

33rd International Conference on Concurrency Theory

CONCUR 2022, September 12–16, 2022, Warsaw, Poland

Edited by

Bartek Klin

Sławomir Lasota

Anca Muscholl



LIPICS

Editors

Bartek Klin 

University of Oxford, UK
bartek.klin@cs.ox.ac.uk

Sławomir Lasota 

University of Warsaw, Poland
s.lasota@uw.edu.pl

Anca Muscholl

Bordeaux University, France
anca@labri.fr

ACM Classification 2012

Theory of computation → Concurrency

ISBN 978-3-95977-246-4

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/978-3-95977-246-4>.

Publication date

September, 2022

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>.

License

This work is licensed under a Creative Commons Attribution 4.0 International license (CC-BY 4.0):
<https://creativecommons.org/licenses/by/4.0/legalcode>.



In brief, this license authorizes each and everybody to share (to copy, distribute and transmit) the work under the following conditions, without impairing or restricting the authors' moral rights:

- Attribution: The work must be attributed to its authors.

The copyright is retained by the corresponding authors.

Digital Object Identifier: 10.4230/LIPIcs.CONCUR.2022.0

ISBN 978-3-95977-246-4

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>

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.

Editorial Board

- Luca Aceto (*Chair*, Reykjavik University, IS and Gran Sasso Science Institute, IT)
- Christel Baier (TU Dresden, DE)
- Mikolaj Bojanczyk (University of Warsaw, PL)
- Roberto Di Cosmo (Inria and Université de Paris, FR)
- Faith Ellen (University of Toronto, CA)
- Javier Esparza (TU München, DE)
- Daniel Král' (Masaryk University - Brno, CZ)
- Meena Mahajan (Institute of Mathematical Sciences, Chennai, IN)
- Anca Muscholl (University of Bordeaux, FR)
- Chih-Hao Luke Ong (University of Oxford, GB)
- Phillip Rogaway (University of California, Davis, US)
- Eva Rotenberg (Technical University of Denmark, Lyngby, DK)
- Raimund Seidel (Universität des Saarlandes, Saarbrücken, DE and Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Wadern, DE)

ISSN 1868-8969

<https://www.dagstuhl.de/lipics>

■ Contents

Preface	
<i>Bartek Klin, Sławomir Lasota, and Anca Muscholl</i>	0:ix

Invited Paper

CONCUR Test-Of-Time Award 2022	
<i>Ilaria Castellani, Paul Gastin, Orna Kupferman, Mickael Randour, and Davide Sangiorgi</i>	1:1–1:3

Invited Talks

Concurrent Separation Logics: Logical Abstraction, Logical Atomicity and Environment Liveness Conditions	
<i>Philippa Gardner</i>	2:1–2:1
Distributed Decision Problems: Concurrent Specifications Beyond Binary Relations	
<i>Sergio Rajsbaum</i>	3:1–3:13
Sequential Decision Making With Information Asymmetry	
<i>Jiarui Gan, Rupak Majumdar, Goran Radanovic, and Adish Singla</i>	4:1–4:18
Involved VASS Zoo	
<i>Wojciech Czerwiński</i>	5:1–5:13

Regular Papers

On the Axiomatisation of Branching Bisimulation Congruence over CCS	
<i>Luca Aceto, Valentina Castiglioni, Anna Ingólfssdóttir, and Bas Luttik</i>	6:1–6:18
Non-Deterministic Abstract Machines	
<i>Małgorzata Biernacka, Dariusz Biernacki, Sergueï Lenglet, and Alan Schmitt</i>	7:1–7:24
Slimming down Petri Boxes: Compact Petri Net Models of Control Flows	
<i>Victor Khomenko, Maciej Koutny, and Alex Yakovlev</i>	8:1–8:16
On the Sequential Probability Ratio Test in Hidden Markov Models	
<i>Oscar Darwin and Stefan Kiefer</i>	9:1–9:16
Parameter Synthesis for Parametric Probabilistic Dynamical Systems and Prefix-Independent Specifications	
<i>Christel Baier, Florian Funke, Simon Jantsch, Toghrul Karimov, Engel Lefaucheur, Joël Ouaknine, David Purser, Markus A. Whiteland, and James Worrell</i>	10:1–10:16
Anytime Guarantees for Reachability in Uncountable Markov Decision Processes	
<i>Kush Grover, Jan Křetínský, Tobias Meggendorfer, and Maximilian Weininger</i> ...	11:1–11:20
Checking Timed Büchi Automata Emptiness Using the Local-Time Semantics	
<i>Frédéric Herbretreau, B. Srivathsan, and Igor Walukiewicz</i>	12:1–12:24

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Sławomir Lasota, and Anca Muscholl



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

Simulations for Event-Clock Automata <i>S. Akshay, Paul Gastin, R. Govind, and B. Srivathsan</i>	13:1–13:18
History-Deterministic Timed Automata <i>Thomas A. Henzinger, Karoliina Lehtinen, and Patrick Totzke</i>	14:1–14:21
Decidability of One-Clock Weighted Timed Games with Arbitrary Weights <i>Benjamin Monmege, Julie Parreaux, and Pierre-Alain Reynier</i>	15:1–15:22
Language Inclusion for Boundedly-Ambiguous Vector Addition Systems Is Decidable <i>Wojciech Czerwiński and Piotr Hofman</i>	16:1–16:22
Complexity of Coverability in Depth-Bounded Processes <i>A. R. Balasubramanian</i>	17:1–17:19
Determinization of One-Counter Nets <i>Shaul Almagor and Asaf Yeshurun</i>	18:1–18:23
Energy Games with Resource-Bounded Environments <i>Orna Kupferman and Naama Shamash Halevy</i>	19:1–19:23
Half-Positional Objectives Recognized by Deterministic Büchi Automata <i>Patricia Bouyer, Antonio Casares, Mickael Randour, and Pierre Vandenhove</i>	20:1–20:18
Two-Player Boundedness Counter Games <i>Emmanuel Filiot and Edwin Hamel-de le Court</i>	21:1–21:23
Different Strokes in Randomised Strategies: Revisiting Kuhn’s Theorem Under Finite-Memory Assumptions <i>James C. A. Main and Mickael Randour</i>	22:1–22:18
Regular Model Checking Upside-Down: An Invariant-Based Approach <i>Javier Esparza, Mikhail Raskin, and Christoph Welzel</i>	23:1–23:19
On an Invariance Problem for Parameterized Concurrent Systems <i>Marius Bozga, Lucas Bueri, and Radu Iosif</i>	24:1–24:16
Towards Concurrent Quantitative Separation Logic <i>Ira Fesefeldt, Joost-Pieter Katoen, and Thomas Noll</i>	25:1–25:24
Completeness Theorems for Kleene Algebra with Top <i>Damien Pous and Jana Wagemaker</i>	26:1–26:18
Expressiveness and Decidability of Temporal Logics for Asynchronous Hyperproperties <i>Laura Bozzelli, Adriano Peron, and César Sánchez</i>	27:1–27:16
Propositional Dynamic Logic and Asynchronous Cascade Decompositions for Regular Trace Languages <i>Bharat Adsul, Paul Gastin, Saptarshi Sarkar, and Pascal Weil</i>	28:1–28:19
A Kleene Theorem for Higher-Dimensional Automata <i>Uli Fahrenberg, Christian Johansen, Georg Struth, and Krzysztof Ziemiański</i>	29:1–29:18

Diamonds for Security: A Non-Interleaving Operational Semantics for the Applied Pi-Calculus <i>Clément Aubert, Ross Horne, and Christian Johansen</i>	30:1–30:26
Weak Progressive Forward Simulation Is Necessary and Sufficient for Strong Observational Refinement <i>Brijesh Dongol, Gerhard Schellhorn, and Heike Wehrheim</i>	31:1–31:23
Strategies for MDP Bisimilarity Equivalence and Inequivalence <i>Stefan Kiefer and Qiyi Tang</i>	32:1–32:22
Pareto-Rational Verification <i>Véronique Bruyère, Jean-François Raskin, and Clément Tamines</i>	33:1–33:20
Concurrent Games with Multiple Topologies <i>Shaul Almagor and Shai Guendelman</i>	34:1–34:18
Generalised Multiparty Session Types with Crash-Stop Failures <i>Adam D. Barwell, Alceste Scalas, Nobuko Yoshida, and Fangyi Zhou</i>	35:1–35:25
An Infinitary Proof Theory of Linear Logic Ensuring Fair Termination in the Linear π -Calculus <i>Luca Ciccone and Luca Padovani</i>	36:1–36:18
On Session Typing, Probabilistic Polynomial Time, and Cryptographic Experiments <i>Ugo Dal Lago and Giulia Giusti</i>	37:1–37:18

■ Preface

This proceedings volume contains peer-reviewed contributions accepted at the 33rd International Conference on Concurrency Theory (CONCUR), 2022.

The CONCUR conference series brings together researchers, developers, and students in order to advance the theory of concurrency, and promote its applications. CONCUR 2022 was organised at Warsaw University (Poland), as part of the umbrella conference CONFEST 2022. In addition to CONCUR 2022, the CONFEST 2022 comprised also the 27th International Conference on Formal Methods for Industrial Critical Systems (FMICS), the 20th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS) and the 19th International Conference on Quantitative Evaluation of SysTems (QEST), alongside with several workshops.

Out of 90 submissions, the PC has accepted 32 papers for presentation at CONCUR 2022. Given the 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 2022 submissions. We are especially grateful to all our reviewers for their efforts in providing high-quality and timely reviews and conducting active discussions on each submission.

We are honored to have had Wojciech Czerwiński (Warsaw University, Poland), Philippa Gardner (Imperial College London, UK), Rupak Majumdar (Max Planck Institute for Software Systems, Germany) and Sergio Rajsbaum (Universidad Nacional Autónoma de México) as our invited speakers.

Starting in 2020, a CONCUR Test-of-Time(ToT) Award has been established by the CONCUR conference and the IFIP 1.8 Working Group on Concurrency Theory. The purpose of this award is to recognise important achievements in Concurrency Theory that were published at CONCUR conferences and have stood the test of time. For the 2022 edition, two periods are considered. Two awards were given to papers published in CONCUR between 1998 and 2001 and two more were given to papers published between 2000 and 2003. The award winners for the CONCUR ToT Awards 2022 have been selected by a jury composed of Ilaria Castellani (chair), Paul Gastin, Orna Kupferman, Mickael Randour, and Davide Sangiorgi. The results and winners of the CONCUR ToT Award 2022 selection process are described in the invited contribution by Ilaria Castellani in these proceedings.

We are very grateful to the University of Warsaw for hosting CONCUR. We thank the Ministry of Science and Higher Education of Poland for its generous financial support. As usual, the CONCUR 2022 proceedings are open access thanks to the LIPIcs series, and we are grateful to LIPIcs and Schloss Dagstuhl – Leibniz Center for Informatics for the invaluable service they provide.

Bartek Klin, Sławomir Lasota and Anca Muscholl
CONCUR 2022 PC Chairs



CONCUR Test-Of-Time Award 2022

Ilaria Castellani ✉ 

INRIA Sophia Antipolis Méditerranée, France

Paul Gastin ✉ 

Université Paris-Saclay, ENS Paris-Saclay, CNRS, France

Orna Kupferman ✉ 

School of Computer Science and Engineering, Hebrew University of Jerusalem, Israel

Mickael Randour ✉

F.R.S.-FNRS & UMONS - Université de Mons, Belgium

Davide Sangiorgi ✉

Department of Computer Science, University of Bologna, Italy

Abstract

This short article recaps the purpose of the CONCUR Test-of-Time Award and presents the four papers that received the Award in 2022.

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases CONCUR Test-of-Time Award

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.1

Category Invited Paper

Acknowledgements We thank Javier Esparza (chair of the CONCUR Steering Committee), Pedro d'Argenio and Ana Sokolova (chair and secretary of the IFIP Working Group 1.8 on Concurrency Theory), and Bartek Klin, Sławomir Lasota and Anca Muscholl (chairs of the CONCUR 2022 Programme Committee) for their assistance throughout our work as a jury for this year's award.

1 Introduction

The CONCUR Test-of-Time Award was established in 2020 by the Steering Committee of the CONCUR conference and by the IFIP Working Group 1.8 on Concurrency Theory. Its purpose is to recognise important achievements in Concurrency Theory that were published at CONCUR and have stood the test of time. At its normal pace, starting from 2024, the CONCUR Test-of-Time Award will be attributed every other year, during the CONCUR conference, to one or two papers published in the 4-year period from 20 to 17 years earlier. In the transient period from 2020 to 2023, on the other hand, two such awards are attributed every year, in order to catch up with papers published in the first fifteen years of the conference, namely between 1990 and 2004. At CONCUR 2020 two awards were given, each rewarding two papers published in the period 1990–1995. Similarly, at CONCUR 2021 two awards were given, each rewarding two papers published in the period 1994–1999.

We had the honour to serve as members of the third CONCUR Test-of-Time Award Jury. All papers published at CONCUR in the period 1998–2003 were eligible, and we were asked to select one or two papers for each of the two periods 1998–2001 and 2000–2003 (the overlap between the two periods allowing for some variability in the number of selected papers over the years). After setting up a shortlist of candidate papers and discussing their relative merits and influence on the CONCUR research community and beyond, we selected the four papers described below for the Award, out of a number of excellent candidates.



© Ilaria Castellani, Paul Gastin, Orna Kupferman, Mickael Randour, and Davide Sangiorgi; licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Sławomir Lasota, and Anca Muscholl; Article No. 1; pp. 1:1–1:3



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The presentation of the Award will take place during CONCUR 2022, the 33th edition of the CONCUR conference, which is co-chaired by Bartek Klin, Sławomir Lasota and Anca Muscholl, and will be held in Warsaw.

2 The Award Winning Contributions

2.1 Period 1998–2001

- Christel Baier, Joost-Pieter Katoen & Holger Hermanns. *Approximate symbolic model checking of continuous-time Markov chains*. CONCUR 1999.

https://doi.org/10.1007/3-540-48320-9_12

This paper presents the first symbolic model-checking algorithm for systems that combine probabilistic and real-time behaviours. Specifically, the model-checking algorithm handles real-time probabilistic systems, modelled by continuous-time Markov chains systems, and specifications in CSL – a branching and continuous-time stochastic logic. This setting significantly extends the scope of systems to which automatic model-checking can be applied. Beyond the new model-checking algorithm, the paper introduces several ideas that have been extensively used since their introduction in the paper. This includes a reduction from a quantitative model-checking problem to the problem of solving a system of equations, as well as a generalisation of BDDs to MTDDs (multi-terminal decision diagrams, which allow both Boolean and real-valued variables), which enables symbolic reasoning.

- Franck Cassez & Kim Larsen. *The Impressive Power of Stopwatches*. CONCUR 2000.

https://doi.org/10.1007/3-540-44618-4_12

This paper studies the expressive power of timed automata enriched with stopwatches and unobservable behaviours. Surprisingly, it is proved with smart constructions that this seemingly mild extension already reaches the full expressive power of linear hybrid automata, a very powerful model using a finite discrete control together with continuous variables, linear guards and linear updates. An important consequence is the reduction of the reachability analysis of linear hybrid automata to that of stopwatch automata. Even though both problems are undecidable, approximate reachability for stopwatch automata is easier to develop and implement. Stopwatch automata find another very important application in the field of scheduling problems for timed pre-emptive systems.

2.2 Period 2000–2003

- James J. Leifer & Robin Milner. *Deriving Bisimulation Congruences for Reactive Systems*. CONCUR 2000.

https://doi.org/10.1007/3-540-44618-4_19

This paper presents a uniform approach for deriving a Labelled Transition System (LTS) semantics from a reduction semantics, in such a way that the resulting bisimilarity is a congruence. LTS semantics, inspired by automata theory, specifies the interactive behaviour of systems, while reduction semantics specifies their internal evolution and is closer to the operational semantics of sequential programs. LTS semantics has been favoured in early work on process calculi, as it lends itself to the definition of a variety of behavioural equivalences that are easy to work with. Subsequently, a wealth of process calculi have been proposed, tailored to specific features (mobility, locations, security, sessions, etc). In these more complex calculi, it became more debatable what to adopt as labels or “observables” for the LTS semantics, and this motivated the shift towards a reduction semantics in conjunction with a structural congruence, allowing for a compact semantic description.

The thrust to retrieve an LTS semantics from a reduction semantics is an important one, and this paper is a milestone in this line of work. The solution proposed is robust, i.e., broadly applicable. It is also mathematically elegant, formulated using the categorical notion of relative pushout (RPO). The paper has spurred a whole trend of research on congruence properties for bisimilarity in which RPOs constitute the key notion. Good examples are applications to bigraphs, graph rewriting and name calculi.

- Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar & Mariëlle Stoelinga. *The Element of Surprise in Timed Games*. CONCUR 2003.
https://doi.org/10.1007/978-3-540-45187-7_9

This paper studies concurrent two-player games played on timed game structures, and in particular the ones arising from playing on timed automata. A key contribution of the paper is the definition of an elegant timed game model, allowing both the representation of moves that can take the opponent by surprise as they are played “faster”, and the definition of natural concepts of winning conditions for the two players – ensuring that players can win only by playing according to a physically meaningful strategy. This approach provides a clean answer to the problem of time convergence, and the responsibility of the players in it. For this reason, it has since been the basis of numerous works on timed games. The algorithm established in the paper to study omega-regular conditions in this neat model of timed games is also enticing, resorting to mu-calculus on a cleverly enriched structure.

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, have been conducted by Luca Aceto with the help of some jury members. The interviews are accessible as blog posts in the Process Algebra Diary maintained by Luca Aceto at <https://processalgebra.blogspot.com/>. Links to these interviews may also be found on the award’s webpage <https://concur2022.mimuw.edu.pl/tot-award/>.

Concurrent Separation Logics: Logical Abstraction, Logical Atomicity and Environment Liveness Conditions

Philippa Gardner ✉
Imperial College London, UK

Abstract

Scalable verification for concurrent programs with shared memory is a long-standing, difficult problem. In 2004, O’Hearn and Brookes introduced concurrent separation logic to provide compositional reasoning about coarse-grained concurrent programs with synchronisation primitives (Gödel prize, 2016).

In 2010, I introduced logical abstraction (the fiction of separation) to CSL, developing the CAP logic for reasoning about fine-grained concurrent programs in general and fine-grained lock algorithms in particular. In one logic, it was possible to provide two-sided specifications of concurrent operations, with formally verified implementations and clients.

In 2014, I introduced logical atomicity (the fiction of atomicity) to concurrent separation logics, developing the TaDA logic to capture when individual operations behave atomically. Unlike CAP, where synchronisation primitives leak into the specifications, with TaDA the specifications are “just right” in that they provide more general atomic functions specifications to capture, for example, the full behaviour of lock operations.

In 2021, I introduced environment liveness conditions to concurrent separation logics, developing the TaDA Live logic for reasoning compositionally about the termination of blocking fine-grained concurrent programs. The crucial challenge is how to deal with abstract atomic blocking: that is, abstract atomic operations that have blocking behaviour arising from busy-waiting patterns as found in, for example, fine-grained spin locks. The fundamental innovation is with the design of abstract specifications that capture this blocking behaviour as liveness assumptions on the environment.

In this talk, I will explain this on-going journey in the wonderful world of concurrent separation logics. I will also explain why I have a bright green office chair in the corner of my office, patterned in gold lamé.

Many thanks to my fabulous coauthors on concurrent separation logics: Thomas Dinsdale-Young, Emanuele D’Osualdo, Mike Dodds, Azadeh Farzan, Matthew Parkinson, Pedro da Rocha Pinto, Julian Sutherland, Viktor Vafeiadis and more.

Suggested Reading:

- Peter O’Hearn: Resources, Concurrency and Local Reasoning, *Journal of Theoretical Computer Science*, Festschrift for John C Reynolds 70th birthday, 2007.
- Thomas Dinsdale-Young, Pedro da Rocha Pinto and Philippa Gardner: A Perspective on Specifying and Verifying Concurrent Modules, *Journal of Logical and Algebraic Methods in Programming*, 2018.
- Emanuele D’Osualdo, Azadeh Farzan, Philippa Gardner and Julian Sutherland: TaDA Live: Compositional Reasoning for Termination of Fine-grained Concurrent Programs, *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 2021.

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases Concurrent separation logic

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.2

Category Invited Talk



© Philippa Gardner;

licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 2; pp. 2:1–2:1

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Distributed Decision Problems: Concurrent Specifications Beyond Binary Relations

Sergio Rajsbaum   

Institute of Mathematics, National Autonomous University of Mexico, Mexico City, Mexico

Abstract

Much discussion exists about what is computation, but less about is a computational problem. Turing's definition of computation was based on computing functions. When we move from sequential computing to interactive computing, discussions concentrate on computations that do not terminate, overlooking notions of distributed problems. Many models where concurrency happens have been proposed, ranging from those equivalent to a Turing machine, to those where much heated discussion has taken place, claiming that interactive models are fundamentally different from Turing machines.

It is argued here that there is no need to go all the way to non-terminating interaction, to appreciate how different distributed computation is from sequential computation. The discussion concentrates on the various ways that exist of representing a *distributed decision problem*. Each process of a distributed system starts with an initial private input value, and after communicating with other processes in the system, produces a local output value. An input/output relation is needed, to specify which output values are legal for a particular assignment of input values to the processes.

An overview is provided of some results that show how rich the topic of distributed decision problems can be, when asynchronous processes can fail, but mostly independent of particular models of distributed computing and their many intricate details (types of failures and of communication). We are in a world very different from that of the functions of sequential computation; moving away from the world of graphs beyond binary relations, to the world of simplicial complexes.

2012 ACM Subject Classification Theory of computation → Distributed computing models

Keywords and phrases Distributed decision tasks, simplicial complex, linearizability, interval-linearizability, Arrow's impossibility, Speedup theorems

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.3

Category Invited Talk

Funding Supported by UNAM-PAPIIT IN106520.

1 What is Computation and what is a Computational Problem?

The tools of a barber include scissors, razor, shave brush, comb, clipper, neck duster; the process that repeatedly uses these tools is barbering. It is awkward to talk about barbering before saying what the problem being solved is: shaving, hair-cutting, and hair-dressing. Yet, it seems we are sometimes more obsessed with understanding what is computation, than with understanding what is a computational problem.

The first ACM Ubiquity symposium (2011) thoroughly discussed the question: What is computation? The most fundamental question of our field, says Peter Denning in the Editor's Introduction [18]. But except for mentioning Turing and how he invented his machine to classify *functions* according to computability, not much is said about computational problems.

