases.

Case $\alpha = \alpha_1 \cdot \alpha_2$: We consider three cases. In the first one, q and q' are both in M_{α_i}. Since only transitions from M_{α_1} into M_{α_2} are possible but not backwards, a path inducing $q \overset{\varepsilon}{\Rightarrow}_X q'$ has to stay in M_{α_i} the whole time. The claim then follows from the induction hypothesis. In the second case, q is in M_{α_1} and q' is in M_{α_2}. A path from q to q' has to transition through $q_{f,1}$ and $q_{0,2}$ to be able to switch automata, thus we have $q \overset{\varepsilon}{\Rightarrow}_X q'$ iff $q \overset{\varepsilon}{\Rightarrow}_Y q_{f,1}$ and $q_{0,2} \overset{\varepsilon}{\Rightarrow}_Z q'$ for some Y, Z with $X = Y \cup Z$. Since for state pairs inside one of the subautomata it does not matter whether $\overset{\varepsilon}{\Rightarrow}_X$ was constructed from M_α or M_{α_i}, the claim follows from the induction hypothesis. In the last case, q is in M_{α_2} and q' is in M_{α_1}. Since q' is not reachable from q, $q \overset{\varepsilon}{\Rightarrow}_X q'$ can not hold for any X and the claim follows immediately.

Case $\alpha = \alpha_1^*$: We consider five cases. In the first case, we have $q, q' \in Q_1$. Since the backwards edge that was added during the construction is ignored for this lemma, no new paths from q to q' are added compared to M_{α_1}. Therefore the claim follows from the induction hypothesis. In the second case, we have $q = q_0$ and $q' = q_f$. The claim follows immediately from the fact that there is an ε-edge in between the two states. The third case, where $q = q_0$ and $q' \in Q_1$, and the fourth case, where $q \in Q_1$ and $q' = q_f$ work in a similar way by considering the added ε edges between old and new starting and final states and by using the induction hypothesis. In the last case, we have $q = q_f$ and $q' = q_0$. The claim holds since the backwards edge connecting the two states is ignored for this lemma.

Case $\alpha = \psi_k?$: We consider the different cases how q' can be reached by ε-transitions from q in M_α. In the first case, $q = q'$ with $q \neq q_1$, we have trivial reachability without encountering a state marking. Here, the claim is established as in previous cases. In the second case, where $q = q_i, q' = q_j$ with $i > j$, q' is not reachable from q since the ε-transitions only point in the other direction. The claim is again established as in previous cases. In all other cases, q' can be reached from q with ε-transitions, but only with encountering the state marking in q_1. Since this is the only state marking in M_α, we have $\bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\} \equiv v_i = dp(q, q', \alpha)$, establishing the claim. ◀

▶ **Theorem 21.** *We have $\varepsilon p(q, q', \alpha) \equiv \bigvee\{\bigwedge_{\psi_i \in X} v_i \mid q \overset{\varepsilon}{\Rightarrow}_X q'\}$ for $\overset{\varepsilon}{\Rightarrow}_X$ constructed from the automaton M_α.*

Proof. Compared to the claim made about dp in Lemma 20, backwards edges must now be considered in our claim about εp. The central observation is that the contribution of all ε-paths from q to q' is already captured by two particular types of ε-paths: either by going from q to q' directly without taking a backwards edge, or by taking only the backwards edge from the construction of $M_{\bar{\alpha}}$ exactly once (where $\bar{\alpha}$ is the innermost *-construction

that contains both q and q'). As was shown in Lemma 20, the first type is captured by $dp(q, q', \alpha)$. It is also straightforward to see from Lemma 20 that the second type is captured by $dp(q, q_{f,\bar{\alpha}}, \alpha) \wedge dp(q_{0,\bar{\alpha}}, q', \alpha)$.

We now argue that further backwards edges need not be considered and thus all paths are subsumed by these two cases. In order to use a backwards edge outside of $M_{\bar{\alpha}}$, a path has to leave $M_{\bar{\alpha}}$ via $q_{f,\bar{\alpha}}$ and finally reenter it via $q_{0,\bar{\alpha}}$. The contribution of such paths to the disjunction is subsumed by paths taking the backwards edge from $q_{f,\bar{\alpha}}$ to $q_{0,\bar{\alpha}}$ directly. Backwards edges on the paths from q to $q_{f,\bar{\alpha}}$ or on the paths from $q_{0,\bar{\alpha}}$ to q' on the other hand that originate in a final state q_f of some subautomaton only lead to paths that later visit q_f a second time. Thus their contribution is again subsumed by the contribution of the path where loops from q_f to itself are cut out. ◀