For sequential computing not much is discussed about computational problems, beyond functions, and for distributed computing even less. The participants of the workshop were asked to consider how three new developments might have affected the traditional answers to the question. One of the three developments is interactive computation, motivated by situations such as operating systems and networks that are based on computations that do



© Sergio Rajsbaum;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 3; pp. 3:1–3:13

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

3:2 Distributed Decision Problems

not terminate and regularly interact with their environments. All through the discussions in the workshop, it seems that the interest on interactive computation comes from their non-terminating nature, e.g. [24].

Some argue about the enduring legacy of the Turing Machine like Lance Fortnow [21], while others strongly against it, like Peter Wegner [54]. But in the conclusions of the workshop, Denning [17] mentions that there is an emerging consensus that interactive models are fundamentally different from Turing machines.

Aho [1] easily describes the computational problem: A function f from strings to strings is computable if there is some Turing machine M that given any input string w always halts in the accepting state with just $f(w)$ on its tape. But describes in detail what a Turing machine is:

The reason we went through this explanation is to point out how much detail is involved in precisely defining the term computation for the Turing machine, one of the simplest models of computation. It is not surprising, then, as we move to more complex models, the amount of effort needed to precisely formulate computation in terms of those models grows substantially.

Aho [1] continues: Many real-world computational systems compute more than just a single function – the world has moved to interactive computing. But there is no discussion of what is it that they compute.

Indeed, as the authors of the workshop discuss, there are many models of distributed computing, consisting of autonomous computing processes that communicate with one another. To model multicore shared memory systems, wide area message passing networks, biological systems such as cells and organisms, even the human brain. There are theoretical models such as message-passing Actor model, Petri nets, process calculi, I/O automata, etc. Many shared-memory and message passing models are discussed in the distributed computing literature, e.g. [5, 33, 48, 49].

2 Distributed Decision Problems

To discuss distributed computing problems, very few details about the computational model need to be considered; the same notions of distributed computing problem are relevant to many of the models mentioned above.

2.1 Distributed computing problems

There are many problems to discuss about distributed computing. Distributed systems can exhibit behaviors such as deadlock, livelock, race conditions. And there are many aspects to study about routing, robot coordination, agents moving along a network, distributed graph algorithms, and the like that cannot be studied using Turing machines. Concerns such as reliability, performance, scalability and adaptivity, mobility, physical locality, are inherently different from sequential computing.

All through the symposium, it is emphasized the importance of models where interaction takes place, assuming as evident that the interest is in non-terminating computations. I would like to slow down here, to show the richness exhibited already in *terminating* distributed computation. Furthermore, that there is no need to get into the intricacies of a distributed computing model, to discuss distributed problems. The goal is to show that indeed very novel issues arise that do not exist in Turing machines, already when we consider input/output problems.

In this paper the goal is to focus on possibly the purest form of distributed computing problem, a direct analogue of the notion of a function for a Turing machine. The input x to the function is now distributed, each process knows only part of x . Also the output $f(x)$ is distributed: after communicating with each other, the processes collectively compute $f(x)$, each one computes one part of it. As we shall see, instead of functions it is of interest to consider relations $T(x)$, called *tasks*, possibly allowing for more than one output for each input x .

Assume the simplest case of a fixed, finite set of n individual processes composing the distributed system. To focus only on the problem of computing a task in a distributed way, disregard any routing and network communication problems, and assume that the processes can directly communicate with each other. Similarly, to focus only on the distributed aspects of the problem, disregard any individual sequential computing limitation. It turns out that some tasks have no solution, even if each process is an infinite state automata, while when there is a solution, each process is a (usually) simple Turing machine.

For the purposes of discussing distributed problems, there is no need to discuss many of the specifics about the computational model – ways in which processes communicate with each other, their relative speeds and failures. Roughly, the only thing needed, is that a process may have to produce an output value without knowing the input values of some of the other processes.

2.2 Distributed decision tasks

Early on in the development of distributed computing theory, Moran and Wolfstahl [43] defined the notion of *distributed decision task*, to encompass the various problems that were being studied at that time, such as consensus, approximate agreement and renaming. It was already known that consensus is impossible to solve in a message passing system even if only one process can fail by crashing [20] (even if each process is an infinite state machine). Moran and Wolfstahl extended the impossibility to general decision tasks, and then Biran, Moran and Zaks [6] extended it to a full characterization.

Consider n -dimensional vectors with entries over some set of possible values V : the i -th entry of a vector is associated to the i -th process. A distributed decision task $\mathcal{T} = \langle \mathcal{I}, \mathcal{O}, \Delta \rangle$ consists of a set of *input vectors*, \mathcal{I} a set of *output vectors*, \mathcal{O} and a relation Δ , specifying, for each input vector $I \in \mathcal{I}$, a set of legal output vectors $\Delta(I) \subseteq \mathcal{O}$. The i -th entry of an input vector is the input value of the i -th process. The i -th entry of an output vector is the output value of the i -th process. It is assumed that each process, has two special variables, a read-only one for the input value and a write-once variable for the output value.

A decision task is *solvable* by a distributed algorithm in some model of computation, if the following holds. The system can start in any of the input vectors $I \in \mathcal{I}$ allowed by the task. Now, consider any execution starting with input vector I , where all processes produce an output value, defining a vector O consisting of all the n output values. Then, it must be the case that $O \in \Delta(I)$.

Notice that task solvability is defined only by a safety requirement. There is also a liveness requirement defined by the specifics of the model of computation. In the sequel of papers by Biran, Moran, Zaks and Wolfstahl [6, 7, 8, 43], the focus was on 1-resilient asynchronous processes (running at arbitrary speeds, independent from each other) communicating by message passing. In this case, the liveness requirement is that, in an execution where at most one process crashes, all processes that do not crash have to produce a decision value. A similar situation but in shared memory was considered by Moran and Taubenfeld [53], including the case where $t < n$ processes may crash, where the liveness is adjusted accordingly.

The following examples are well-known by now.

1. *Consensus*. For a set of values V , the inputs are all n -vectors over V . There is one output vector for each $v \in V$, consisting of all output values equal to v , denoted O_v , and $\mathcal{O} = \cup_{v \in V} \{O_v\}$. For any input vector with at least two different input values, $\Delta(I) = \mathcal{O}$, for an input vector I_v with a single input value v , $\Delta(I_v) = \{O_v\}$.
2. *Approximate agreement*. It is defined in [6] for any given $\epsilon > 0$, and V the set of rational numbers. Any n -vector over V is a possible input vector, and the output vectors contain rational numbers so that for any two entries d_i, d_j , $|d_i - d_j| \leq \epsilon$. Then, $\Delta(I)$ contains all output vectors with entries d_i such that $m \leq d_i \leq M$, where m is the smallest value of I and M is the largest.

There are many variants of consensus and approximate agreement, including multidimensional ones e.g. [41].

2.3 Participating processes

The discovery of the intimate connection between distributed computing and topology, overviewed in [30], was facilitated by the realization that the 1-resilient case is not the most fundamental situation, and surprisingly not the easiest to analyze – it is the *wait-free* case. Wait-freedom is a progress condition which guarantees that each process can make progress in a finite number of steps regardless of the behavior of other processes. So long as processes are scheduled, wait-freedom guarantees progress for all processes. Thus, a distributed algorithm that is wait-free never includes instructions by which a process waits for an event of another process (if that process crashes, the event might never happen).

For this paper, the important property is that any set of processes may have to produce output values, without knowing the input values of the remaining processes. Therefore, the vectors of a decision task need to incorporate a notion of *participating* processes. Not all entries in a given input (output) vector need contain an input (output) value; some may contain the special value \perp , indicating that some processes do not participate in the execution (crashes before taking any steps). Thus, the set of input vectors is required to be prefix closed. Meaning that if I is an input vector, then the task has to consider also any input vector I' contained in I , in the sense that any subset of the entries of I is replaced by \perp . Furthermore, for each such input vector I' , where the input values of some subset of the processes P' is defined as \perp , the input/output relation Δ has to specify what are the legal output vectors. Namely, $\Delta(I')$ is a set of vectors, all with \perp in the entries for processes P' .

As already discussed by Herlihy and Shavit [32] and Hoest and Shavit [35], the intuitive notion of “order of actions in time” is captured through the use of participating processes. The example given is how it can be used to distinguish between tasks such as Unique-Id and Fetch-And-Increment, which have the same sets of input and output vectors, have the same Δ when all processes participate, but have quite different task specification maps when subsets of participating processes are taken into account. The Figure 1 is from [35], for a set of $n + 1$ processes.

The Unique-Id task is defined as follows: each participating process $i \in \{0, \dots, n\}$ has an input $x_i = 0$ and chooses an output $y_i \in \{0, \dots, n\}$ such that for any pair of processes $i \neq j$, $y_i \neq y_j$.

In the Fetch-And-Increment task, each participating process $i \in \{0, \dots, n\}$ has an input $x_i = 0$ and chooses a unique output $y_i \in \{0, \dots, n\}$ such that (1) for some participating process i , $y_i = 0$, and (2) for $1 \leq k \leq n$, if $y_i = k$, then for some $j \neq i$, $y_j = k - 1$.

(0, ⊥, ⊥)	(0, ⊥, ⊥), (1, ⊥, ⊥), (2, ⊥, ⊥)	(0, ⊥, ⊥)	(0, ⊥, ⊥)
(⊥, 0, ⊥)	(⊥, 0, ⊥), (⊥, 1, ⊥), (⊥, 2, ⊥)	(⊥, 0, ⊥)	(⊥, 0, ⊥)
(⊥, ⊥, 0)	(⊥, ⊥, 0), (⊥, ⊥, 1), (⊥, ⊥, 2)	(⊥, ⊥, 0)	(⊥, ⊥, 0)
(0, 0, ⊥)	(0, 1, ⊥), (1, 0, ⊥), (0, 2, ⊥), (0, 2, ⊥), (2, 1, ⊥), (1, 2, ⊥)	(0, 0, ⊥)	(0, 1, ⊥), (1, 0, ⊥)
(0, ⊥, 0)	(0, ⊥, 1), (1, ⊥, 0), (0, ⊥, 2), (0, ⊥, 2), (2, ⊥, 1), (1, ⊥, 2)	(0, ⊥, 0)	(0, ⊥, 1), (1, ⊥, 0)
(⊥, 0, 0)	(⊥, 0, 1), (⊥, 1, 0), (⊥, 0, 2), (⊥, 0, 2), (⊥, 2, 1), (⊥, 1, 2)	(⊥, 0, 0)	(⊥, 0, 1), (⊥, 1, 0)
(0, 0, 0)	(0, 1, 2), (0, 2, 1), (1, 0, 2), (1, 2, 0), (2, 0, 1), (2, 1, 0)	(0, 0, 0)	(0, 1, 2), (0, 2, 1), (1, 0, 2), (1, 2, 0), (2, 0, 1), (2, 1, 0)

(a) Unique-Id task.

(b) Fetch-And-Increment task.

■ **Figure 1** Two tasks with the same set of vectors when all participate. First column is the input vector, and for each row I , in the second column $\Delta(I)$.

2.4 Beyond binary relations

These two examples already hint at why by moving from vectors to partial vectors, the notion of decision task is interestingly enriched. This is clearly exposed using the appropriate mathematical structure for partial vectors closed under containment: simplicial complexes. Here follows an overview of how to use them to represent tasks, additional details are in e.g. [30].

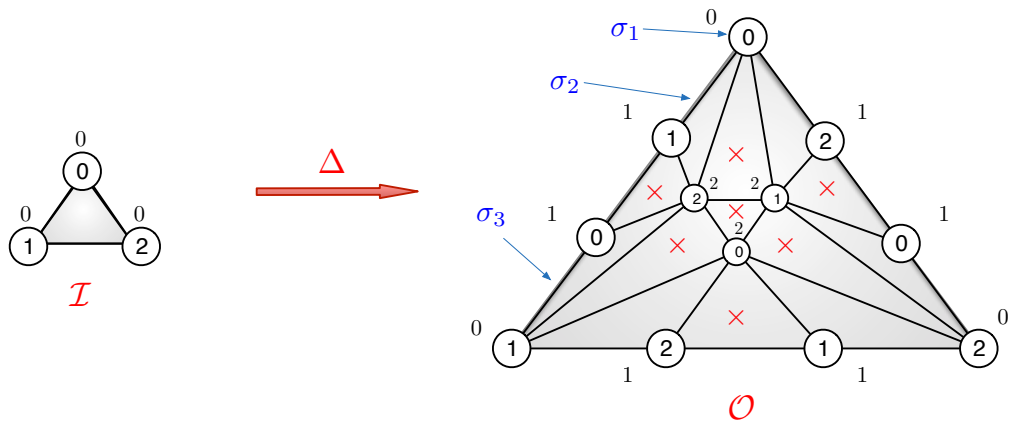
The following notions are illustrated in Figure 2, where the Fetch-And-Increment task is represented using simplicial complexes, for three processes denoted 0, 1, 2. Intuitively, triangles represent vectors with 0 entries equal to \perp , edges represent vectors with 1 entry equal to \perp , and vertices correspond to vectors with 0 entries equal to \perp . An input vertex (i, x) means that process i has input value x , and for the vertex $(0, 0)$, $\Delta(i, x) = \sigma_1$. For the input edge $\{(0, 0), (1, 0)\}$, $\Delta(\{(0, 0), (1, 0)\}) = \{\sigma_2, \sigma_3\}$, while for the input triangle $\{(0, 0), (1, 0), (2, 0)\}$, $\Delta(\{(0, 0), (1, 0), (2, 0)\})$ consists of all 6 triangles of \mathcal{O} .

A *simplicial complex* is a generalization of a graph, where sets of vertices of cardinality more than two can also be grouped into a *simplex* (the generalization of an edge). Formally, it is a collection \mathcal{K} of non-empty sets, closed under containment, i.e., if $\sigma \in \mathcal{K}$ then, for every non-empty set $\sigma' \subseteq \sigma$, $\sigma' \in \mathcal{K}$. Every set in \mathcal{K} is called a *simplex*. A subset of a simplex is called a *face*, and a *facet* of \mathcal{K} is a face that is maximal for inclusion in \mathcal{K} . The *dimension* of a simplex σ is $|\sigma| - 1$, where $|\sigma|$ denotes the cardinality of σ . The dimension of a complex is the maximal dimension of its facets. A complex in which all facets are of the same dimension is called *pure*. The *vertices* of \mathcal{K} are all simplices with a single element (i.e., of dimension 0). The set of vertices of a complex \mathcal{K} are denoted by $V(\mathcal{K})$.

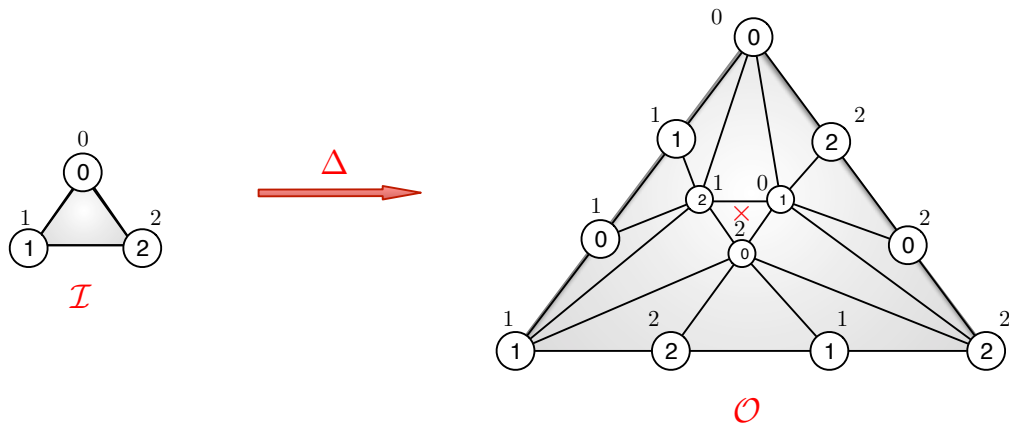
All complexes in this paper are *chromatic*, i.e., every vertex is a pair $v = (i, x)$ where $i \in [n] = \{1, \dots, n\}$ for some $n \geq 1$ is the color of v denoting a process, and x is some value (an input value or an output value). Moreover, in a chromatic complex, a color i must appear at most once in every simplex. Let $\sigma = \{(i, x_i) : i \in I\}$ be a simplex. We denote by $\text{ID}(\sigma)$ the set of colors in σ , i.e., $\text{ID}(\sigma) = I$. Indeed, in the following, the color of a vertex is actually the *identity* of a process.

A task can be defined using vectors as above, or using simplicial complexes as follows, exposing the role of combinatorial topology notation, and why going beyond binary relations is intrinsic to distributed computing problems.

A *task* for n processes is a triple $\Pi = (\mathcal{I}, \mathcal{O}, \Delta)$ where \mathcal{I} and \mathcal{O} are $(n - 1)$ -dimensional complexes, respectively called *input* and *output* complexes, and $\Delta : \mathcal{I} \rightarrow 2^{\mathcal{O}}$ is an input-output specification. Every simplex $\sigma = \{(i, x_i) : i \in I\}$ of \mathcal{I} , where $I = \text{ID}(\sigma)$ is a non-empty subset of $[n]$, defines a legal input state corresponding to the scenario in which, for every $i \in I$, process i starts with input value x_i . Similarly, every simplex $\tau = \{(i, y_i) : i \in I\}$ of \mathcal{O} defines a legal output state corresponding to the scenario in which, for every $i \in I$, process i outputs the value y_i . The map Δ is an input-output relation specifying, for every input state



■ **Figure 2** The Fetch-And-Increment task. Inside a vertex is its id, outside is its input or output value. The input complex \mathcal{I} consists of a single triangle, and its faces. The input complex \mathcal{O} consists of 6 triangles, and its faces. The triangles marked with an \times are deleted.



■ **Figure 3** The input complex of the set agreement task for 3 processes, and part of the output complex. The triangle marked with \times is deleted. The corners of the input triangle are mapped by Δ to the corners of \mathcal{O} . The boundary of the input triangle is mapped to the boundary of \mathcal{O} . The input triangle is mapped to all the depicted 12 triangles of \mathcal{O} .

$\sigma \in \mathcal{I}$, the set of output states $\tau \in \mathcal{O}$ with $\text{ID}(\tau) = \text{ID}(\sigma)$ that are legal with respect to σ . That is, assuming that only the processes in $\text{ID}(\sigma)$ participate to the computation (the set of participating processes is not known a priori to the processes in σ), these processes are allowed to output any simplex $\tau \in \Delta(\sigma)$. It is often assumed that Δ is a carrier map (that is, for every $\sigma, \sigma' \in \mathcal{I}$, if $\sigma' \subseteq \sigma$ then $\Delta(\sigma') \subseteq \Delta(\sigma)$ as subcomplexes).

An important example is the *set agreement* task, with a single input facet (and all its faces), where process i starts with input value i . The n processes need to agree on at most $n - 1$ input values of participating processes. For three processes, at most two different values can be decided, as illustrated in Figure 3, where part of the output complex is depicted. This task is important, because it is unsolvable wait-free [9, 32, 52], and the reason for the impossibility is a topological one: intuitively, the task has a hole while no wait-free distributed algorithm has one.

3 Selected Topics

Here is a selection of the various aspects about tasks that have been studied. Consensus is the most fundamental task, in a sense the most difficult one together with variants such as *interactive consistency* where all of the processes have to agree on the same vector such that the i th entry of the vector contains the value proposed by the i -process; any task is solvable, if processes can agree on their inputs. Much can be said about consensus in long-lived situations, and consensus is known to be enormously important in real systems since early on [37], as well as in theory e.g. [29], for reasons including the consensus hierarchy [39] and as a universal object [50], but here the focus is on decision problems.

3.1 Colorless tasks, local tasks, continuous tasks: decidability and reductions

The class of *colorless tasks* was identified in [10]. Such a task can be defined in terms of sets of input and output values, without referring to which process is assigned which input value or produces which output value, and without referring to the number of processes in the system. Many widely-studied tasks are colorless, including consensus, set-agreement, and approximate agreement. Some important tasks like renaming [13] and others [12] are not colorless, and are more difficult to study, but easier than set agreement [11]. A notion of *continuous task* has been proposed aiming at obtaining wait-free solvability characterization [25] in a more intuitive way than the original one [32].

The *rendezvous task* [38] is a colorless task that models scenarios where autonomous agents move around in a specific space to meet one another. A chromatic version where a process must end in a vertex of its own color in a chromatic subdivision of an input simplex, is the *chromatic simplex agreement task*, important for the wait-free task solvability theorem [32], and the *affine tasks*, on subcomplexes of the chromatic subdivision by Kuznetsov and Rieutord [36]. The *loop agreement task* is an example of rendezvous task, which is defined in terms of an edge loop in a 2-complex. Herlihy and Rajsbaum [31] showed that a loop agreement task is wait-free solvable if and only if the loop is contractible in the 2-complex, as a result, the wait-free solvability of loop agreement tasks is undecidable. Rendezvous on the vertices of a graph was introduced in [15], and variants were studied in [3] including applications to robot coordination problems [2].

A task G *implements* task F if one can construct a protocol for F by calling any protocol for G , possibly followed by access to a shared read/write memory. This notion of implementation induces a partial order on tasks and hence it induces a classification of a set of tasks, into disjoint classes such that tasks in the same class implement each other. In this sense, all tasks in a class are computationally equivalent. A classification of loop agreement tasks was presented in [31], and extended in [55] to rendezvous tasks.

A task T is wait-free checkable if and only if it satisfies a certain *locality* condition. Notions of locality considered by Fraigniaud, Travers and Rajsbaum [23] are mostly independent of the computing model. Wait-free solvability of local tasks remains undecidable. A strong notion of locality is defined by *covering tasks* whose output complex is a covering of the input complex. This topological property yields obstacles for wait-free solvability different in nature from the classical agreement impossibility results, and, apart from the identity task, locality-preserving tasks are not wait-free solvable. A classification of locality-preserving tasks in term of their computational power is presented. Also closely related to covering tasks and with a similar impossibility argument [26], is the *equality negation task*. For two processes, each of which has an input from a set of three distinct values, each process must

decide a binary output value so that the decisions of the processes are the same if and only if the initial values of the processes are different. This task was defined by Lo and Hadzilacos [39], as the central idea to prove that the consensus hierarchy is not robust.

Fraigniaud, Paz and Rajsbaum [22] study consensus and approximate agreement, through an approach for proving lower bounds and impossibility results, called the *asynchronous speedup theorem*. For a given task T and a given computational model M , the *closure* of T with respect to M is a task that is supposed to be a slightly easier version of T . The asynchronous speedup theorem states that if a task T is solvable in $t \geq 1$ rounds in M , then its closure w.r.t. M is solvable in $t - 1$ rounds in M . As an application they study the power of test&set and binary consensus, for wait-free solving approximate agreement faster.

3.2 Domain restrictions and social choice

A research line started by Mostefaoui, Rajsbaum and Raynal [44] considers restricting the input domain of a task, to obtain an easier task. A restriction of the input complex is called a *condition*. For example, although consensus is unsolvable even if only one process can crash, if we assume that more than a majority of processes propose the same value then consensus becomes solvable ($n \geq 4$). The paper identified the conditions for which consensus is solvable in an asynchronous distributed system with t crash failures. In a sequel paper [45] they study conditions for consensus in a synchronous system where processes can fail by crashing. A hierarchy of conditions parametrized by d is presented, that allows solving synchronous consensus with less and less rounds, as we go from $d = t$ to $d = 0$.

There are remarkable analogies between social choice theory and distributed computing, despite the fact that social choice theory is typically not concerned with concurrency (for decentralised studies see [16, 40]). The modern field of social choice theory took off with Kenneth Arrow's remarkable 1950 result [4] for the basic problem of democracy: it is impossible to aggregate individual preferences into a single social preference, under some reasonable-looking axioms. In Arrow's setting, each process proposes a total order on the possible candidates, and the outcome of the election, computed by a centralized aggregation function f , is also a total order, that should reflect the social preference. One requirement is *unanimity*, if everyone prefers candidate x over y , so should the social preference. With only this requirement, the aggregation function can simply decide on the preferences of one individual, say the 1st one, which would become a dictator. Arrow's impossibility says that f must be dictatorial, if one requires, additionally to the unanimity requirement, an *independence of irrelevant alternatives* (IIS) requirement, stating that f depends only on pairwise preferences.

Much research has been devoted to identify domain restriction to circumvent Arrow's impossibility theorem. Rajsbaum and Raventós [47] identify the exact domain restrictions for the case of two voters and three alternatives, and present a new proof of Arrow's impossibility based on a task formalization using simplicial complexes, showing that any unanimous IIS aggregation function must be dictatorial, on any of the corresponding restricted domains. The proof uses techniques analogous to those used in distributed computing [13, 26].

3.3 Tasks and objects

Tasks are not the only possible input/output distributed specifications. Objects are defined in terms of sequential specifications, and can specify ongoing, never-ending behavior, such as for concurrent data structures [42]. For this paper consider their one-shot version, and one method that can be invoked only once by each process, with an input parameter. The object

returns an output value to the invoking process. Thus, one-shot objects are similar to tasks, in that they specify input/output problems. Objects come with an accompanying notion of when a concurrent execution satisfies the object's sequential specification, linearizability.

In fact we encounter in the literature three ways of talking about distributed decision problems. As a set of informal requirements, as a sequential object plus a consistency condition (linearizability), and as a task. For example, we have seen that consensus can be defined as a task. But often it is defined by two safety requirements. Validity: a decided value is the input of some participating process; Agreement: any two decided values are equal. The third way is to think of consensus as an object, defined by a sequential automata, whose states represent which values have been proposed to the object, and which values can be returned to a process.

The relation between tasks and objects has been studied by Castañeda, Rajsbaum and Raynal [14], motivated by Neiger [46], who proposed a generalization of linearizability to be able to specify tasks, such as set agreement, which have no natural specification as sequential objects. Set-sequential objects can define executions in which a set of processes access an object concurrently. The notion of an *interval-sequential* object [14], together with a corresponding consistency condition, is able to express any concurrency pattern of overlapping invocations of operations, that might occur in an execution [27]. While some important tasks have no specification either as a sequential object nor as a set-sequential object, all tasks can be naturally expressed as interval-sequential objects. Remarkably, there are objects that cannot be expressed as tasks. An extension of the task framework is described, called *refined tasks*, that has more expressive power, and is able to specify any one-shot interval-sequential object.

An interesting notion appears with objects, *composability*, which has not been studied as much for tasks. Linearizability is very popular to design components of large systems because one can consider linearizable object implementations in isolation and compose them for free, without sacrificing linearizability of the whole system [34]. It was shown that by going from linearizability to interval-linearizability, one does not sacrifice the benefits of composability [14].

3.4 Tasks and knowledge

Rosenbloom [51] argues that computation is information transformation. In this sense, one may view a distributed problem as setting the goals, from an initial state of information, to a final one. More precisely, a task is reformulated by Goubault, Rajsbaum and Ledent [28] as a knowledge transformation goal. The input complex defines what processes know about each other inputs, formalized as a *simplicial model*, the dual of the classic one-dimensional Kripke model, that exposes relations beyond binary. A task can be re-interpreted as a goal in terms of knowledge gain, using an output simplicial model, which is the product update of the initial simplicial model and an action model. This formally specifies the knowledge gain required by the task.

The importance of common knowledge for reaching agreement is well understood [19]. Consensus and common knowledge is discussed in [28], as well as approximate agreement, in terms of knowledge gain. After all, the difficulty of distributed decisions comes from the absence of common knowledge about the inputs and consensus gives us just that. Indeed, in the epistemic setting, consensus is the requirement of achieving common knowledge on an input value. This is impossible in asynchronous systems. In contrast, approximate agreement is solvable, because it is a finite version of common knowledge, requiring only that everybody knows that everybody knows, and so on, a certain number of times.

References

- 1 Alfred V. Aho. Computation and Computational Thinking. *The Computer Journal*, 55(7):832–835, July 2012. doi:10.1093/comjnl/bxs074.
- 2 Manuel Alcantara, Armando Castañeda, David Flores-Peñaloza, and Sergio Rajsbaum. The topology of look-compute-move robot wait-free algorithms with hard termination. *Distributed Comput.*, 32(3):235–255, 2019. doi:10.1007/s00446-018-0345-3.
- 3 Dan Alistarh, Faith Ellen, and Joel Rybicki. Wait-free approximate agreement on graphs. In Tomasz Jurdzinski and Stefan Schmid, editors, *Structural Information and Communication Complexity - 28th International Colloquium, SIROCCO 2021, Wrocław, Poland, June 28 - July 1, 2021, Proceedings*, volume 12810 of *Lecture Notes in Computer Science*, pages 87–105. Springer, 2021. doi:10.1007/978-3-030-79527-6_6.
- 4 Kenneth J. Arrow. A difficulty in the concept of social welfare. *Journal of Political Economy*, 58(4):328–346, 1950. doi:10.1086/256963.
- 5 Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons, Hoboken, NJ, USA, 2004.
- 6 Ofer Biran, Shlomo Moran, and Shmuel Zaks. A combinatorial characterization of the distributed 1-solvable tasks. *J. Algorithms*, 11(3):420–440, 1990. Preliminary version in PODC 1988. doi:10.1016/0196-6774(90)90020-F.
- 7 Ofer Biran, Shlomo Moran, and Shmuel Zaks. Deciding 1-solvability of distributed task is np-hard. In Rolf H. Möhring, editor, *Graph-Theoretic Concepts in Computer Science, 16th International Workshop, WG '90, Berlin, Germany, June 20-22, 1990, Proceedings*, volume 484 of *Lecture Notes in Computer Science*, pages 206–220. Springer, 1990. doi:10.1007/3-540-53832-1_44.
- 8 Ofer Biran, Shlomo Moran, and Shmuel Zaks. Tight bounds on the round complexity of distributed 1-solvable tasks. In Jan van Leeuwen and Nicola Santoro, editors, *Distributed Algorithms, 4th International Workshop, WDAG '90, Bari, Italy, September 24-26, 1990, Proceedings*, volume 486 of *Lecture Notes in Computer Science*, pages 373–389. Springer, 1990. doi:10.1007/3-540-54099-7_25.
- 9 Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t-resilient asynchronous computations. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 91–100. ACM, 1993. doi:10.1145/167088.167119.
- 10 Elizabeth Borowsky, Eli Gafni, Nancy A. Lynch, and Sergio Rajsbaum. The BG distributed simulation algorithm. *Distributed Comput.*, 14(3):127–146, 2001. doi:10.1007/PL00008933.
- 11 Armando Castañeda, Damien Imbs, Sergio Rajsbaum, and Michel Raynal. Renaming is weaker than set agreement but for perfect renaming: A map of sub-consensus tasks. In David Fernández-Baca, editor, *LATIN 2012: Theoretical Informatics - 10th Latin American Symposium, Arequipa, Peru, April 16-20, 2012. Proceedings*, volume 7256 of *Lecture Notes in Computer Science*, pages 145–156. Springer, 2012. doi:10.1007/978-3-642-29344-3_13.
- 12 Armando Castañeda, Damien Imbs, Sergio Rajsbaum, and Michel Raynal. Generalized symmetry breaking tasks and nondeterminism in concurrent objects. *SIAM J. Comput.*, 45(2):379–414, 2016. doi:10.1137/130936828.
- 13 Armando Castañeda, Sergio Rajsbaum, and Michel Raynal. The renaming problem in shared memory systems: An introduction. *Comput. Sci. Rev.*, 5(3):229–251, 2011. doi:10.1016/j.cosrev.2011.04.001.
- 14 Armando Castañeda, Sergio Rajsbaum, and Michel Raynal. Unifying concurrent objects and distributed tasks: Interval-linearizability. *J. ACM*, 65(6):45:1–45:42, 2018. doi:10.1145/3266457.
- 15 Armando Castañeda, Sergio Rajsbaum, and Matthieu Roy. Two convergence problems for robots on graphs. In *2016 Seventh Latin-American Symposium on Dependable Computing, LADC 2016, Cali, Colombia, October 19-21, 2016*, pages 81–90. IEEE Computer Society, 2016. doi:10.1109/LADC.2016.21.

- 16 Himanshu Chauhan and Vijay K. Garg. Democratic elections in faulty distributed systems. In Davide Frey, Michel Raynal, Saswati Sarkar, Rudrapatna K. Shyamasundar, and Prasun Sinha, editors, *Distributed Computing and Networking, 14th International Conference, ICDCN 2013, Mumbai, India, January 3-6, 2013. Proceedings*, volume 7730 of *Lecture Notes in Computer Science*, pages 176–191. Springer, 2013. doi:10.1007/978-3-642-35668-1_13.
- 17 Peter J. Denning. Reflections on a Symposium on Computation. *The Computer Journal*, 55(7):799–802, July 2012. doi:10.1093/comjnl/bxs064.
- 18 Peter J. Denning and Peter Wegner. Introduction to What is Computation. *The Computer Journal*, 55(7):803–804, July 2012. doi:10.1093/comjnl/bxs065.
- 19 Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge*. MIT Press, 1995. doi:10.7551/mitpress/5803.001.0001.
- 20 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.
- 21 Lance Fortnow. The enduring legacy of the turing machine. *Comput. J.*, 55(7):830–831, July 2012. doi:10.1093/comjnl/bxs073.
- 22 Pierre Fraigniaud, Ami Paz, and Sergio Rajsbaum. A speedup theorem for asynchronous computation with applications to consensus and approximate agreement. *To appear in ACM PODC 2022*, abs/2206.05356, 2022. To appear in ACM PODC 2022. doi:10.48550/arXiv.2206.05356.
- 23 Pierre Fraigniaud, Sergio Rajsbaum, and Corentin Travers. Locality and checkability in wait-free computing. *Distributed Comput.*, 26(4):223–242, 2013. doi:10.1007/s00446-013-0188-x.
- 24 Dennis J. Frailey. Computation is Process. *The Computer Journal*, 55(7):817–819, July 2012. doi:10.1093/comjnl/bxs069.
- 25 Hugo Rincon Galeana, Sergio Rajsbaum, and Ulrich Schmid. Continuous tasks and the asynchronous computability theorem. In Mark Braverman, editor, *13th Innovations in Theoretical Computer Science Conference, ITCS 2022, January 31 - February 3, 2022, Berkeley, CA, USA*, volume 215 of *LIPICs*, pages 73:1–73:27. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ITCS.2022.73.
- 26 Éric Goubault, Marijana Lazic, Jérémy Ledent, and Sergio Rajsbaum. Wait-free solvability of equality negation tasks. In Jukka Suomela, editor, *33rd International Symposium on Distributed Computing, DISC 2019, October 14-18, 2019, Budapest, Hungary*, volume 146 of *LIPICs*, pages 21:1–21:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.DISC.2019.21.
- 27 Éric Goubault, Jérémy Ledent, and Samuel Mimram. Concurrent specifications beyond linearizability. In Jiannong Cao, Faith Ellen, Luis Rodrigues, and Bernardo Ferreira, editors, *22nd International Conference on Principles of Distributed Systems, OPODIS 2018, December 17-19, 2018, Hong Kong, China*, volume 125 of *LIPICs*, pages 28:1–28:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.OPODIS.2018.28.
- 28 Éric Goubault, Jérémy Ledent, and Sergio Rajsbaum. A simplicial complex model for dynamic epistemic logic to study distributed task computability. *Inf. Comput.*, 278:104597, 2021. doi:10.1016/j.ic.2020.104597.
- 29 Rachid Guerraoui, Michel Hurfin, Achour Mostéfaoui, Rui Carlos Oliveira, Michel Raynal, and André Schiper. Consensus in asynchronous distributed systems: A concise guided tour. In Sacha Krakowiak and Santosh K. Shrivastava, editors, *Advances in Distributed Systems, Advanced Distributed Computing: From Algorithms to Systems*, volume 1752 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 1999. doi:10.1007/3-540-46475-1_2.
- 30 Maurice Herlihy, Dmitry N. Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Morgan Kaufmann, 2013. URL: <https://store.elsevier.com/product.jsp?isbn=9780124045781>.
- 31 Maurice Herlihy and Sergio Rajsbaum. A classification of wait-free loop agreement tasks. *Theor. Comput. Sci.*, 291(1):55–77, 2003. doi:10.1016/S0304-3975(01)00396-6.



- 32 Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, 1999. doi:10.1145/331524.331529.
- 33 Maurice Herlihy and Nir Shavit. *The art of multiprocessor programming*. Morgan Kaufmann, 2008.
- 34 Maurice Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990. doi:10.1145/78969.78972.
- 35 Gunnar Hoest and Nir Shavit. Toward a topological characterization of asynchronous complexity. *SIAM J. Comput.*, 36(2):457–497, 2006. doi:10.1137/S0097539701397412.
- 36 Petr Kuznetsov and Thibault Rieutord. Affine tasks for k-test-and-set. In Stéphane Devismes and Neeraj Mittal, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 151–166, Cham, 2020. Springer International Publishing.
- 37 Butler Lampson. How to build a highly available system using consensus. In Ozalp Babaoglu and Keith Marzullo, editors, *10th International Workshop on Distributed Algorithms (WDAG 1996)*, pages 1–17. Springer, October 1996. The proceedings are: Distributed Algorithms, Lecture Notes in Computer Science 1151, Springer, 1996. URL: <https://www.microsoft.com/en-us/research/publication/how-to-build-a-highly-available-system-using-consensus/>.
- 38 Xingwu Liu, Zhiwei Xu, and Jianzhong Pan. Classifying rendezvous tasks of arbitrary dimension. *Theor. Comput. Sci.*, 410(21-23):2162–2173, 2009. doi:10.1016/j.tcs.2009.01.033.
- 39 Wai-Kau Lo and Vassos Hadzilacos. All of us are smarter than any of us: Nondeterministic wait-free hierarchies are not robust. *SIAM J. Comput.*, 30(3):689–728, 2000. doi:10.1137/S0097539798335766.
- 40 Darya Melnyk, Yuyi Wang, and Roger Wattenhofer. Byzantine preferential voting. In George Christodoulou and Tobias Harks, editors, *Web and Internet Economics*, pages 327–340, Cham, 2018. Springer International Publishing.
- 41 Hammurabi Mendes, Maurice Herlihy, Nitin Vaidya, and Vijay K. Garg. Multidimensional agreement in byzantine systems. *Distributed Computing*, 28(6):423–441, 2015. doi:10.1007/s00446-014-0240-5.
- 42 Mark Moir and Nir Shavit. Concurrent data structures. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004. doi:10.1201/9781420035179.ch47.
- 43 Shlomo Moran and Yaron Wolfstahl. Extended impossibility results for asynchronous complete networks. *Inf. Process. Lett.*, 26(3):145–151, 1987. doi:10.1016/0020-0190(87)90052-4.
- 44 Achour Mostéfaoui, Sergio Rajsbaum, and Michel Raynal. Conditions on input vectors for consensus solvability in asynchronous distributed systems. *J. ACM*, 50(6):922–954, 2003. doi:10.1145/950620.950624.
- 45 Achour Mostéfaoui, Sergio Rajsbaum, and Michel Raynal. Synchronous condition-based consensus. *Distributed Computing*, 18(5):325–343, 2006. doi:10.1007/s00446-005-0148-1.
- 46 Gil Neiger. Set-linearizability. In James H. Anderson, David Peleg, and Elizabeth Borowsky, editors, *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, California, USA, August 14-17, 1994*, page 396. ACM, 1994. doi:10.1145/197917.198176.
- 47 Sergio Rajsbaum and Armajac Raventós-Pujol. A distributed combinatorial topology approach to arrow’s impossibility theorem. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing, PODC ’22*, page to appear, New York, NY, USA, 2022. Association for Computing Machinery.
- 48 Michel Raynal. *Concurrent Programming - Algorithms, Principles, and Foundations*. Springer, 2013. doi:10.1007/978-3-642-32027-9.
- 49 Michel Raynal. *Fault-Tolerant Message-Passing Distributed Systems - An Algorithmic Approach*. Springer, 2018. doi:10.1007/978-3-319-94141-7.

- 50 Michel Raynal. The notion of universality in crash-prone asynchronous message-passing systems: A tutorial. In *2019 38th Symposium on Reliable Distributed Systems (SRDS)*, pages 334–33416, 2019. doi:10.1109/SRDS47363.2019.00046.
- 51 Paul S. Rosenbloom. Computing and Computation. *The Computer Journal*, 55(7):820–824, July 2012. doi:10.1093/comjnl/bxs070.
- 52 Michael E. Saks and Fotios Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *SIAM J. Comput.*, 29(5):1449–1483, 2000. doi:10.1137/S0097539796307698.
- 53 Gadi Taubenfeld and Shlomo Moran. Possibility and impossibility results in a shared memory environment. *Acta Informatica*, 33(1):1–20, 1996. Preliminary version in WDAG 1989. doi:10.1007/s002360050034.
- 54 Peter Wegner. The Evolution of Computation. *The Computer Journal*, 55(7):811–813, July 2012. doi:10.1093/comjnl/bxs067.
- 55 Yunguang Yue, Jie Wu, and Fengchun Lei. The evolution of non-degenerate and degenerate rendezvous tasks. *Topology and its Applications*, 264:187–200, 2019. doi:10.1016/j.topol.2019.06.015.

Sequential Decision Making With Information Asymmetry

Jiarui Gan  

University of Oxford, UK

Rupak Majumdar  

Max Planck Institute for Software Systems (MPI-SWS), Kaiserslautern, Germany

Goran Radanovic  

Max Planck Institute for Software Systems (MPI-SWS), Saarbrücken, Germany

Adish Singla  

Max Planck Institute for Software Systems (MPI-SWS), Saarbrücken, Germany

Abstract

We survey some recent results in sequential decision making under uncertainty, where there is an information asymmetry among the decision-makers. We consider two versions of the problem: persuasion and mechanism design. In persuasion, a more-informed principal influences the actions of a less-informed agent by signaling information. In mechanism design, a less-informed principal incentivizes a more-informed agent to reveal information by committing to a mechanism, so that the principal can make more informed decisions. We define Markov persuasion processes and Markov mechanism processes that model persuasion and mechanism design into dynamic models. Then we survey results on optimal persuasion and optimal mechanism design on myopic and far-sighted agents. These problems are solvable in polynomial time for myopic agents but hard for far-sighted agents.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Bayesian persuasion, Automated mechanism design, Markov persuasion processes, Markov mechanism processes, Myopic agents

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.4

Category Invited Talk

Funding *Rupak Majumdar*: This research was funded in part by the Deutsche Forschungsgemeinschaft project 389792660 TRR 248-CPEC.

1 Introduction

Sequential decision making under uncertainty is a fundamental problem in modeling and analysis of systems. In concurrency theory and formal verification, many such models have been studied extensively. In *Markov decision processes* (MDPs), a single agent observes the state of the world, picks an action, and the new state of the world is determined by an uncertain transition relation. The goal of the agent is to find a policy that optimizes her expected utility, usually over an infinite horizon. In *partially observable* MDPs (POMDPs), the state is no longer perfectly observed; the agent gets a signal about the state of the world and has to find a policy with partial information about the world. Finally, in *stochastic games*, multiple agents play against each other. The objectives of the agents can be zero-sum (the two player, purely adversarial situation) or non-zero sum. The complexity landscape of these models have been studied extensively. Broadly, full information settings (MDPs) are polynomial time solvable [14], partial observation settings are undecidable [20, 4], and games are intermediate in complexity [6, 13, 5].



© Jiarui Gan, Rupak Majumdar, Goran Radanovic, and Adish Singla;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 4; pp. 4:1–4:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

4:2 Sequential Decision Making with Information Asymmetry

There are a number of applications of sequential decision making where the interaction between agents and the world involve *information asymmetry*. These are games of imperfect information on one side, in which one agent influences the behavior of another by selectively signaling additional information about the state of the world, or incentivizes the other to provide accurate information about the world. These models have been largely studied in the economics and artificial intelligence literature, as problems of *persuasion* or of *mechanism design*, but have not received attention in the concurrency theory literature.

In persuasion (also called information design), a knowledgeable principal knows some aspects of the state of the world and interacts with an agent who does not. However, only the agent has the capacity to take an action. Since the objectives of the principal and the agent may be misaligned, the agent may not do the principal's bidding. The goal of the principal is to strategically reveal information about the world, through a process of *signaling*, so that the agent's actions optimize the principal's own interests.

In mechanism design, one or more agents know the state of the world; the principal can take an action based on the report from the agents. Again, it is possible that the agent misrepresents the state of the world to optimize their own payoff. The goal of the principal is to design incentive mechanisms to elicit the agent's private information about the state of the world, so as to make more informed decisions.

If the principal and the agent are completely aligned in their utilities, the signals or the mechanisms involve revealing the unknown information; the more interesting case is when the objectives are misaligned. Persuasion and mechanism design problems in the sequential setting involve partial information and strategic interaction but have not been considered in the concurrency theory literature. The goals of this paper are to provide an introduction to these models, describe some basic results and pointers to the literature, and to point out open problems in the domain.

Persuasion. Kamenica and Gentzkow [17] introduced a fundamental and very influential model of *Bayesian persuasion* as a formal model for persuasion problems. They consider a two player game between a principal and an agent. The players share a common prior on the state of the world, but only the principal observes the realization. The principal commits to a signaling strategy before the game starts. On observing the realization, the principal signals the agent and the agent picks an action based on the signal. They each receive a payoff dependent on the realized state of the world and the action. Kamenica and Gentzkow characterize the optimal signaling strategy of the principal.

Since the publication of this work [17], Bayesian persuasion has seen many applications in the field of economics and algorithmic game theory. The basic model has also been extended in many ways. We refer the reader to the comprehensive survey [16] for pointers to the literature. Our focus in this survey is on algorithmic problems in dynamic models, where persuasion is performed repeatedly over time. Work in this direction is fairly new [12, 23, 15, 26].

Mechanism Design. In automated mechanism design, we consider models where the roles of the players are reversed: now, the principal is the receiver of information, and commits to a mechanism that specifies the action they will take upon receiving each signal. The agent is the signal sender and, knowing the principal's mechanism, sends signals optimally in response. Intuitively, to design a good mechanism requires balancing between the goals of eliciting more information from the agent and of acting optimally based on the elicited information. The principal aims to find a mechanism that maximizes their overall utility from the interaction.

The model follows the line of work on automated mechanism design, initiated by Conitzer and Sandholm [7, 8]. It is shown in their work that the problem of computing an optimal mechanism is NP-hard in general, in settings that allow restrictions to be placed on what signals can be sent given the true state of the world. We consider models without such restrictions, which are less expressive in this regard but arguably also captures a wide range of applications. Following the seminal work of Conitzer and Sandholm, variants of their model have been proposed and studied [24, 18, 19, 27, 28]. A recent work of Zhang and Conitzer [28] introduces a dynamic model of automated mechanism design, and studies some fundamental algorithmic questions for this model. There is a broader literature on various forms of dynamic mechanism design in economics. We refer the reader to the comprehensive surveys [22, 2].

Dynamic Models. Most problems in persuasion and mechanism design were studied in the one-shot setting. More recently, *dynamic* versions of these models have been introduced to capture persuasion and mechanism design in sequential decision making [12, 23, 26, 3, 15, 28]. Dynamic models generalize MDPs from a single agent to settings in which a principal and an agent interact, with an information asymmetry between them. The game is played over a state space. In addition, there is an external parameter, chosen from a known prior distribution, that is the source of information asymmetry. In a *Markov persuasion process* (MPP), in each step, the principal observes the realizations of the external parameters and signals the agent to elicit a favorable action. The agent picks the action based on the current state of the MPP and the signal, both the principal and the agent receive a reward, and the game moves to the next state based on a probabilistic transition relation. In a *Markov mechanism process* (MMP), in each step, the agent observes the realizations of the external parameters. The agent is incentivized by the principal to provide true information by a mechanism – a precommitment to act in a certain way. The agent reports the external parameters as a best response to the precommitment, and the principal chooses an action based on this information. Both principal and agent receive a reward, and the game moves to a new state based on the current state and the chosen action.

Dynamic models of persuasion and mechanism design are special cases of stochastic games of incomplete information [1, 25] and many fundamental insights in characterizing optimal strategies carry over. By focusing on the subclass of games with persuasion and mechanism design as the central aspects, we are able to provide specialized algorithmic results that are applicable to many problems of practical interest.

Myopic and Far-sighted Agents. A new aspect in the study of dynamic persuasion and mechanism design problems is the nature of the agent. In models of concurrency, we usually assume that all players are long-lived, that is, survive throughout the game. In MPPs and MMPs, we distinguish between *far-sighted* and *myopic* agents. A far-sighted agent is long lived and optimizes their expected utility in the long run – it is the “usual case” we study in concurrent games.

In contrast, a myopic agent is short-lived, and only interested in optimizing the payoff in the current stage of the game. In a game with myopic agents, the long-lived principal interacts with a sequence of independent myopic agents, one for each time step. As we shall see, decision problems often become easier when we deal with myopic agents.

There is good motivation for studying myopic agents in both persuasion and mechanism design problems. As an example of a dynamic persuasion problem with myopic agents, consider a ride-sharing app, where the application developer is the long-running principal,

and users of the app can be seen as myopic agents. The users are interested in optimizing their current commute times. The application developer may have a different goal, that of minimizing congestion. The application developer may provide a noisy signal about the status of roads to persuade the commuters to choose routes that minimize overall congestion.

As an example of a dynamic mechanism design problem with myopic agents, consider a firm that consults with a research organization to decide upon a product strategy [28]. Each year, the research organization presents its market research. The firm decides to invest in certain directions based on the reports. The goal of the firm is to have a strong long term business while keeping costs low. On the other hand, the research organization's goal can be myopic – to generate as much revenue from the firm each year, by possibly misrepresenting market conditions. A mechanism in this case is a compensation strategy of the firm that ensures each research report truthfully represents market conditions.

Current Status. In this article, we summarize some recent decidability and complexity results for MPPs and MMPs [15, 28, 26]. We shall see that the principal's optimal signaling strategy and optimal mechanism design problems can be solved in polynomial-time in the infinite horizon setting, against myopic agents. In contrast, we can only show some intractability for these problems against far-sighted agents but a complete characterization remains open.

We have collected the basic results of persuasion and mechanism design in this article and we hope it can serve as the starting point for investigating the specification and verification of dynamic models with information asymmetry in the context of concurrency theory.

2 Persuasion: Principal Observes, Agents Act

2.1 One-shot Bayesian Persuasion

The basic persuasion model by Kamenica and Gentzkow [17] considers two agents: Sender and Receiver (who are the principal and agent, respectively). Receiver has a utility function $u(a, \omega)$ that depends on her action a from a fixed set A of available actions, as well as a state of the world ω from a set Ω (chosen by nature). Sender has a utility function $v(a, \omega)$, that also depends on the receiver's action a and ω . Both players share a common prior μ_0 on Ω . Sender does not influence the world by picking an action himself, but influences Receiver by transmitting a *signal*.

A signal, broadly construed, is some information about the state of the world that Sender can transmit to Receiver. Let G be a sufficiently large space of *signal realizations*. A signaling strategy $\pi : \Omega \rightarrow \Delta(G)$ of Sender is a map that associates each realization of the state of the world to a distribution over G . Using π , Sender will send a signal g to Receiver with probability $\pi(\omega, g)$ whenever ω is observed. Intuitively, the strategy specifies a statistical relationship between the state of the world and Receiver's observed data.

For example, one simple signaling strategy is to always reveal the true information, which always sends a deterministic signal g_ω associated with the observed ω (i.e., g_ω is a message saying “The current state of the world is ω .”, and $\pi(\omega)$ is a Dirac delta distribution at g_ω). In contrast, if the same signal is sent irrespective of the realized ω , i.e., $\pi(\omega) = \pi(\omega')$ for all $\omega, \omega' \in \Omega$, then the signaling strategy is completely uninformative: observing the signal gives Receiver no information about the current realization of ω .

The steps of Bayesian persuasion are as follows.

1. Sender and Receiver share a prior μ_0 .
2. Sender picks a signaling strategy $\pi : \Omega \rightarrow \Delta(G)$ and commits to it; Receiver observes π .

3. Nature picks $\omega \sim \mu_0$ and reveals it to Sender.
4. Sender picks $g \sim \pi(\omega)$ according to his commitment.
5. Receiver observes the realized g , and takes some action $a \in A$ (we describe below how the action is chosen).
6. Sender receives utility $v(a, \omega)$ and Receiver receives $u(a, \omega)$.

Upon receiving a signal g , Receiver updates her posterior belief about the state of the world using the Bayes' rule, whereby the following conditional probability is derived:

$$\Pr(\omega \mid g, \pi) = \frac{\mu_0(\omega) \cdot \pi(\omega, g)}{\sum_{\omega' \in \Omega} \mu_0(\omega') \cdot \pi(\omega', g)}. \quad (1)$$

Receiver picks an action $a^*(\Pr(\cdot \mid g, \pi))$ that maximizes $\mathbb{E}_{\omega \sim \Pr(\cdot \mid g, \pi)}[u(a, \omega)]$. By convention, we assume that Receiver breaks ties in favor of Sender when there are multiple optimal actions. Given the choice of Receiver, Sender solves

$$\max_{\pi \in \Pi} \mathbb{E}_{\omega \sim \mu_0} \mathbb{E}_{g \sim \pi(\omega)} v(a^*(\Pr(\cdot \mid g, \pi)), \omega) \quad (2)$$

to optimize her expected utility, where Π is the set of all signaling strategies.

The optimization problem seems complicated at a first glance, since the space G of signals can be arbitrary, and the choice of π influences the utility of Sender both by influencing how the signal realizations are distributed and by influencing the action that Receiver picks based on the signal realization. However, we shall show that the problem can be reduced to an optimization problem of a simpler form.

2.2 The Revelation Principle and Action Advice

According to a standard argument via the revelation principle [21, 17], we can restrict attention to signaling strategies in the form of *action advice* without any loss of generality. Specifically, for any signaling strategy in an arbitrary space of signals, there exists an equivalent strategy π that uses only a finite set $G_A := \{g_a : a \in A\}$ of signal realizations, where each signal g_a corresponds to an action $a \in A$. With the signal g_a , Sender ‘‘advises’’ Receiver to play a . Moreover, we can additionally ensure that π is *incentive compatible* (IC), which means that Receiver is indeed incentivized to take the corresponding action a upon receiving g_a . Formally, π ensures that

$$\mathbb{E}_{\omega \sim \Pr(\cdot \mid g_a, \pi)} u(a, \omega) \geq \mathbb{E}_{\omega \sim \Pr(\cdot \mid g_a, \pi)} u(a', \omega)$$

for all $a' \in A$, or equivalently:

$$\sum_{\omega \in \Omega} \Pr(\omega \mid g_a, \pi) \cdot (u(a, \omega) - u(a', \omega)) \geq 0 \quad \text{for all } a' \in A. \quad (3)$$

In other words, π signals which action Receiver should take and it is designed in a way such that Receiver cannot be better off deviating from the advised action with respect to the posterior belief. (Again, we assume that Receiver breaks ties in favor of Sender, which means following the advice in this case.) We call a signaling strategy that only uses signals in G_A an *action advice*, and call it an IC action advice if it also satisfies (3).

4:6 Sequential Decision Making with Information Asymmetry

In case A and Ω are finite sets, we can write Sender's optimization problem as a linear program (LP) with variables $\{\pi(\omega, g_a) \mid \omega \in \Omega, a \in A\}$ (see, e.g., [11, 10]):

$$\max \sum_{\omega \in \Omega} \sum_{a \in A} \mu_0(\omega) \cdot \pi(\omega, g_a) \cdot v(a, \omega) \quad (4)$$

$$\text{subject to } \sum_{\omega \in \Omega} \mu_0(\omega) \cdot \pi(\omega, g_a) \cdot (u(a, \omega) - u(a', \omega)) \geq 0, \quad \text{for } a, a' \in A \quad (5)$$

$$\sum_{a \in A} \pi(\omega, g_a) = 1, \quad \text{for } \omega \in \Omega \quad (6)$$

$$\pi(\omega, g_a) \geq 0, \quad \text{for } \omega \in \Omega, a \in A \quad (7)$$

The variable $\pi(\omega, g_a)$ denotes the conditional probability of recommending action a when the state of the world is ω . The LP maximizes the expected utility of Sender over the joint distribution of ω and a , subject to incentive compatibility (i.e., (5), where $\Pr(\omega \mid g_a, \pi)$ in (3) is replaced by $\mu_0(\omega) \cdot \pi(\omega, g_a)$ according to (1)). Since linear programming can be solved in polynomial time, the above formulation shows that one-shot persuasion can be solved in polynomial time when the actions and the external parameters are given explicitly.

► **Theorem 2.1** [11]. *Sender's optimization problem can be solved in polynomial time in $|A|$ and $|\Omega|$.*

More generally, Kamenica and Gentzkow showed a characterization of the optimal function for compact action spaces and payoff functions that are continuous in the action [17].

Given a signal, each signal realization g_a induces a posterior belief $\mu_a \in \Delta(\Omega)$. The marginal probability of signal realization g_a is $\Pr[g_a] = \sum_{\omega \in \Omega} \mu_0(\omega) \cdot \pi(\omega, a)$ and the posterior distribution $\Pr(\omega \mid g_a, \pi) = \frac{\mu_0(\omega) \cdot \pi(\omega, g_a)}{\Pr[g_a]}$.

Thus, we can think of a feasible solution of the LP as a distribution over posteriors (an element of $\Delta(\Delta(\Omega))$), one per signal realization, whose expectation equals the prior μ_0 (such a distribution of posteriors is called *Bayes plausible*). Thus, if μ_0 is represented as a point in the simplex $\Delta(\Omega)$, then the signal corresponds to writing μ_0 as a convex combination of posterior distributions in $\Delta(\Omega)$. The incentive compatibility constraints ensure that action a is preferred by Receiver on the posterior distribution on Ω induced by a .

Each posterior distribution $\mu \in \Delta(\Omega)$ is associated with a preferred action $a^*(\mu)$ for Receiver, i.e., the action that maximizes $\mathbb{E}_{\omega \sim \mu} u(a, \omega)$. We can plot Sender's utility as a function $V : \Delta(\Omega) \rightarrow \mathbb{R}$ of the posterior: $V(\mu) = \mathbb{E}_{\omega \sim \mu} v(a^*(\mu), \omega)$. Define $\text{cav}(V)$ as the *concavification* of V : the pointwise smallest concave function that is an upper bound for V . Equivalently,

$$\text{cav}(V)(\mu) = \sup\{z : (\mu, z) \in \text{co}(V)\} \quad (8)$$

where $\text{co}(V)$ is the convex hull of the graph of V . The convex hull $\text{co}(V)$ is the set of pairs (μ, z) such that if the prior is μ , there exists a signal with value z . Thus, $\text{cav}(V)(\mu_0)$ is the optimal utility that Sender can achieve when the prior is μ_0 .

This is a very general result, holding also for compact spaces of actions and continuous reward functions. It also follows from an older result on games of imperfect information studied by Aumann and Maschler [1].

Note that if V is already concave, then Sender reveals no information. For example, in the zero-sum case when the utility functions of Sender and Receiver sum to zero, V is concave. On the other hand, if the Sender and Receiver have completely aligned utility functions, V is convex and Sender reveals all information.

In general, we do not know how to compute the concavification of an arbitrary function $V : \Delta(\Omega) \rightarrow \mathbb{R}$. If the graph of V is semi-algebraic (defined by a Boolean combination of polynomial inequalities), we can use techniques from the theory of reals, using the characterization that the concavification of V evaluated at μ is $\sup\{z \mid (\mu, z) \in \text{co}(V)\}$ and that $\text{co}(V)$ is a semi-algebraic set if the graph of V is semi-algebraic.

The above LP assumes that the world is given explicitly. In case the world is given symbolically, as valuations to a set of variables, it still works if we assume that the prior has small (polynomial-size in the size of the problem) support. The optimization problem can sometimes be solved even when this assumption is not true. Consider the case in which $u(a, \omega)$ and $v(a, \omega)$ are real-valued random variables that can be arbitrarily correlated. We say actions are independent if $u(a) = u(a, \omega)$ and $u(a') = u(a', \omega)$ are independent random variables for distinct actions $a \neq a'$, and the same is true for $v(a) = v(a, \omega)$ and $v(a') = v(a', \omega)$. Then, the distribution μ_0 is fully specified by the marginal distribution of the pair $(u(a), v(a))$ for each action a . We assume that each action's marginal distribution has finite support, and refer to each element of the support as a *type*.

Dughmi and Xu [11] show that in case $u(a)$ and $u(a')$ are independent and identically distributed (IID) for $a \neq a'$, and $v(a)$ and $v(a')$ are also IID, Sender's optimization problem can be solved in polynomial time in the number of actions n and the number of types m . This is non-trivial, since the above LP has exponentially many (m^n) states of the world. On the other hand, the problem becomes #P-hard if the distributions are arbitrary.

2.3 Examples

Prosecution. Kamenica and Gentzkow [17] give an example of Bayesian persuasion in a courtroom setting. A prosecutor (Sender) is trying to convince a judge (Receiver) that a defendant is guilty. When the defendant is guilty, revealing all the evidence will help the prosecutor, but when the defendant is innocent, revealing all the evidence will likely hurt the prosecutor's case. Kamenica and Gentzkow show that when the prosecutor and the judge are rational Bayesian, a prosecutor can organize their argument to increase the probability of conviction.

Concretely, assume that the judge has two actions: *acquit* or *convict*. The states of the world correspond to the defendant's status: *guilty* or *innocent*. The judge gets a utility of 1 for choosing the just action (convict the guilty and acquit the innocent) and utility 0 for the unjust action. The prosecutor gets a utility of 1 if the judge convicts and 0 otherwise – regardless of the defendant's status. Assume that the prior $\Pr[\textit{guilty}] = 0.3$ is common knowledge.

We model the prosecutor's possible investigations into the case as distributions $\pi(\cdot \mid \textit{guilty})$ and $\pi(\cdot \mid \textit{innocent})$. The prosecutor has to pick π and truthfully report the realization to the judge (the *commitment* step). (It is required by law that the prosecutor cannot hide evidence, even it makes a conviction unlikely.)

If there is no communication, e.g., if the investigation is completely uninformative, the judge always acquits, since innocence is more likely than guilt according to the prior. If the investigation is fully informative, i.e., reveals the defendant's status with probability 1, then the judge convicts 30% of the time. However, suppose that the prosecutor picks an investigation as follows:

$$\begin{aligned} \pi(\textit{acquit} \mid \textit{innocent}) &= \frac{4}{7} & \pi(\textit{acquit} \mid \textit{guilty}) &= 0 \\ \pi(\textit{convict} \mid \textit{innocent}) &= \frac{3}{7} & \pi(\textit{convict} \mid \textit{guilty}) &= 1 \end{aligned}$$

4:8 Sequential Decision Making with Information Asymmetry

This constitutes an IC action advice for the judge. Notice that the judge convicts with probability 60% (Bayes' rule!). This is true even though the judge knows that 70% of defendants are innocent and even though the judge is fully aware that the prosecutor's advice (the signal) is designed to maximize the probability of conviction!

Traffic Control. Das et al. [9] describe a simple example of persuasion to improve congestion in uncertain traffic conditions. Imagine a traffic network with two paths between a source and an origin. Travel time on Path I is independent of the number of agents using it, but depends on an uncertain state of nature (e.g., Path I is a highway that is prone to repair). Travel time on Path II depends on the number of agents taking the path: the more agents take the path, the more time it takes. The goal of Sender (a social planner) is to signal the state of Path I to the agents so that the congestion on Path II is reduced to a social optimum. Hence, each agent is an individual Receiver, and they are modeled as non-atomic players, who individually is a zero-measure and have negligible influence to the system (but collectively their influence integrates).

Let us be more precise. There are two paths P_1 and P_2 , and the state of the world is $\omega \in \{0, 1\}$, both states are equally likely. The travel times are given by $c(P_1) = \omega$ and $c(P_2) = \frac{1}{3} + 2s$. Agents seek to minimize their travel costs.

If Sender can mandate how everyone drives, the socially optimum cost is calculated as follows. If $\omega = 0$, everyone uses P_1 and the total cost is zero. If $\omega = 1$, the socially optimum move is to send $\frac{1}{6}$ of the agents to P_2 so that the aggregate cost is $\frac{17}{18}$. Thus, the expected aggregate travel cost is $\frac{17}{36}$.

Suppose Sender provides exact information. Then, when $\omega = 1$, agents will crowd P_2 until the costs of the two paths are equalized: $\frac{1}{3} + 2s = 1$, or $s = \frac{1}{3}$. The aggregate cost is 1 and therefore the expected aggregate cost is $\frac{1}{2}$, which is worse than the optimum.

Now consider the following signaling strategy.

$$\begin{aligned} \pi(\text{take } P_1 \mid \omega = 0) &= 1 & \pi(\text{take } P_2 \mid \omega = 0) &= 0 \\ \pi(\text{take } P_1 \mid \omega = 1) &= \frac{5}{6} & \pi(\text{take } P_2 \mid \omega = 1) &= \frac{1}{6} \end{aligned}$$

(Namely, when $\omega = 1$, we send the message "take P_1 " to $5/6$ of the agents and "take P_2 " to the rest.) Then, when $\omega = 0$, everyone takes P_1 and the cost is zero. When $\omega = 1$, we expect $\frac{1}{6}$ fraction to go on P_2 . The overall expected cost is the same as the optimal: $\frac{17}{36}$. Thus, the social planner persuades some fraction of people to take P_1 .

We observe that the signal is incentive compatible. Upon seeing the advice "take P_1 " the expectation of the cost of P_1 is

$$\Pr[\omega = 1 \mid \text{take } P_1] \cdot 1 = \frac{\frac{5}{6}}{\frac{5}{6} + 1} = \frac{5}{11}$$

(where $\Pr[\omega = 1 \mid \text{take } P_1]$ is the posterior belief given π) and the expectation of the cost of P_2 is

$$\frac{1}{3} + 2 \left(\Pr[\omega = 0 \mid \text{take } P_1] \cdot 0 + \Pr[\omega = 1 \mid \text{take } P_1] \cdot \frac{1}{6} \right) = \frac{16}{33} > \frac{5}{11}$$

Thus, the agent should pick P_1 . Similarly, on seeing "take P_2 ", the expectation of P_1 is 1 and the expectation of P_2 is $\frac{2}{3} < 1$. Thus, the agent should again pick P_2 .

2.4 Markov Persuasion Processes

We now extend the model of Bayesian persuasion to the sequential setting. Our formal model, called *Markov persuasion processes* (MPP),¹ is an MDP with reward uncertainties, given by a tuple

$$\mathcal{M} = \langle S, A, P, \Omega, (\mu_s)_{s \in S}, u, v \rangle \quad (9)$$

that represents the repeated interaction between Sender and Receiver.

Similar to a standard MDP, S is a finite state space; A is a finite action space available to Receiver; $P : S \times A \times S \rightarrow [0, 1]$ is the transition dynamics of the state. When the environment is in state s and Receiver takes action a , the state transitions to s' with probability $P(s, a, s')$; both Sender and Receiver are aware of the state throughout. Meanwhile, rewards are generated for both Sender and Receiver, and are specified by the reward functions $u : S \times \Omega \times A \rightarrow \mathbb{R}$ and $v : S \times \Omega \times A \rightarrow \mathbb{R}$, respectively. That is, unlike in a standard MDP, the rewards in our setting also depend on an external parameter $\omega \in \Omega$ (akin to the state of the world in the basic model). This parameter captures an additional layer of uncertainty of the environment. At each state $s \in S$, we assume that the parameter follows a distribution $\mu_s \in \Delta(\Omega)$ and is drawn anew every time the state changes. μ_s is common prior knowledge shared between Sender and Receiver, but only Sender has access to the realization of ω .

Since the actions are taken only by Receiver, Sender does not directly influence the state. As in Bayesian persuasion, Sender influences Receiver's action by signaling. We only consider *Markovian signaling strategies*, whereby signals only depend on the current state (independent of the history). As in the one-shot case, a revelation theorem argument shows that Sender only needs to consider IC action advice at each state.

Formally, a signaling strategy $\pi = (\pi_s)_{s \in S}$ of Sender consists of a function $\pi_s : \Omega \rightarrow \Delta(G_A)$ for each state $s \in S$. Sender will commit to a strategy before the start of play. In every step, upon observing the realization of the external parameter ω , Sender will send an action advice sampled from $\pi_s(\omega)$ when the current state is s .

2.5 Optimal Signaling Problem

Similarly to the one-shot setting, we take Sender's point of view and investigate the problem of optimal signaling strategy design: given \mathcal{M} , find a signaling strategy π that maximizes Sender's (discounted) cumulative reward. The cumulative reward is defined as

$$\mathbb{E} \left[\sum_{t=0}^T \gamma^t \cdot v(s_t, a_t, \omega_t) \mid \mathbf{z}, \pi, P \right], \quad (10)$$

where $\mathbf{z} = (z_s)_{s \in S}$ is the distribution of the starting state, $\gamma \in [0, 1]$ is a discount factor, T is a given horizon, and the expectation is taken over the trajectory $(s_t, \omega_t, a_t)_{t=0}^T$ induced by \mathbf{z} , the signaling strategy π , and the dynamics P . If T is finite, we call the problem the *finite horizon* setting, and if T is infinite, we call the setting *infinite horizon*.

Finally, we introduce a behavioral model for Receiver. We will consider two major types of Receivers – *myopic* and *far-sighted*. A myopic Receiver only cares about their instant reward in each step, whereas a far-sighted Receiver considers the cumulative reward with respect to a discount factor $\tilde{\gamma} > 0$ (which need not be equal to γ).

¹ The nomenclature comes from [26].

In summary, the game proceeds as follows. At the beginning, Sender commits to a signaling strategy π and announces it to Receiver. Then in each step, an external parameter $\omega \sim \mu_s$ is drawn (by nature) according to the state $s \in S$ of the MPP; Sender observes $\omega \in \Omega$, samples an action advice $g \sim \pi_s(\omega)$, and sends g to Receiver. Receiver receives g , updates their belief about ω and decides an action $a \in A$ to take. Sender receives $v(s, \omega, a)$ and Receiver receives $u(s, \omega, a)$. The state then transitions to $s' \sim P(s, a, \cdot)$, which both players observe. The game proceeds until the horizon T (or forever, if $T = \infty$).

2.6 Solving the Optimal Signaling Problem

2.6.1 Myopic Receiver

We first consider the case where Receiver is myopic. In this case, Receiver aims to maximize her reward in each individual step. Upon receiving a signal g in state s , Receiver takes a best action $a \in A$, which maximizes the immediate expected reward $\mathbb{E}_{\omega \sim \text{Pr}(\cdot | g, \pi_s)} u(s, \omega, a)$. Think of a myopic Receiver as a sequence of “short-lived” Receivers, one for each time step. Receiver in step t plays a one-shot Bayesian persuasion game with Sender, collects their reward, and disappears.

We consider the problem of computing an optimal signaling strategy in an infinite-horizon MPP ($T = \infty$) with a myopic Receiver. We call this problem $\text{OPTIMALSIGNALING}_\infty\text{-MYOPIC}$.

► **Theorem 2.2** [15]. $\text{OPTIMALSIGNALING}_\infty\text{-MYOPIC}$ can be solved in polynomial time.

The proof of Theorem 2.2 is via a reduction from the problem to linear programming. The approach is as follows.

We can easily characterize the outcome of an IC action advice π : at each state s , since Receiver is incentivized to follow the advice, with probability $\phi_s^\pi(\omega, a) := \mu_s(\omega) \cdot \pi_s(\omega, g_a)$ they will take action a when the realized external parameter is ω . Thus, ϕ_s^π is a distribution over $\Omega \times A$.

We then define the following set $\mathcal{A}_s \subseteq \Delta(\Omega \times A)$, which contains all such distributions that can be induced by some IC action advice:

$$\mathcal{A}_s = \{\phi_s^\pi \in \Delta(\Omega \times A) : \pi \text{ is an IC action advice}\}.$$

We can now view the problem facing Sender as an (single-agent) MDP

$$\mathcal{M}^* = \langle S, (\mathcal{A}_s)_{s \in S}, P^*, v^* \rangle,$$

where S is the same state space in \mathcal{M} ; \mathcal{A}_s defines an (possibly infinite) action space for each s ; the transition dynamics $P^* : S \times \Delta(\Omega \times A) \times S \rightarrow [0, 1]$ and reward function $v^* : S \times \Delta(\Omega \times A) \rightarrow \mathbb{R}$ are such that

$$P^*(s, \mathbf{x}, s') = \mathbb{E}_{(\omega, a) \sim \mathbf{x}} P(s, a, s') \quad \text{and} \quad v^*(s, \mathbf{x}) = \mathbb{E}_{(\omega, a) \sim \mathbf{x}} v(s, a, \omega)$$

for any $\mathbf{x} \in \mathcal{A}_s$. Namely, \mathcal{M}^* is defined as if Sender can choose actions (which are (ω, a) pairs) freely from \mathcal{A}_s , whereas the choice is actually realized through persuasion. A policy σ for \mathcal{M}^* maps each state s to an action $\mathbf{x} \in \mathcal{A}_s$, and it corresponds to an IC action advice π in \mathcal{M} , with $\phi_s^\pi = \sigma(s)$ for all s . The problem of designing an optimal action advice then translates to computing an optimal policy for \mathcal{M}^* .

The standard approach to computing an optimal policy for an MDP is to compute a value function $V : S \rightarrow \mathbb{R}$ that satisfies the Bellman equation:

$$V(s) = \max_{\mathbf{x} \in \mathcal{A}_s} \left[v^*(s, \mathbf{x}) + \gamma \cdot \sum_{s' \in S} P^*(s, \mathbf{x}, s') \cdot V(s') \right] \quad \text{for all } s \in S.$$

There exists a unique solution to the above system of constraints, from which an optimal policy can be extracted. The solution is posed as the following linear program over variables $\{V(s) : s \in S\}$:

$$\min \sum_{s \in S} z_s \cdot V(s) \quad (11)$$

$$\text{subject to } V(s) \geq v^*(s, \mathbf{x}) + \gamma \cdot \sum_{s' \in S} P^*(s, \mathbf{x}, s') \cdot V(s') \quad \text{for all } s \in S, \mathbf{x} \in \mathcal{A}_s \quad (12)$$

The optimal value of this LP directly gives the cumulative reward of optimal policies under a given initial state distribution \mathbf{z} .

The issue with this LP formulation is that there may be infinitely many constraints as (12) must hold for all $\mathbf{x} \in \mathcal{A}_s$. This is unlike MDPs with a finite action space, where there are a finite number of constraints, one for each action.

Gan et al. [15] show that LP (11) can nevertheless be solved in polynomial time by using the ellipsoid method. The key to this approach is to implement the *separation oracle* in polynomial time. For any given value assignment of the variables (in the above LP, values of $V(s)$), the oracle should decide correctly whether all the constraints of the LP are satisfied or not and, if not, output a violated one.

Implementing the separation oracle for the LP requires solving $\max_{\mathbf{x} \in \mathcal{A}_s} v^*(s, \mathbf{x}) + \gamma \cdot \sum_{s' \in S} P^*(s, \mathbf{x}, s') \cdot V(s') - V(s)$ for all $s \in S$: by checking if the maximum value is positive, we can identify if (12) is violated for some $\mathbf{x} \in \mathcal{A}_s$. Indeed, the set of IC action advice can be characterized by (5)–(7). Hence, we obtain the following LP implementation of the separation oracle, where $\{x(\omega, a) : \omega \in \Omega, a \in A\}$ and $\{\pi_s(\omega, g_a) : \omega \in \Omega, a \in A\}$ are the variables.

$$\begin{aligned} \max \quad & v^*(s, \mathbf{x}) + \gamma \cdot \sum_{s' \in S} P^*(s, \mathbf{x}, s') \cdot V(s') - V(s) \\ \text{s.t.} \quad & x(\omega, a) = \mu_s(\omega) \cdot \pi_s(\omega, g_a) && \text{for all } \omega \in \Omega, a \in A, s \in S \\ & \sum_{\omega \in \Omega} \mu_s(\omega) \cdot \pi_s(\omega, g_a) \cdot (u(s, a, \omega) - u(s, a', \omega)) \geq 0, && \text{for } a, a' \in A, s \in S \\ & \sum_{a \in A} \pi_s(\omega, g_a) = 1, && \text{for } \omega \in \Omega, s \in S \\ & \pi_s(\omega, g_a) \geq 0, && \text{for } \omega \in \Omega, a \in A, s \in S \end{aligned}$$

Since the ellipsoid method runs in polynomial time, the tractability of OPTIMALSIGNALING_∞-MYOPIC follows immediately. By exploiting the duality of linear programming, one can provide a different, “direct” encoding into a linear programming problem as well (see [15]).

► **Remark 2.3 Finite Horizon.** When the horizon is finite, one can set up the Bellman equation and evaluate it by backward induction. Each step in the process solves a one-shot persuasion problem using the linear programming formulation. This gives a polynomial time algorithm when the time horizon is given in unary. Wu et al. [26] study several variants of this problem, as well as the setting of reinforcement learning.

► **Remark 2.4.** In the *reachability problem* for Markov persuasion processes, there is a subset of marked states and Sender receives a unit reward if and only if one of these states is reached along a trajectory. The reachability problem asks what is the expected probability that the subset is reached. The above linear programming formulation can be used to solve the reachability problem against myopic Receivers. Since the reachability problem is at the

core of model checking logics on MDPs, we should be able to build up a logic on Markov persuasion processes and obtain efficient model checking algorithms in case of myopic agents. We leave the design of appropriate logics and model checking, as well as the computation of optimal signals for omega-regular properties, as future work.

2.6.2 Far-sighted Receiver

A far-sighted (FS) Receiver looks beyond the immediate reward and optimizes the cumulative reward

$$\mathbb{E} \left[\sum_{t=0}^T \tilde{\gamma}^t \cdot u(s_t, a_t, \omega_t) \mid \mathbf{z}, \pi, P \right], \quad (13)$$

where, as in (10), $\mathbf{z} = (z_s)_{s \in S}$ is the distribution of the starting state, $\tilde{\gamma} \in [0, 1)$ is a discount factor possibly different from Sender's discount factor, T is the horizon, and the expectation is taken over the trajectory $(s_t, a_t, \omega_t)_{t=0}^T$ induced by the initial distribution \mathbf{z} , the signaling strategy π , and the dynamics P .

When facing an FS Receiver, we cannot define a set \mathcal{A}_s independently for each state. Sender needs to take a global view and aim to induce Receiver to use a *policy* that benefits Sender. We consider the problem of optimal signaling strategy design in an infinite horizon setting against an FS Receiver, called $\text{OPTIMALSIGNALING}_\infty\text{-FS}$.

At this point, we know very little about the decidability and complexity of this problem or a characterization of optimal strategies. For example, we know that Sender can do better with history-dependent signaling. We also know that the problem is hard.

► **Theorem 2.5** [15]. *Assuming that $P \neq NP$, $\text{OPTIMALSIGNALING}_\infty\text{-FS}$ does not admit any polynomial-time $\frac{1}{\lambda^{1-\epsilon}}$ -approximation algorithm for any constant $\epsilon > 0$, where λ is the number of states $s \in S$ in which the prior distribution μ_s is non-deterministic (i.e., supported on at least two external parameters). This holds even when $|\Theta| = 2$ and the discount factors $\gamma, \tilde{\gamma} \in (0, 1)$ are fixed.*

The proof of Theorem 2.5 is via a reduction from the $\text{MAXIMUM INDEPENDENT SET}$ problem, which is known to be NP-hard to approximate [29].

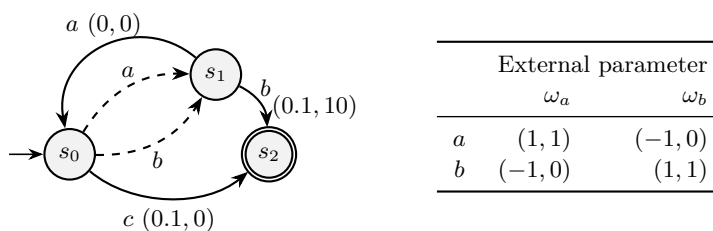
2.6.3 Advice-myopic Receiver

Between the tractable (myopic) Receivers and the intractable (FS) Receivers lie the *advice-myopic* Receivers. An advice-myopic (AM) Receiver accounts for the cumulative future rewards just as an FS Receiver, but behaves myopically in ignoring the future signals of Sender. In other words, an AM Receiver always assumes that Sender will disappear in the next step and relies only on their own prior knowledge to estimate any future payoff.

► **Theorem 2.6** [15]. *$\text{OPTIMALSIGNALING}_\infty\text{-AM}$ is solvable in polynomial time.*

The idea is that, since an AM Receiver does not consider future signals, their future reward is independent of Sender's signaling strategy. One can compute the future payoff in polynomial time by fixing the uninformative signal for Sender and solving the resulting MDP. This payoff is added to the reward function of the AM Receiver, but now we can consider Receiver to be myopic since the future payoffs have been taken into account.

The interest in AM Receiver is that an optimal signaling policy of Sender assuming an AM Receiver can be used to define a strategy against an arbitrary FS Receiver. The idea is to provide a threat: if Receiver ever deviates from the action advice, Sender will forever provide only uninformative signals. One can show that this threat strategy enables Sender to get an expected payoff that is at least as much against any AM Receiver.



■ **Figure 1** A simple example from [15].

The threat strategy uses one bit of memory (to remember if Receiver had deviated from the advice). However, this threat-based strategy may not be an optimal one-memory strategy. Indeed, for any positive integer k , the problem of computing an optimal k -memory strategy against FS Receivers is inapproximable (via an adapted version of the reduction for proving Theorem 2.5). In contrast, in the myopic and advice-myopic settings, since Receiver’s behavior is Markovian, the optimal signaling strategies we designed remain optimal even when we are allowed to use memory-based strategies.

2.7 Example

Figure 1 shows a simple example to distinguish myopic, far-sighted, and advice-myopic Receivers. In the MPP, Sender wishes to reach s_2 while maximizing rewards. Transitions are deterministic. Each edge is labeled with the corresponding action and (in the brackets) rewards for Receiver and Sender, respectively. The rewards for state-action pairs (s_0, a) and (s_0, b) (dashed edges) also depend on the 2-valued state of the world $\{\omega_a, \omega_b\}$, as specified in the table. The state of the world is sampled uniformly at random at each step. Assume discount factor $\frac{1}{2}$ both for Sender and for Receivers.

With no signaling, Receiver will always take action c in s_0 , so Sender will obtain payoff 0. Sender can reveal information about the external parameter to attract Receiver to move to s_1 . If Receiver is myopic, Sender can reveal full information, which leads to Receiver moving to s_1 , taking action b , and ending in s_2 . As a result, Sender obtains payoff 6.

However, if Receiver is FS, this strategy will not work. Receiver will loop between s_0 and s_1 , resulting in overall payoff $4/3$ for Sender. To improve, Sender can choose to be less informative in s_0 , e.g., advising Receiver to take the more profitable action 10% of the time and a uniformly sampled action in $\{a, b\}$ the remaining 90% of the time. Receiver will move to s_1 under this signaling, breaking ties in favor of Sender. Sender’s expected payoff is 5.55.

Alternatively, Sender can also use the following threat-based strategy, which again yields a payoff of 6. Sender always reveals the true information in s_0 , advises Receiver to take b in s_1 , and threatens to stop providing any information if Receiver does not follow the advice. The outcome of this strategy coincides with how an advice-myopic Receiver behaves. Such a Receiver will choose b at s_1 as future disclosures are not considered.

2.8 Extensions to the Model

In our model of MPPs thus far, the external parameter ω is picked independently at each step. We can envision a more general model, in which the external parameter also evolves according to a stochastic process. For example, we can assume that the external parameter evolves according to a Markov chain. Such extensions have been studied [12, 23], but we do not know of any general algorithmic results.

One can show that against myopic Receivers, the optimal value can be calculated on a Markov process on the space of distributions in $S \times \Delta(\Omega)$; the initial belief is the initial distribution of the state of the world and the value function maps beliefs to values and is the fixpoint of a functional mapping beliefs to beliefs. The functional is a contraction map on a suitable topological space, and therefore the fixpoint exists and is unique. While one can approximately evaluate the fixpoint numerically, we do not know how to characterize the complexity of the decision problem. Since the belief space $\Delta(\Omega)$ is infinite, we can no longer set up a (finite) linear programming problem nor argue about termination of the iterations.

3 Mechanism: Agent Observes, Principal Acts

A dual scenario of persuasion is one where Receiver is the principal and Sender is the agent. In this case Receiver can commit to a mechanism to influence Sender's signaling behavior. A mechanism $\sigma : G \rightarrow \Delta(A)$ is a map from Sender's signal space G to a distribution over the action space A , which specifies how Receiver will act, upon receiving each signal from Sender.

3.1 One-shot Mechanism Design

In the one-shot setting, the steps in this scenario are as follows.

1. Sender and Receiver share a prior μ_0 .
2. Receiver picks a mechanism $\sigma : G \rightarrow \Delta(A)$ and commits to it; Sender observes σ .
3. Nature picks $\omega \sim \mu_0$ and reveals it to Sender.
4. Sender observes ω and sends a signal $g \in G$ (we describe below how this signal is chosen).
5. Receiver observes g and takes an action $a \sim \sigma(g)$ according to her commitment.
6. Sender receives utility $v(a, \omega)$ and Receiver receives $u(a, \omega)$.

In Step 4, as a rational player, Sender best-responds to the mechanism σ , sending a signal so that the action taken by Receiver in Step 5 maximizes Sender's payoff in expectation. Namely, the following signal is sent:

$$g \in \arg \max_{g \in G} \mathbb{E}_{a \sim \sigma(g)} v(a, \omega). \quad (14)$$

Here, one subtlety, similar to the one in the persuasion setting, is that there is actually no predefined signal space or one that is agreed upon between the two players, so the mechanism is not well-defined if Sender picks a signal outside of G . The revelation principle then comes in again, which now says that it is without loss of generality to consider *direct mechanisms*, whereby the signal space is restricted to a finite set $G_\Omega := \{g_\omega : \omega \in \Omega\}$; each signal $g_\omega \in G_\Omega$ corresponds to a realization of the state of the world. In other words, the interaction in Step 4 can be viewed as an information elicitation process, where Receiver asks Sender: what is the realization of the external parameter? Sender answers ω by sending the corresponding signal g_ω .

Specifically, given an arbitrary mechanism $\sigma : G \rightarrow \Delta(A)$, an equivalent mechanism $\varsigma : G_\Omega \rightarrow \Delta(A)$ can be constructed by letting $\varsigma(g_\omega) = \sigma(f(\omega))$ for all $\omega \in \Omega$, where $f : \Omega \rightarrow G$ is a map defined by (14) (by fixing an arbitrary tie-breaking rule to select g in case there are multiple optimal signals). It is not hard to see that ς induces an equivalent signaling behavior of Sender and the same payoffs in Step 6. Moreover, it also elicits truthful information from Sender, incentivizing Sender to send g_ω whenever the realization is ω .

In summary, the revelation principle indicates that it is without loss of generality to consider mechanisms that are both direct and IC. Given this result, the problem of computing an optimal mechanism for Receiver can be formulated as the following LP with variables $\{\sigma(g_\omega, a) : \omega \in \Omega, a \in A\}$, i.e., $\sigma(g_\omega, a)$ is the probability of Receiver taking action a upon receiving g_ω .

$$\max \sum_{\omega \in \Omega} \sum_{a \in A} \mu_0(\omega) \cdot \sigma(g_\omega, a) \cdot u(a, \omega) \quad (15)$$

$$\text{subject to } \sum_{a \in A} \sigma(g_\omega, a) \cdot v(a, \omega) \geq \sum_{a \in A} \sigma(g_{\omega'}, a) \cdot v(a, \omega), \quad \text{for } \omega, \omega' \in A \quad (16)$$

$$\sum_{a \in A} \sigma(g_\omega, a) = 1, \quad \text{for } a \in A \quad (17)$$

$$\sigma(g_\omega, a) \geq 0 \quad \text{for } \omega \in \Omega, a \in A \quad (18)$$

The formulation takes a form symmetric to LP (4). The first constraint requires σ to be IC.

3.2 Markov Mechanism Process

Moving to the dynamic setting, we consider the same MDP $\mathcal{M} = \langle S, A, P, \Omega, (\mu_s)_{s \in S}, u, v \rangle$ as in (9). Receiver commits to a state-dependent mechanism $\sigma_s : G_\Omega \rightarrow \Delta(A)$. At every step, both players observe the state s of \mathcal{M} , and nature samples an external parameter $\omega \sim \mu_s$. Sender observes ω and sends a signal g to Receiver. Receiver plays an action $a \sim \sigma_s(g)$ according to a pre-committed state-dependent mechanism. Consequently, rewards $v(s, a, \omega)$ and $u(s, a, \omega)$ are generated for the players, and \mathcal{M} transitions to a next state $s' \sim P(s, a, \cdot)$. We ask the infinite-horizon optimal mechanism design problem from Receiver's perspective. In what follows we present a polynomial-time algorithm for this problem when Sender is myopic. The approach is similar to the LP-based algorithm for `OPTIMALSIGNALING∞-MYOPIC`.

3.3 Optimal Mechanism Design for Myopic Sender

Call the optimal mechanism design problem `OPTIMALMECHANISM∞-MYOPIC` when Sender is myopic.

► **Theorem 3.1.** `OPTIMALMECHANISM∞-MYOPIC` can be solved in polynomial time.

The proof is similar to that of Theorem 2.2. We reduce the problem to linear programming and use the ellipsoid method. We define the set of possible outcomes of a direct IC mechanism σ as follows:

$$\mathcal{A}_s = \{\phi_s^\sigma \in \Delta(\Omega \times A) : \sigma \text{ is a direct IC mechanism}\},$$

where ϕ_s^σ is a distribution with $\phi_s^\sigma(\omega, a) := \mu_s(\omega) \cdot \sigma_s(g_\omega, a)$ being the probability that Receiver takes action a while the realized external parameter is ω . The problem facing Receiver then reduces to an (single-agent) MDP $\mathcal{M}^* = \langle S, (\mathcal{A}_s)_{s \in S}, P^*, u^* \rangle$, where the transition dynamics P^* and reward function u^* are such that $P^*(s, \mathbf{x}, s') = \mathbb{E}_{(\omega, a) \sim \mathbf{x}} P(s, a, s')$, and $u^*(s, \mathbf{x}) = \mathbb{E}_{(\omega, a) \sim \mathbf{x}} u(s, \omega, a)$ for any $\mathbf{x} \in \mathcal{A}_s$. The following LP, similar to LP (11), is then devised to compute an optimal mechanism (with variables $\{V(s) : s \in S\}$).

$$\min \sum_{s \in S} z_s \cdot V(s) \quad (19)$$

$$\text{subject to } V(s) \geq u^*(s, \mathbf{x}) + \gamma \cdot \sum_{s' \in S} P^*(s, \mathbf{x}, s') \cdot V(s') \quad \text{for } s \in S, \mathbf{x} \in \mathcal{A}_s \quad (20)$$

The separation oracle of this LP can further be implemented by solving the following LP for all $s \in S$, where $\{x(\omega, a) : \omega \in \Omega, a \in A\}$ and $\{\sigma_s(g_\omega, a) : \omega \in \Omega, a \in A\}$ are the variables.

$$\begin{aligned} \max \quad & u^*(s, \mathbf{x}) + \gamma \cdot \sum_{s' \in S} P^*(s, \mathbf{x}, s') \cdot V(s') - V(s) \\ \text{subject to} \quad & x(\omega, a) = \mu_s(\omega) \cdot \sigma_s(g_\omega, a) && \text{for } s \in S, \omega \in \Omega, a \in A \\ & \sum_{a \in A} \sigma_s(g_\omega, a) \cdot v(s, a, \omega) \geq \sum_{a \in A} \sigma_s(g_{\omega'}, a) \cdot v(s, a, \omega), && \text{for } \omega, \omega' \in A, s \in S \\ & \sum_{a \in A} \sigma_s(g_\omega, a) = 1, && \text{for } a \in A, s \in S \\ & \sigma_s(g_\omega, a) \geq 0 && \text{for } \omega \in \Omega, a \in A, s \in S \end{aligned}$$

► **Remark 3.2.** Zhang and Conitzer [28] studied a more general model in the finite-horizon case and consider history-dependent mechanisms. In their model, Receiver cannot observe the state of the MDP and has to rely on Sender to make observations; essentially, the state is equivalent to the external parameter in our model but follows a stochastic process. They show that the problem is polynomial time solvable in the finite horizon case when Sender is myopic, but NP-hard to approximate when Sender is FS. They also characterize optimal mechanisms and show that the optimal mechanism against an FS sender depends on the history of state-action trajectories, as well as the current state. Note that the NP-hardness does not imply the hardness of the optimal mechanism design problem we defined against an FS Sender, where the goal is to compute an optimal Markov mechanism for an infinite horizon, whereas the external parameter is sampled independently in each step. We leave the complexity of this problem open for future work.

4 Conclusion

We have described some basic results in the theory of Markov decision processes with information asymmetry. We show that in the two settings we study, persuasion and mechanism design, one can obtain optimal signaling policy and optimal mechanism design in polynomial time against myopic agents. As we point out throughout the article, many algorithmic questions in these domains remain open. While the models have been applied to many problems in economics and game theory, their applications to system design have not been explored so far. We hope our article can act as a starting point for studying these models and their algorithmic properties, in the context of concurrency theory and system design.

References

- 1 Robert J. Aumann and Michael B. Maschler. *Repeated Games with Incomplete Information*. MIT Press, 1995.
- 2 Dirk Bergemann and Juuso Välimäki. Dynamic mechanism design: An introduction. *Journal of Economic Literature*, 57(2):235–74, 2019.
- 3 Andrea Celli, Stefano Coniglio, and Nicola Gatti. Private Bayesian persuasion with sequential games. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI'20)*, pages 1886–1893, 2020.
- 4 Krishnendu Chatterjee, Martin Chmelik, and Mathieu Tracol. What is decidable about partially observable markov decision processes with ω -regular objectives. *J. Comput. Syst. Sci.*, 82(5):878–911, 2016. doi:10.1016/j.jcss.2016.02.009.
- 5 Krishnendu Chatterjee and Thomas A. Henzinger. A survey of stochastic ω -regular games. *J. Comput. Syst. Sci.*, 78(2):394–413, 2012. doi:10.1016/j.jcss.2011.05.002.

- 6 Anne Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992. doi:10.1016/0890-5401(92)90048-K.
- 7 Vincent Conitzer and Tuomas Sandholm. Complexity of mechanism design. In Adnan Darwiche and Nir Friedman, editors, *Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence (UAI'02)*, pages 103–110. Morgan Kaufmann, 2002.
- 8 Vincent Conitzer and Tuomas Sandholm. Self-interested automated mechanism design and implications for optimal combinatorial auctions. In *Proceedings of the 5th ACM Conference on Electronic Commerce (EC'04)*, pages 132–141, 2004.
- 9 Sanmay Das, Emir Kamenica, and Renee Mirka. Reducing congestion through information design. In *Proceedings of the 55th Allerton Conference on Communication, Control, and Computing*, pages 1279–1284, 2017.
- 10 S. Dughmi. Algorithmic information structure design. *ACM SIGecom Exch.*, 15(2):2–24, 2017.
- 11 Shaddin Dughmi and Haifeng Xu. Algorithmic Bayesian persuasion. In Daniel Wicks and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 412–425. ACM, 2016. doi:10.1145/2897518.2897583.
- 12 J. Ely. Beeps. *American Economic Review*, 107(1):31–53, 2017.
- 13 Kousha Etessami and Mihalis Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM J. Comput.*, 39(6):2531–2597, 2010. doi:10.1137/080720826.
- 14 J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag, 1997.
- 15 Jiarui Gan, Rupak Majumdar, Goran Radanovic, and Adish Singla. Bayesian persuasion in sequential decision-making. In *Proceedings of the 36th AAAI Conference on Artificial Intelligence, (AAAI'22)*. AAAI Press, 2022.
- 16 Emir Kamenica. Bayesian persuasion and information design. *Annual Review of Economics*, 11:249–272, 2019.
- 17 Emir Kamenica and Matthew Gentzkow. Bayesian persuasion. *American Economic Review*, 101(6):2590–2615, 2011.
- 18 Andrew Kephart and Vincent Conitzer. Complexity of mechanism design with signaling costs. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems (AAMAS'15)*, pages 357–365, 2015.
- 19 Andrew Kephart and Vincent Conitzer. The revelation principle for mechanism design with reporting costs. In *Proceedings of the 2016 ACM Conference on Economics and Computation (EC'16)*, pages 85–102, 2016.
- 20 Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and related stochastic optimization problems. *Artif. Intell.*, 147(1-2):5–34, 2003. doi:10.1016/S0004-3702(02)00378-8.
- 21 Roger B. Myerson. Incentive compatibility and the bargaining problem. *Econometrica*, 47(1):61–73, 1979.
- 22 Alessandro Pavan. Dynamic mechanism design: Robustness and endogenous types. In *Advances in Economics and Econometrics: Eleventh World Congress*, volume 1, pages 1–62, 2017.
- 23 J. Renault, E. Solan, and N. Vieille. Optimal dynamic information provision. *Games and Economic Behavior*, 104:329–349, 2017.
- 24 Tuomas Sandholm, Vincent Conitzer, and Craig Boutilier. Automated design of multistage mechanisms. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*, volume 7, pages 1500–1506, 2007.
- 25 Sylvain Sorin. *A First Course on Zero-Sum Repeated Games*. Springer, 2008.
- 26 Jibang Wu, Zixuan Zhang, Zhe Feng, Zhaoran Wang, Zhuoran Yang, Michael I. Jordan, and Haifeng Xu. Sequential information design: Markov persuasion process and its efficient reinforcement learning. *CoRR*, abs/2202.10678, 2022. arXiv:2202.10678.
- 27 Hanrui Zhang, Yu Cheng, and Vincent Conitzer. Automated mechanism design for classification with partial verification. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI'21)*, volume 35(6), pages 5789–5796, 2021.

4:18 Sequential Decision Making with Information Asymmetry

- 28 Hanrui Zhang and Vincent Conitzer. Automated dynamic mechanism design. *Advances in Neural Information Processing Systems (NeurIPS'21)*, 34, 2021.
- 29 David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC'06)*, pages 681–690. Association for Computing Machinery, 2006.

Involved VASS Zoo

Wojciech Czerwiński  

University of Warsaw, Poland

Abstract

We briefly describe recent advances on understanding the complexity of the reachability problem for vector addition systems (or equivalently for vector addition systems with states - VASSes). We present a zoo of a few involved VASS examples, which illustrate various aspects of hardness of VASSes and various techniques of proving lower complexity bounds.

2012 ACM Subject Classification Theory of computation → Parallel computing models

Keywords and phrases vector addition systems, reachability problem, low dimensions, examples

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.5

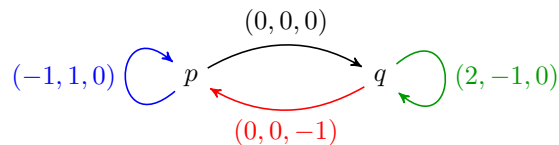
Category Invited Talk

Funding *Wojciech Czerwiński*: Supported by the ERC grant INFSYS, agreement no. 950398.

1 Introduction

Vector addition systems and essentially equivalent Petri nets are one of the most natural models of computation. They are also widely used in practise [21]. A convenient way to work with vector addition systems is to consider its extension by states (which is also essentially equivalent), namely vector addition systems with states (VASSes). A d -dimensional VASS (shortly a d -VASS) is a finite automaton equipped with d integer counters. Each transition can increase or decrease the counters by fixed values. Importantly, no counter can be ever decreased below zero. The counter represents the current number of items of some resource in the modelled system, thus it is natural to assume that this number is nonnegative.

► **Example 1.** The following 3-VASS was introduced in [9], we call it the *HP-gadget* after the names of authors of [9]. This VASS has interesting properties, which we use in the sequel. Transition colours are just to distinguish particular transitions, they have no semantics in the VASS behaviour.



The following is an example of a run

$$p(2, 0, 7) \xrightarrow{\text{blue}} p(1, 1, 7) \xrightarrow{\text{black}} p(0, 2, 7) \xrightarrow{\text{black}} q(0, 2, 7) \xrightarrow{\text{green}} q(2, 1, 7) \xrightarrow{\text{green}} q(4, 0, 7) \xrightarrow{\text{red}} p(4, 0, 6)$$

Observe that in a similar way there is a run from $p(k, 0, n)$ to $p(2k, 0, n - 1)$: we apply k times the blue transition reaching $p(0, k, n)$, then once the black transition reaching $q(0, k, n)$, then k times the green transition reaching $q(2k, 0, n)$ and finally once the red transition reaching $p(2k, 0, n - 1)$. Intuitively in the state p we transfer value k from the first counter to the second one and then jump to state q . In the state q we transfer back value k to the



© Wojciech Czerwiński;

licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Sławomir Lasota, and Anca Muscholl; Article No. 5; pp. 5:1–5:13

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

first counter while multiplying it by 2. Finally we jump back to the state p decreasing the third counter by one. We will use similar approach many times in the sequel. Notice that repeating this process n times we have the following run

$$p(1, 0, n) \longrightarrow p(2, 0, n - 1) \longrightarrow \dots \longrightarrow p(2^{n-1}, 0, 1) \longrightarrow p(2^n, 0, 0),$$

where each black arrow represents a sequence of transitions (in the sequel we often draw a sequence of transitions as one arrow). We also have

$$p(2^n, 0, 0) \longrightarrow p(x, y, 0)$$

for any $x + y = 2^n$, so the set of configurations reachable from $p(1, 0, n)$ is of at least exponential size.

On the other hand the size of the *reachability set* of $p(1, 0, n)$ (set of configurations reachable from $p(1, 0, n)$) is finite. Indeed, the red transition can be fired at most n times and it is easy to see that in between of two firings of the red transition all the other transitions also have to be fired only finitely many times. Thus the above example is the first interesting one: the reachability set is finite, but of at least exponential size (in that case of exactly exponential size).

Various decision problems for VASSes are studied since the 70-ties (with the proviso that in those times they were known under the name of Petri nets). Probably the most central one is the *reachability problem*. It asks whether in a given VASS there is a run from a given source *configuration*, to a given target configuration. A configuration is a state together with a counter valuation. Another related fundamental problem is the *coverability problem*, which asks whether in a given VASS there is a run from a given source configuration to a configuration which is *above* a given target configuration. We say that one configuration is above the other one if it has the same state, but counter values may be higher.

2 History of the problem

The reachability and coverability problems are considered since the 70-ties. The first milestone result was ExpSpace-hardness of the coverability problem by Lipton in 1976 [17]. Notice that this implies ExpSpace-hardness of the reachability problem, as coverability can be reduced to reachability by adding to a VASS additional transitions decreasing counters in the target state (one transition for each counter). In 1978 Rackoff has proven that the coverability problem is in ExpSpace [19]. He achieved it by showing that if there is a run from the source configuration s to some configuration $t' \succeq t$ (namely t' is above t) then there is also some *short* run from the source configuration s to some configuration $t'' \succeq t$, where by short we mean at most doubly-exponential in the input size. This approach, by small witness (which is often a short run) turns out to be successful in many cases for the reachability problem in VASSes. In 1982 finally decidability of the reachability problem was proved by Mayr [18]. The construction was very involved, so the follow-up works by Kosaraju and Lambert tried to simplify the solution and phrase it in a bit simpler setting [10, 11]. This construction is currently often known by the name KLM decomposition, as it decomposes the input VASS into many simpler ones.

After these breakthrough results there was a long period of not much progress on the reachability problem. The community tried to improve the state of art, but it was hard, so results about VASSes are scarce in the 90-ties. In 2009 Haase et al. proved that in 1-VASSes with numbers on transitions encoded in binary (we call such VASSes binary) the reachability

problem is NP-complete [8]. It is easy to show that for unary 1-VASSes the problem is NL-complete. More progress on low dimensional VASSes followed. In 2015 Blondin et al. proved that in binary 2-VASSes the reachability problem is PSpace-complete [1], while a year later this result was improved by Englert et al. to NL-completeness in unary 2-VASSes [6]. Both the upper complexity bounds in dimension two were shown by the use of short run approach: authors of [1] proved that if there is any run from the source to the target in binary 2-VASS then there is also one of at most exponential length, while in [6] the same was shown for unary 2-VASSes and polynomial length runs.

Recently there was also a big progress in fixing complexity of the reachability problem. In 2015 Leroux and Schmitz have obtained first complexity upper bound on the problem [15]. By careful analysis of the KLM decomposition algorithm they proved that it runs in cubic-Ackermann time. In 2019 the same authors improved their previous result. They proposed a slight modification of the KLM decomposition algorithm and elegantly analysing the dimension a some vector spaces proved that the modified version runs in Ackermann time [16]. Also in 2019 Czerwiński et al. proved that the reachability problem is Tower-hard [2]. This was a surprise as many people felt that the problem should rather be ExpSpace-complete, but we probably lack some insight to prove the upper bound. In [2] we have used the technique of multiplication triples described later. Just two years later the complexity of the problem was finally settled to be Ackermann-complete. Two teams have independently shown Ackermann-hardness using slightly different techniques: Leroux [13] and Czerwiński and Orlikowski [4]. In [4] we have used the technique of controlling-counter and amplifiers, the technique of controlling-counter is described later.

3 Remaining challenges

Despite the fact that the complexity of the reachability problem in VASSes was established the problem still remains elusive in my opinion. The gap in our understanding is most striking in dimension three. For binary 2-VASSes the problem is PSpace-complete [1]. However for binary 3-VASSes the best complexity lower bound is still PSpace-complete inherited from the dimension two, while the best known upper bound is higher than Tower, namely in the \mathcal{F}_7 complexity class of the fast growing hierarchy [16]. We define the hierarchy of fast growing functions as $F_1(n) = 2n$ and $F_{k+1}(n) = \underbrace{F_{k-1} \circ \dots \circ F_{k-1}}_n(1)$ for any $k > 1$. One can easily

see that in particular $F_2(n) = 2^n$ and $F_3(n) = \text{Tower}(n)$. Based on the hierarchy of fast growing functions F_i one defines a hierarchy of fast growing complexity classes \mathcal{F}_i , which roughly speaking is the class of problems solvable in time F_i closed under a few natural operations [20]. Thus in particular we do not know whether existence of a run from the source to the target always implies existence of exponential length run or not. Or maybe length of this short run is doubly-exponential or tower size. Similarly we lack knowledge about other low dimensions.

Generally in dimension d the best upper bound for the reachability problem is \mathcal{F}_{d+4} [16] (that is how we get \mathcal{F}_7 in dimension three). The current best lower complexity bound is \mathcal{F}_d -hardness in dimension $2d + 4$, so $\mathcal{F}_{(d-4)/2}$ -hardness for d -VASSes [14]. The current research goal here is to find out whether we can get \mathcal{F}_d -hardness in dimension $d + C$ for some constant $C \in \mathbb{N}$.

Recently we worked with co-authors on the reachability problem for low dimensional VASSes [3, 5] motivated by the following two main ideas: 1) low dimensional VASSes are by itself a natural computation model, 2) understanding problems in low dimensional VASSes often turns out to be the best way of developing techniques very useful in general

dimension. Indeed, understanding low dimensions was actually the triggering point for our results [2] and [4]. Current best complexity lower bounds for low dimensional VASSes are proven in our work with Łukasz Orlikowski [4]:

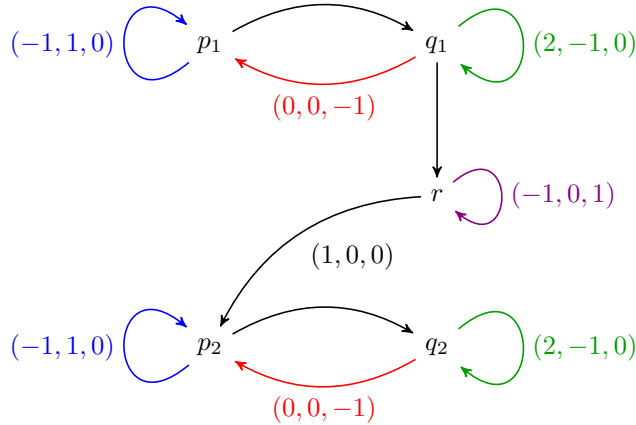
- NP-hardness for unary 4-VASSes;
- PSpace-hardness for unary 5-VASSes;
- ExpSpace-hardness for binary 6-VASSes;
- Tower-hardness for unary 8-VASSes.

The rest of this text focuses on presenting techniques of proving lower complexity bounds from the perspective of concrete low dimensions or concrete examples of involved low dimensional VASSes. We believe this perspective is the best way to illustrate the intuitions behind various approaches and to introduce various techniques useful in general.

4 Big finite reachability sets

We start the involved examples zoo from a family of examples, which is a folklore since years. These are VASSes, which have finite reachability set, but this set is very big. We first present a 3-VASS with finite, but doubly-exponential reachability set. For simplicity we do not write a vector on the transition if it does not change the counters at all (we often colour such transitions black).

► **Example 2.** The following 3-VASS has doubly-exponential reachability set.



Notice that the above example consists of two copies of the HP-gadget from Example 1. Thus we have the following run:

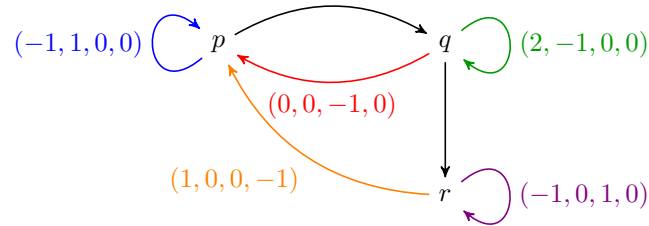
$$\begin{aligned}
 p_1(1, 0, n) &\longrightarrow \dots \longrightarrow q_1(2^n, 0, 0) \longrightarrow r(2^n, 0, 0) \longrightarrow \dots \longrightarrow r(0, 0, 2^n) \\
 &\longrightarrow p_2(1, 0, 2^n) \longrightarrow \dots \longrightarrow q_2(2^{2^n}, 0, 0).
 \end{aligned}$$

In other words in the first copy of the HP-gadget from $p_1(1, 0, n)$ we reach $p_2(2^n, 0, 0)$. Then in state r we transfer value from the third counter to the first one. The transition from r to p_2 adds one to the first counter such that we start from $p_2(1, 0, 2^n)$ in the second copy of the HP-gadget.

It is easy to show that the reachability set of $p_1(1, 0, n)$ is finite, the proof goes as in Example 1.

In a similar way one can construct a 3-VASS which has k -fold exponential reachability set, we just take k copies of the HP-gadget and connect them by states r_i as above. However this requires a growing number of states in a VASS. Here comes another idea: by adding just one additional counter we can simulate any number of copies on this counter.

► **Example 3.** The following 4-VASS has finite, but tower size reachability set. It is just a slight modification of Example 2.



In this 4-VASS we have added the fourth counter and the only transition which modifies this counter is the orange transition. The rest is exactly like in the HP gadget with additional state r . Thus for any k we have to following run:

$$p(1, 0, k, n) \longrightarrow \dots \longrightarrow q(2^k, 0, 0, n) \longrightarrow r(2^k, 0, 0, n) \longrightarrow \dots \longrightarrow r(0, 0, 2^k, n) \longrightarrow p(1, 0, 2^k, n-1).$$

In other words we can exponentiate the first counter for the cost of decreasing the fourth counter by one. Thus for any $n \in \mathbb{N}$ there is also the following run:

$$p(1, 0, 1, n) \longrightarrow p(2, 0, 1, n-1) \longrightarrow p(4, 0, 1, n-2) \longrightarrow \dots \longrightarrow p(\text{Tower}(n), 0, 1, 0).$$

This easily implies that the reachability set from $p(1, 0, 1, n)$ is of at least $\text{Tower}(n)$ size. It remains to show that this reachability set is finite. To see this notice first that the orange transition can be fired at most n times. Now it is easy to see that in between of any two firings of the orange transition other transitions can be fired at most exponentially many times wrt. the current counter values, which finishes the argument.

The Example 3 already shows that a very simple VASS can have a pretty complicated behaviour. It is not hard to see that in a similar vein one can construct in any dimension d a unary d -VASS with finite reachability set of size around $F_{d-1}(n)$, where n is the size of the source configuration.

5 Finite reachability sets are enough

It is a good moment to emphasise that authors of [16] not only have shown that the reachability problem in d -VASSes can be solved in \mathcal{F}_{d+4} , but they proved that if there is a run from the source to the target then there is also one of length bounded by roughly speaking $F_{d+4}(n)$. Using this result and the generalised Example 3 one can show that VASSes with finite reachability sets are actually not much simpler than VASSes without that restriction. More concretely speaking one can reduce the reachability problem for d -VASSes to the reachability problem for $(d+6)$ -VASSes with finite reachability sets. Assume we need to check whether $s \longrightarrow t$ in a d -VASS V . We construct a $(d+6)$ -VASS U as follows. First part of U behaves like generalised Example 3 in dimension $d+5$, thus on one of the counters (say counter number $d+5$) can have values up to $F_{d+4}(n)$. We use the last $(d+6)$ -th counter

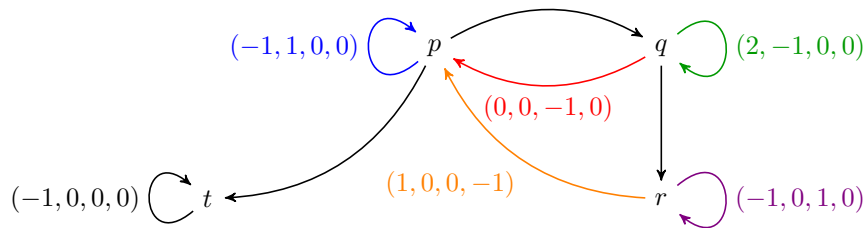
to keep the sum of all the dimensions numbered from 1 to $d + 4$. In the second part U simulates V on dimensions from 1 to d . In the target configuration of U we demand that dimension $d + 6$ is equal to zero, so after the first part all the dimensions from 1 to $d + 4$ need also to be zero. The only change of the second part of U wrt. to V is that to simulate any transition of V in U we decrease the $(d + 5)$ -th counter by one. Notice now that if there is a run from s to t in U by [16] there is also one of length at most $F_{d+4}(n)$ thus there is also one in V . Of course no run in U implies no run in V as the simulation is faithful. On the other hand the reachability set of any configuration in V is finite as in each step we decrease the $(d + 5)$ -th counter. This finishes the argument.

The above reasoning does not show that considering VASSes with finite reachability sets is enough, because we have added six additional dimensions. However it suggests that in order to understand well low dimensions it might be sufficient to look sometimes at this special case of finite reachability set. Notice that this is a strong statement, as the reachability problem can be easily solved for VASSes with finite reachability set: we just compute the whole set of configurations reachable from the source and after this computation stops (it has to, as the reachability set is finite) we check whether the target belongs to the set. Moreover we have a pretty good complexity upper bounds for this very naive algorithm. By [7] the longest sequence of configurations in a d -VASS without a *domination* (situation that a configuration further in the sequence is strictly bigger than a configuration earlier in the sequence) is bounded roughly speaking by $F_{d+1}(n)$, where n upper bounds the size of VASS and the source configuration. Notice that in VASSes with finite reachability set no run has a domination, as domination allows for pumping counters up and would imply an infinite reachability set. Thus [7] shows that exploring the whole space of reachable configurations in a d -VASS can be achieved in the complexity class \mathcal{F}_{d+1} .

Notice however that for 3-VASSes even assuming finite reachability set we still get complexity \mathcal{F}_4 , which is much higher than the known lower bound of PSpace-hardness. Thus there might be a possibility of constructing a 3-VASS or other lower dimensional VASS with shortest run being exponential, doubly exponential or even Tower length. Below we show a few current, still very weak, techniques which can lead in the future to some involved examples.

6 Telescope equations

Example 3 and its generalisations exhibit a complicated behaviour of low dimensional VASSes. Notice however that it does not eliminate a possibility that in low dimensional VASSes there are always some short paths. Imagine the following slight modification of Example 3.



From Example 3 we know that in the above VASS there are $\text{Tower}(n)$ -long paths from $p(1, 0, 1, n)$ to $t(0, 0, 1, 0)$: such a path first reaches $p(\text{Tower}(n), 0, 1, 0)$, then goes to $t(\text{Tower}(n), 0, 1, 0)$ and then in a loop decreases the first counter. However there are also some very short runs: we n times apply the sequence

$$p(\ell, 0, 1, k) \longrightarrow q(\ell, 0, 1, k) \longrightarrow r(\ell, 0, 1, k) \longrightarrow p(\ell + 1, 0, 1, k - 1)$$

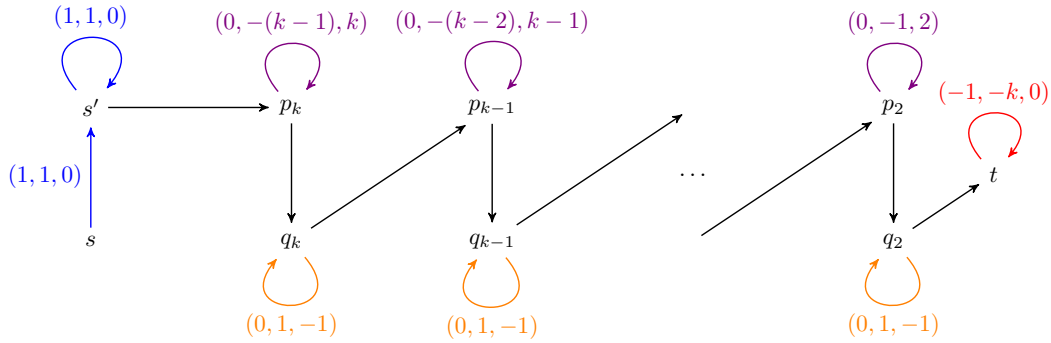
then go to state t and quickly decrease the first counter.

This illustrates the main challenge with proving lower bounds for the reachability problem in VASSes: it is very hard to force a VASS to take some long run from the source to the target. Here we present one approach how to force a VASS to have only long runs, the example is taken from [3]. It is based on the following simple telescope equation:

$$k = \frac{k}{k-1} \cdot \frac{k-1}{k-2} \cdot \dots \cdot \frac{3}{2} \cdot \frac{2}{1}. \quad (1)$$

Based on (1) we build a 3-VASS V_k with size of all the transitions bounded by k and a property that the shortest path from the source to the target is of length exponential in k .

► **Example 4.** In this example the source configuration is $s(0, 0, 0)$ and the target configuration is $t(0, 0, 0)$.



Let us analyze how a run from the source to the target in VASS V_k can look like. In the state s we fire the blue transition to $s'(1, 1, 0)$ and then some number $N - 1$ times the blue loop in state s' reaching the configuration $s'(N, N, 0)$. So the prefix of our run is the following

$$s(0, 0, 0) \longrightarrow s'(1, 1, 0) \longrightarrow \dots \longrightarrow s(N, N, 0) \longrightarrow p_k(N, N, 0).$$

States s and s' are distinguished to assure that $N \geq 1$. Notice now that the only other transition in V_k which modifies the first counter is the red transition in state t . Thus the considered run need to finish in the following way:

$$q_2(N, Nk, 0) \longrightarrow t(N, Nk, 0) \longrightarrow \dots \longrightarrow t(0, 0, 0).$$

Observe now that for each $i \in \{2, \dots, k\}$ the orange transitions in q_i do not change the sum of the second and the third counter while the violet transitions in p_i can multiply this sum by at most $\frac{i}{i-1}$. Moreover this is the case if and only if the run enters p_i in the configuration of the form $p_i(0, K, 0)$ where K is divisible by $i - 1$ and leaves it in the configuration of the form $p_i(0, 0, K \cdot \frac{i}{i-1})$. In other words in the state p_i the whole value of second counter needs to be transferred to the third counter while multiplying it by $\frac{i}{i-1}$. Notice now that from $p_k(N, N, 0)$ till $q_2(N, Nk, 0)$ the second counter needs to be multiplied by exactly k . Using Equation (1) we derive that in any run from $p_k(N, N, 0)$ to $q_2(N, Nk, 0)$ in all the states the

whole value of the second counter have to be transferred to the third counter or vice versa. In particular each loop have to be fired the maximal number of times. So the run needs to look as follows:

$$\begin{aligned}
p_k(N, N, 0) &\longrightarrow p_k(N, 0, \frac{Nk}{k-1}) \longrightarrow q_k(N, 0, \frac{Nk}{k-1}) \longrightarrow q_k(N, \frac{Nk}{k-1}, 0) \longrightarrow p_{k-1}(N, \frac{Nk}{k-1}, 0) \\
&\longrightarrow p_{k-1}(N, 0, \frac{Nk}{k-2}) \longrightarrow q_{k-1}(N, 0, \frac{Nk}{k-2}) \longrightarrow q_{k-1}(N, \frac{Nk}{k-2}, 0) \longrightarrow p_{k-2}(N, \frac{Nk}{k-2}, 0) \\
&\dots \\
&\longrightarrow p_3(N, 0, \frac{Nk}{2}) \longrightarrow q_3(N, 0, \frac{Nk}{2}) \longrightarrow q_3(N, \frac{Nk}{2}, 0) \longrightarrow p_2(N, \frac{Nk}{2}, 0) \\
&\longrightarrow p_2(N, 0, Nk) \longrightarrow q_2(N, 0, Nk) \longrightarrow q_2(N, Nk, 0).
\end{aligned}$$

Now notice that in the run for each $i \in \{2, \dots, k-1\}$ we have a configuration $q_i(N, \frac{Nk}{i}, 0)$, which means that Nk is divisible by each $i \in \{2, \dots, k-1\}$. Thus Nk is a multiplicity of the $\text{lcm}(2, \dots, k-1)$, which is known to be exponential wrt. k (see [3], Claim 6). This finishes the proof that any run from the source to the target needs to be of length exponential wrt. k .

The above example can also be expressed by another formalism, which is often much more convenient to present VASSes then drawing them as automata. This formalism is called the counter programs. We do not introduce counter programs formally, instead we present VASS from Example 4 as a counter program hoping that this clarifies the issue. We assume that the three counters are named x , y and z . For more details look into [3].

```

1: x += 1   y += 1
2: loop
3:   x += 1   y += 1
4: for i := k down to 2 do
5:   loop
6:     y -= i - 1   z += i
7:   loop
8:     y += 1   z -= 1
9: loop
10:  x -= 1   y -= k

```

Using similar trick with the telescope equation (but a bit more involved) we have shown in [3] an example a 4-VASS in which the shortest run from the source to the target is of doubly-exponential length.

7 Controlling-counter

VASSes, in contrast to counter machines lack zero-tests, thus it is pretty hard to force their runs to be exact. Notice that with zero-tests we can easily force the modified Example 3 (mentioned in paragraph Telescopic equations) to have only runs of Tower length. We just enforce that all the loops are fired maximally by zero-testing appropriate counters after the loops. Of course we cannot hope to simulate zero-tests by VASSes as VASSes with zero-tests (called counter machines) have undecidable reachability problem.

However, we are able to simulate some restricted number of zero-tests in VASSes. First of all notice that in the reachability problem we ask whether we can reach the target configuration, so we already have some very weak form of zero-tests: if we set the target configuration to be zero at some counter then we can test this counter to be zero at the end of the run. Now the idea is to boost this single zero-test to simulate more zero-tests during the run.

Let us assume that we have a d -VASS V with some the counter x and we want to zero-test counter x in some three moments during the run. First very naive idea is to add three additional counters x_1, x_2, x_3 to V , which are copies of x and modify them exactly as x . The first one is stopped being modified after the first moment, the second one is not modified after the second moment and the third one is not modified after the third moment. In this way if in the modified $(d + 3)$ -VASS we set the target configuration to be zero on counters x_1, x_2, x_3 then we enforce that any run reaching the target indeed have value zero in the three considered moments. The main drawback of this idea is that it introduces additional counters, so is too costly. However, already this technique illustrates that zero-test in the target configuration can be used to simulate zero-tests in other moments in the run.

Here we introduce the technique of the controlling-counter, which was proposed in [4]. Assume we have a run ρ in our d -VASS V of the following form:

$$s \xrightarrow{\rho_1} c_1 \xrightarrow{\rho_2} c_2 \xrightarrow{\rho_3} c_3 \xrightarrow{\rho_4} t$$

and we want to zero-test the counter x in the configurations c_1, c_2, c_3 . Let us assume that the value of the counter x in the source configuration s is zero. Let the value of the counter x in configurations c_i be x_i , for $i \in \{1, 2, 3\}$. We need to check whether $x_1 = x_2 = x_3 = 0$. Notice that it is enough to check if $x_1 + x_2 + x_3 = 0$ as all the counter values x_i are nonnegative. Let Δ_i be the effect of the run ρ_i on the counter x . Thus we have $x_1 = \Delta_1$, $x_2 = \Delta_1 + \Delta_2$ and $x_3 = \Delta_1 + \Delta_2 + \Delta_3$. Therefore $x_1 + x_2 + x_3 = 3\Delta_1 + 2\Delta_2 + \Delta_3$ and it is enough to check whether this expression has value zero. In order to do that we introduce one additional *controlling-counter* y which is tested for zero in the target configuration t . We set the value of the counter y in the configuration s to be zero. Each change of x by C in ρ_1 is matched by change of y by $3C$. Similarly, each change of x by C in ρ_2 is matched by change of y by $2C$. Finally, each change of x by C in ρ_3 is matched by change of y by the same value C . Thus indeed final value of y is exactly $3\Delta_1 + 2\Delta_2 + \Delta_3$ and it is enough to check y for zero in the target configuration in order to assure that $x_1 = x_2 = x_3 = 0$.

It is easy to observe that this reasoning can be extended to any number of zero-tests. In general if we are in the part of the run ρ such that after this part still k zero-tests are performed on x then each change of x by C needs to be matched by the change of y by $k \cdot C$. We only need that configurations c_1, c_2, c_3, \dots are distinguishable in the sense that we can change behaviour of counter y after any c_i . This can be often easily implemented by use of states.

It is also not hard to see that one controlling-counter can control many original counters, not just one.

Below we present the simplest possible application of the controlling-counter to 3-VASSes. Consider the following 2-VASS with two counters x and y starting in the counter valuation $(x, y) = (1, 0)$.

```

1: for  $i := 1$  to  $k$  do
2:   loop
3:      $x -= 1$     $y += 2$ 
4:   loop
5:      $x += 1$     $y -= 1$ 
6: loop
7:    $x -= 1$ 

```

5:10 Involved VASS Zoo

It is easy to see that if all the loops are fired maximally then before entering line 6 counter values are $(x, y) = (2^k, 0)$ and loop in lines 6-7 can be fired 2^k times. Thus if we want to reach values $(0, 0)$ at the end of the counter program there exists an exponential run. However, there is also a very short run, the one totally ignoring the loops in lines 2-3 and in lines 4-5 and immediately jumping to the loop in lines 6-7 which is fired just once. However, introducing a controlling-counter z we may enforce the loops to be fired maximal number of times and thus obtain another example of a VASS with shortest one run being exponential.

► **Example 5.** In the 2-VASS above both counters x and y are tested exactly k times. Thus as the starting valuation is $(x, y) = (1, 0)$ we should start from value $z = k$. Therefore in the i -th iteration of the for-loop in the line 3 the counter x is still waiting for $k - (i - 1)$ zero-tests as well as the counter y . Similarly as in the line 3 the counter x in the line 5 is still waiting for $k - i$ zero-tests while the counter y is waiting for $k - (i - 1)$ zero-tests. Therefore in the line 3 we should increase z by $(-1) \cdot (k - i + 1) + 2 \cdot (k - i + 1) = k - i + 1$ while in the line 5 we should increase z by $1 \cdot (k - i) + (-1) \cdot (k - i + 1) = -1$. Therefore the resulting 3-VASS have the property that the shortest (and the only) run from $(1, 0, k)$ to $(0, 0, 0)$ is exponential in k .

```
1: for i := 1 to k do
2:   loop
3:     x -= 1   y += 2   z += k - i + 1
4:   loop
5:     x += 1   y -= 1   z -= 1
6: loop
7:   x -= 1
```

The use of the controlling-counter technique may be much more intricate, however the above Example 5 presents its main idea. In [4] the whole Ackermann-hardness idea was based on controlling-counters. Lasota in [12] simplified our approach and presented it without the use of controlling-counters. It turns however that in low dimensions controlling-counter technique can be very convenient as it uses only one additional dimension to control others in contrast to the multiplication triple technique (explained below), which requires three dimensions (at least in its classical version). Below we briefly describe the multiplication triple technique. We also show how to use it together with the controlling-counter technique to obtain Tower-hardness for the reachability problem in VASSes already in dimension eight.

8 Multiplication triples

Both the above presented techniques of telescope equations and controlling-counter are useful for designing VASSes with long runs, but it is not clear how they solely can be used to get some complexity lower bounds.

Here we briefly introduce the technique of multiplication triples and show how to use it to get pretty easily PSpace-hardness lower bound for the reachability problem in unary 7-VASSes. Notice that it is not hard to improve this result, in [4] we have show PSpace-hardness for unary 5-VASSes. Here we present this simple result to illustrate briefly an application of the multiplication triple technique.

Recall that a d -counter machine is just a d -VASS with possibility of zero-tests. We say that a run of a counter machine is B -bounded if at each configuration on this run the sum of all the counter values does not exceed B . We first recall the following theorem, which is a folklore.

► **Theorem 6.** *The problem whether a given three-counter machine for a given number $n \in \mathbb{N}$ has a 2^n -bounded run from a given source configuration to a given target configuration is PSpace-hard.*

The main idea behind the multiplication triple technique is that a d -VASS equipped with three additional counters (x, y, z) with initial values (B, C, BC) can simulate $C/2$ zero-tests on B -bounded counters. Here we do not explain how this simulation exactly works and why this is the case, explanations can be found in [12, 4]. It is important for us here that in order to obtain PSpace-hardness for 7-VASSes it is enough to construct a family V_n of 7-VASSes with the following properties:

- transition values of V_n are bounded by $2n$ (any polynomial function of n is fine),
- all the reachable configurations of the form $t(x_1, \dots, x_7)$, where t is the target state, have the property that if $x_7 = 0$ then $x_1 = x_2 = x_3 = 0$, $x_4 = 2^n$ and $x_6 = 2^n \cdot x_5$.

Intuitively speaking by testing x_7 for zero in the target configuration we get a triple of the form $(2^n, C, 2^n \cdot C)$ on counters (x_4, x_5, x_6) . In the latter part of the VASS run we can simulate a three-counter machine on counters (x_1, x_2, x_3) and use the counters (x_4, x_5, x_6) to check whether x_1, x_2, x_3 are indeed 2^n -bounded and for simulating zero-tests on them. Thus in the rest of this paragraph we focus on showing how to construct the above family V_n . Recall that counter programs are just ways of presenting VASSes, so we interchangeably speak about VASSes and counter programs.

► **Example 7.** The idea is simple. We only use counters x_1, x_4, x_5, x_6, x_7 . We first set $x_4 = 1$ and $x_5 = x_6 = C$ for some guessed value C . Then using x_1 as an auxiliary counter we multiply n times counters x_4 and x_6 by 2. Counter x_7 is used as the controlling-counter to assure that the multiplications are exact. During this process the counters x_4 and x_6 are zero-tested n times while the counter x_1 is zero-tested $2n$ times. Therefore in the line 1 the increase of x_4 by 1 results in the increase of x_7 by n . Similarly in line 3 the increase of x_6 by 1 results in the increase of x_7 by $2n$. In the i -th iteration of the for-loop we have that:

- in the line 6 counter x_4 is waiting for $n - i + 1$ zero-tests and counter x_1 is waiting for $2(n - i + 1)$ zero-tests, so x_7 should be increased by $3n - 3i + 3$,
- in the line 8 counter x_1 is waiting for $2(n - i + 1)$ zero-tests and counter x_4 is waiting for $n - i$ zero-tests, so x_7 should be decreased by $n - i + 2$,
- in the line 6 counter x_6 is waiting for $n - i + 1$ zero-tests and counter x_1 is waiting for $2(n - i + 1) - 1$ zero-tests, so x_7 should be increased by $3n - 3i + 1$,
- in the line 8 counter x_1 is waiting for $2(n - i + 1) - 1$ zero-tests and counter x_6 is waiting for $n - i$ zero-tests, so x_7 should be decreased by $n - i + 1$,

```

1:  $x_4 \ += \ 1 \quad x_7 \ += \ n$ 
2: loop
3:    $x_5 \ += \ 1 \quad x_6 \ += \ 1 \quad x_7 \ += \ 2n$ 
4: for  $i \ := \ 1 \quad \mathbf{to} \ n \ \mathbf{do}$ 
5:   loop
6:      $x_4 \ -= \ 1 \quad x_1 \ += \ 2 \quad x_7 \ += \ 3n - 3i + 3$ 
7:   loop
8:      $x_1 \ -= \ 1 \quad x_4 \ += \ 1 \quad x_7 \ -= \ n - i + 2$ 
9:   loop
10:     $x_6 \ -= \ 1 \quad x_1 \ += \ 2 \quad x_7 \ += \ 3n - 3i + 1$ 
11:  loop
12:     $x_1 \ -= \ 1 \quad x_6 \ += \ 1 \quad x_7 \ -= \ n - i + 1$ 

```

If after this counter program the controlling-counter x_7 has value zero then it means that indeed $x_1 = 0$, $x_4 = 2^n$, $x_6 = 2^n \cdot x_5$ and clearly $x_2 = x_3 = 0$, so all the necessary conditions for PSpace-hardness are fulfilled.

The above example shows how to join forces of controlling-counter and multiplication triples technique to rather easily show some not entirely trivial PSpace-hardness lower bound for 7-VASSes. By more clever constructions we can get a bit stronger lower bounds, but we are still very far away from matching the upper and the lower bounds for the reachability problem in low dimensional VASSes.

9 Afterthought

In this short tutorial we tried to present in the simplest possible way almost the whole spectrum of current techniques of designing involved VASSes. Many of the applications are more elaborate than the presented once, however it is still surprising that most of them are not extremely complicated and some problems open for decades are solvable by techniques which are at the end of the day rather simple. In my opinion we still need at least a few more techniques in order to understand what phenomena are hiding in the low dimensional VASSes.

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 *Proceedings of LICS 2015*, pages 32–43, 2015.
- 2 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. In *Proceedings of STOC 2019*, pages 24–33. ACM, 2019.
- 3 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. Reachability in fixed dimension vector addition systems with states. In *Proceedings of CONCUR 2020*, pages 48:1–48:21, 2020.
- 4 Wojciech Czerwinski and Lukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. In *Proceedings of FOCS 2021*, pages 1229–1240, 2021.
- 5 Wojciech Czerwinski and Lukasz Orlikowski. Lower bounds for the reachability problem in fixed dimensional vasses. *CoRR*, abs/2203.04243, 2022.
- 6 Matthias Englert, Ranko Lazic, and Patrick Totzke. Reachability in two-dimensional unary vector addition systems with states is NL-complete. In *Proceedings of LICS 2016*, pages 477–484, 2016.
- 7 Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and Primitive-Recursive Bounds with Dickson’s Lemma. In *Proceedings of LICS 2011*, pages 269–278, 2011.
- 8 Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in succinct and parametric one-counter automata. In *Proceedings of CONCUR 2009*, pages 369–383, 2009.
- 9 John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8:135–159, 1979.
- 10 S. Rao Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *Proceedings of STOC 1982*, pages 267–281, 1982.
- 11 Jean-Luc Lambert. A structure to decide reachability in Petri nets. *Theor. Comput. Sci.*, 99(1):79–104, 1992.
- 12 Slawomir Lasota. Improved Ackermannian Lower Bound for the Petri Nets Reachability Problem. In *Proceedings of STACS 2022*, volume 219 of *LIPICs*, pages 46:1–46:15, 2022.

- 13 Jérôme Leroux. The reachability problem for petri nets is not primitive recursive. In *Proceedings of FOCS 2021*, pages 1241–1252, 2021.
- 14 Jérôme Leroux. The reachability problem for petri nets is not primitive recursive. *CoRR*, abs/2104.12695, 2021.
- 15 Jérôme Leroux and Sylvain Schmitz. Demystifying reachability in vector addition systems. In *Proceedings of LICS 2015*, pages 56–67, 2015.
- 16 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *Proceedings of LICS 2019*, pages 1–13. IEEE, 2019.
- 17 Richard J. Lipton. The reachability problem requires exponential space. Technical report, Yale University, 1976.
- 18 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *Proceedings of STOC 1981*, pages 238–246, 1981.
- 19 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978.
- 20 Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016.
- 21 Richard Zurawski and MengChu Zhou. Petri nets and industrial applications: A tutorial. *IEEE Trans. Ind. Electron.*, 41(6):567–583, 1994.

On the Axiomatisation of Branching Bisimulation Congruence over CCS

Luca Aceto 

Reykjavik University, Iceland
Gran Sasso Science Institute, L'Aquila, Italy

Valentina Castiglioni 

Reykjavik University, Iceland

Anna Ingólfssdóttir 

Reykjavik University, Iceland

Bas Luttik 

Eindhoven University of Technology, The Netherlands

Abstract

In this paper we investigate the equational theory of (the restriction, relabelling, and recursion free fragment of) CCS modulo rooted branching bisimilarity, which is a classic, bisimulation-based notion of equivalence that abstracts from internal computational steps in process behaviour. Firstly, we show that CCS is not finitely based modulo the considered congruence. As a key step of independent interest in the proof of that negative result, we prove that each CCS process has a unique parallel decomposition into indecomposable processes modulo branching bisimilarity. As a second main contribution, we show that, when the set of actions is finite, rooted branching bisimilarity has a finite equational basis over CCS enriched with the left merge and communication merge operators from ACP.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting

Keywords and phrases Equational basis, Weak semantics, CCS, Parallel composition

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.6

Related Version *Technical report version with full proofs*: <http://arxiv.org/abs/2206.13927> [4]

Funding This work has been partially supported by the project “*Open Problems in the Equational Logic of Processes*” (OPEL) of the Icelandic Research Fund (grant No. 196050-051). V. Castiglioni has been supported by the project “*Programs in the wild: Uncertainties, adaptability and verification*” (ULTRON) of the Icelandic Research Fund (grant No. 228376-051).

Acknowledgements We thank the reviewers for their valuable comments that helped us to improve our contribution.

1 Introduction

This paper is a new chapter in the saga of the axiomatisation of the *parallel composition operator* \parallel (also known as “*full*” *merge* [12, 13]) of the Calculus of Communicating Systems (CCS) [27]. The saga has its roots in the works [22, 23], in which Hennessy and Milner studied the *equational theory* of (recursion free) CCS and proposed a *ground-complete axiomatisation* for it modulo *strong bisimilarity* and *observational congruence*, two classic notions of behavioural *congruence* (i.e., an equivalence relation that is compositional with respect to the language operators) that allow one to establish whether two processes have the same *observable behaviour* [34]. That axiomatisation included infinitely many axioms, which were instances of the *expansion law* used to “simulate equationally” the operational semantics of \parallel . Then, Bergstra and Klop showed, in [12], that a *finite* ground-complete axiomatisation



© Luca Aceto, Valentina Castiglioni, Anna Ingólfssdóttir, and Bas Luttik;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Sławomir Lasota, and Anca Muscholl; Article No. 6; pp. 6:1–6:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

modulo bisimilarity can be obtained by enriching CCS with two auxiliary operators, i.e., the *left merge* \parallel and the *communication merge* $|$, expressing one step in the pure interleaving and the synchronous behaviour of \parallel , respectively. Their result was strengthened by Aceto et al. in [7], where it is proved that, over the fragment of CCS without recursion, restriction and relabelling, the auxiliary operators \parallel and $|$ allow for finitely axiomatising \parallel modulo bisimilarity also when CCS terms with variables are considered. Moreover, in [9] that result is extended to the fragment of CCS with relabelling and restriction, but without communication. From those studies, we can infer that \parallel and $|$ are *sufficient* to finitely axiomatise \parallel over CCS modulo bisimilarity. (*Henceforth, we only consider the recursion, restriction and relabelling free fragment of CCS.*) Moller showed, in [30,31], that they are also *necessary*. He considered a minimal fragment of CCS, including only the inactive process, action prefixing, nondeterministic choice and interleaving, and proved that, even in the presence of a single action, bisimilarity does not afford a finite ground-complete axiomatisation over that language. Moller's proof technique was then used to show that the same negative result holds if we replace \parallel and $|$ with the so called *Hennessy's merge* [21], which denotes an asymmetric interleaving with communication, or, more generally, with a single binary auxiliary operator satisfying three assumptions given in [3].

The aforementioned works considered equational characterisations of \parallel modulo strong bisimilarity. However, a plethora of behavioural congruences have been proposed in the literature, corresponding to different levels of abstraction from the information on process execution. Hence, another chapter in the saga consisted in extending the studies recalled above to the behavioural congruences in van Glabbeek's linear time-branching time spectrum [15]. The work [5] delineated the *boundary* between finite and non-finite axiomatisability of \parallel modulo all the congruences in the spectrum.

Our contribution: branching bisimulation congruence. Some information on process behaviour can either be considered irrelevant or be unavailable to an external observer. *Weak behavioural semantics* have been introduced to study the effects of these unobservable (or *silent*) actions, usually denoted by τ , on the observable behaviour of processes, each semantics considering a different level of abstraction. A taxonomy of weak semantics is given in [17], and studies on the equational theories of various of these semantics have been carried out over the algebra BCCSP, which consists of the basic operators from CCS and CSP [24] but does not include \parallel (see, among others, [6, 14, 20, 23, 33]). A finite, ground-complete axiomatisation of parallel composition modulo *rooted weak bisimilarity* (also known as *observational congruence* [23]) is provided by Bergstra and Klop in [13] over the algebra ACP_τ that includes the auxiliary operators \parallel and $|$. To the best of our knowledge, the only study on the axiomatisability of CCS's \parallel over open terms modulo weak congruences is the negative result from [2], which shows that a class of weak congruences (including rooted weak bisimilarity) does not afford a finite, complete axiomatisation over the open terms of the minimal fragment of CCS with interleaving.

In this paper we focus on *branching bisimilarity* [19], which generalises strong bisimilarity to abstract away from τ -steps of terms while preserving their *branching structure* [19,20], and its *rooted* version, which is a congruence with respect to CCS operators.

As a first main contribution, we show that *rooted branching bisimilarity affords no finite ground-complete axiomatisation over CCS*. To this end, we adapt the proof-theoretic technique used by Moller to prove the corresponding negative result for strong bisimilarity. We remark that, even though the general proof strategy is a natural extension of Moller's, our proof requires a number of original, non-trivial technical results on (rooted) branching bisimilarity.

In particular, we observe that equational proofs of τ -free equations might involve terms having occurrences of τ in some intermediate steps (see, e.g., page 175 of Moller's thesis [30]), and our proof of the negative result for rooted branching bisimilarity will account for those uses of τ , thus making our results special for the considered weak congruence. Moreover, as an intermediate step in our proof, we establish a result of independent interest: we show that *each CCS process has a unique decomposition into indecomposable processes modulo branching bisimilarity*. A similar result was proven in [26], but only for interleaving parallel composition. Here, we extend this result to the full merge operator, including thus the possibility of communication between the parallel components.

Having established the negative result, a natural question is whether the use of the auxiliary operators from [12] can help us to obtain an equational basis for rooted branching bisimilarity. Hence, as our second main contribution, we consider the language CCS_{LC} , namely CCS enriched with \parallel and $|$, and *we provide a complete axiomatisation for rooted branching bisimilarity over CCS_{LC} that is finite when so is the set of actions over which terms are defined*. This axiomatisation is obtained by extending the complete axiom system for strong bisimilarity over CCS_{LC} from [7] with axioms expressing the behaviour of \parallel and $|$ in the presence of τ -actions (from [13]), and with the suitable τ -laws (from [20, 23]) necessary to deal with rooted branching bisimilarity. Specifically, we will see that we can express equationally the fact that left merge and communication merge distribute over choice (left merge in one argument, communication merge in both), thus allowing us to expand the behaviour of the parallel components using only a finite number of axioms, regardless of their size. A key step in the proof of the completeness result consists in another intermediate original contribution of this work: the definition of the semantics of *open* CCS_{LC} terms.

Our contribution can then be summarised as follows:

1. We show that every branching equivalence class of CCS processes has a unique parallel decomposition into indecomposables.
2. We prove that rooted branching bisimilarity admits no finite equational axiomatisation over CCS.
3. We define the semantics of open CCS_{LC} terms.
4. We provide a (finite) complete axiomatisation for \sim_{REB} over CCS_{LC} .

2 Background

Labelled transition systems. As semantic model we consider classic *labelled transition systems* [25]. We assume a non-empty set of action names \mathcal{A} , and we let $\bar{\mathcal{A}}$ denote the set of action co-names, i.e., $\bar{\mathcal{A}} = \{\bar{a} \mid a \in \mathcal{A}\}$. As usual, we postulate that $\bar{\bar{a}} = a$ and $a \neq \bar{a}$ for all $a \in \mathcal{A}$. Then, we define $\mathcal{A}_\tau = \mathcal{A} \cup \bar{\mathcal{A}} \cup \{\tau\}$, where $\tau \notin \mathcal{A} \cup \bar{\mathcal{A}}$. Henceforth, we let μ, ν, \dots range over actions in \mathcal{A}_τ , and α, β, \dots range over actions in $\mathcal{A} \cup \bar{\mathcal{A}}$.

► **Definition 1** (Labelled Transition System). *A labelled transition system (LTS) is a triple $(\mathbf{P}, \mathcal{A}_\tau, \rightarrow)$, where \mathbf{P} is a set of processes (or states), \mathcal{A}_τ is a set of actions, and $\rightarrow \subseteq \mathbf{P} \times \mathcal{A}_\tau \times \mathbf{P}$ is a (labelled) transition relation.*

As usual, we use $p \xrightarrow{\mu} p'$ in lieu of $(p, \mu, p') \in \rightarrow$. For each $p \in \mathbf{P}$ and $\mu \in \mathcal{A}$, we write $p \xrightarrow{\mu}$ if $p \xrightarrow{\mu} p'$ holds for some p' , and $p \not\xrightarrow{\mu}$ otherwise. The *initials* of p are the actions that label the outgoing transitions of p , that is, $\text{init}(p) = \{\mu \in \mathcal{A}_\tau \mid p \xrightarrow{\mu}\}$.

■ **Table 1** The SOS rules for CCS operators ($\mu \in \mathcal{A}_\tau$, $\alpha \in \mathcal{A} \cup \bar{\mathcal{A}}$).

$$\frac{}{\mu.t \xrightarrow{\mu} t} \quad \frac{t \xrightarrow{\mu} t'}{t + u \xrightarrow{\mu} t'} \quad \frac{t \xrightarrow{\mu} t'}{t \parallel u \xrightarrow{\mu} t' \parallel u} \quad \frac{t \xrightarrow{\alpha} t' \quad u \xrightarrow{\bar{\alpha}} u'}{t \parallel u \xrightarrow{\tau} t' \parallel u'}$$

The language CCS. We consider the recursion, relabelling and restriction free fragment of Milner's CCS [28], which for simplicity we still call CCS, given by the following grammar:

$$t ::= \mathbf{0} \mid x \mid \mu.t \mid t + t \mid t \parallel t ,$$

where x is a variable drawn from a countably infinite set \mathcal{V} disjoint from \mathcal{A}_τ , and $\mu \in \mathcal{A}_\tau$. We use the *Structural Operational Semantics* (SOS) framework [35,36] to equip processes with an operational semantics. The SOS rules (or inference rules) for the CCS operators given above are reported in Table 1 (symmetric rules for $+$ and \parallel are omitted).

We shall use the meta-variables t, u, v, w to range over process terms, and write $\text{var}(t)$ for the collection of variables occurring in the term t . We use a *summation* $\sum_{i \in \{1, \dots, k\}} t_i$ to abbreviate $t_1 + \dots + t_k$, where the empty sum represents $\mathbf{0}$. We call the term t_j ($j \in \{1, \dots, k\}$) a *summand* of $t = \sum_{i \in \{1, \dots, k\}} t_i$ if it does not have $+$ as head operator. The *size* of a term t , denoted by $\text{size}(t)$, is the number of operator symbols in t . A term is *closed* if it does not contain any variables. Closed terms, or *processes*, will be denoted by p, q, r . Moreover, we omit trailing $\mathbf{0}$'s from terms. A (*closed*) *substitution* is a mapping from process variables to (closed) terms. Substitutions are extended from variables to terms, transitions, and rules in the usual way. Note that $\sigma(t)$ is closed, if so is σ . We let $\sigma[x \mapsto p]$ denote the substitution that maps the variable x into process p and behaves like σ on all other variables. In particular, $[x \mapsto p]$ denotes the substitution that maps the variable x into process p and behaves like the identity on all other variables.

The inference rules in Table 1 allow us to derive valid transitions between closed terms. The operational semantics for our language is then modelled by the LTS whose processes are the closed terms, and whose labelled transitions are those that are provable from the SOS rules. Henceforth, we let \mathbf{P} denote the set of CCS processes. We remark that whenever $p \xrightarrow{\mu} p'$, then $\text{size}(p) > \text{size}(p')$.

Branching bisimilarity. *Branching bisimilarity* is a bisimulation-based behavioural equivalence that abstracts away from computation steps in processes that are deemed unobservable, while preserving their *branching structure*. The abstraction is achieved by labelling these computation steps with τ , and giving τ -labelled transitions a special treatment in the definition of the behavioural equivalence. Preservation of the branching structure is mainly due to the *stuttering* nature of branching bisimulation, which guarantees that the behaviour of a term is preserved in the execution of a sequence of silent steps [19,20].

Let $\xrightarrow{\varepsilon}$ denote the reflexive and transitive closure of the transition $\xrightarrow{\tau}$.

► **Definition 2** (Branching bisimilarity). *Let $(\mathbf{P}, \mathcal{A}_\tau, \rightarrow)$ be a LTS. Branching bisimilarity, denoted by \sim_{BB} , is the largest symmetric relation over \mathbf{P} such that, whenever $p \sim_{\text{BB}} q$, if $p \xrightarrow{\mu} p'$, then either:*

- $\mu = \tau$ and $p' \sim_{\text{BB}} q$, or
- there are processes q', q'' such that $q \xrightarrow{\varepsilon} q'' \xrightarrow{\mu} q'$, $p \sim_{\text{BB}} q''$, and $p' \sim_{\text{BB}} q'$.

■ **Table 2** Some axioms for rooted branching bisimilarity.

Some axioms for bisimilarity over CCS:

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	$(x + y) + z \approx x + (y + z)$	P2	$(x \parallel y) \parallel z \approx x \parallel (y \parallel z)$
A3	$x + x \approx x$		

Additional axioms for rooted branching bisimilarity over CCS:

TB	$\mu(\tau(x + y) + y) \approx \mu(x + y)$	T1	$\mu\tau x \approx \mu x$
----	---	----	---------------------------

Branching bisimilarity satisfies the *stuttering property* [20, Lemma 2.5]: *Assume that $p \sim_{\text{BB}} q$. Whenever $p \xrightarrow{\tau} p_1 \xrightarrow{\tau} \dots \xrightarrow{\tau} p_n$ and $p_n \sim_{\text{BB}} q$, for some $n \geq 1$, then $p_i \sim_{\text{BB}} q$ for all $i = 1, \dots, n - 1$.*

To guarantee compositional reasoning over a process language, we require a behavioural equivalence \sim to be a *congruence* with respect to all language operators. This consists in verifying whether, for all n -ary operators f

if $t_i \sim t'_i$ for all $i = 1, \dots, n$, then $f(t_1, \dots, t_n) \sim f(t'_1, \dots, t'_n)$.

It is well known that branching bisimilarity is an equivalence relation [11, 20]. Moreover, action prefixing and parallel composition satisfy the *simple BB cool rule format* [18] and hence \sim_{BB} is compositional with respect to those operators. However, \sim_{BB} is not a congruence with respect to nondeterministic choice. To remedy this inconvenience, the *root condition* is introduced: *rooted branching bisimilarity* behaves like strong bisimilarity on the initial transitions, and like branching bisimilarity on subsequent transitions.

► **Definition 3** (Rooted branching bisimilarity). *Rooted branching bisimilarity, denoted by \sim_{RBB} , is the symmetric relation over \mathbf{P} such that, whenever $p \sim_{\text{RBB}} q$, if $p \xrightarrow{\mu} p'$, then there is a process q' such that $q \xrightarrow{\mu} q'$ and $p' \sim_{\text{BB}} q'$.*

It is well known that rooted branching bisimilarity is an equivalence relation [11, 20], and that \sim_{RBB} is a congruence over CCS (see, e.g., [18]).

Equational Logic. An *axiom system* \mathcal{E} is a collection of (*process*) *equations* $t \approx u$ over the considered language, thus CCS in this paper. 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*.

We assume, without loss of generality, that the substitution rule is only applied on equations $(t \approx u) \in \mathcal{E}$. In this case, $\sigma(t) \approx \sigma(u)$ is called a *substitution instance* of an axiom in \mathcal{E} . Moreover, by postulating that for each axiom in \mathcal{E} also its symmetric counterpart is present in \mathcal{E} , one may assume that the symmetry rule is never used in equational proofs.

We are interested in equations that are valid modulo some congruence relation \sim over terms. The equation $t \approx u$ is said to be *sound* modulo \sim if $\sigma(t) \sim \sigma(u)$ for all closed substitutions σ . For simplicity, if $t \approx u$ is sound, then we write $t \sim u$. An axiom system is *sound* modulo \sim if, and only if, all of its equations are sound modulo \sim . Conversely, we say that \mathcal{E} is *complete* modulo \sim if $t \sim u$ implies $\mathcal{E} \vdash t \approx u$ for all terms t, u . If we restrict ourselves to consider only equations over closed terms then \mathcal{E} is said to be *ground-complete* modulo \sim . We say that \sim has a finite, (ground) complete axiomatisation, if there is a finite axiom system \mathcal{E} that is sound and (ground) complete for \sim .

Henceforth, we exploit the associativity and commutativity of $+$ and \parallel modulo the relevant behavioural equivalences. The symbol $=$ will then denote equality modulo A1–A2 and P1–P2 in Table 2.

3 The main results

Our aim is to study the axiomatisability of rooted branching bisimilarity over CCS. Our investigations produced, as main outcomes, a negative result (Theorem 4) and a positive one (Theorem 5). In detail, in the first part of the paper we prove the following theorem:

► **Theorem 4.** *Rooted branching bisimilarity has no finite equational ground-complete axiomatisation over CCS.*

Given the negative result, it is natural to wonder whether an equational basis for rooted branching bisimilarity can be obtained if we enrich CCS with some auxiliary operators. Considering the similarities between \sim_{RBB} and strong bisimilarity, the principal candidates for this role are the left merge \ll and the communication merge $|$ from [12]. Indeed, we show that if we add those two operators to the syntax of CCS, then we can obtain a complete axiomatisation of rooted branching bisimilarity over the new language, denoted by CCS_{LC} . The desired equational basis is given by the axiom system \mathcal{E}_{RBB} , which is presented fully in Table 7 in Section 10. \mathcal{E}_{RBB} is an extension of the complete axiom system for strong bisimilarity over CCS_{LC} from [7] with axioms expressing the behaviour of left merge and communication merge in the presence of τ -actions (taken from [13]), and with the suitable τ -laws necessary to deal with rooted branching bisimilarity (taken from [20, 23]).

Formally, our second main contribution consists in a proof of the following theorem:

► **Theorem 5 (Completeness).** *Let t, u be CCS_{LC} terms. If $t \sim_{\text{RBB}} u$, then $\mathcal{E}_{\text{RBB}} \vdash t \approx u$.*

We will also argue that this axiomatisation is finite when so is the set of actions. Hence, when \mathcal{A} is finite, CCS_{LC} modulo \sim_{RBB} is finitely based, unlike CCS.

Considering the amount of technical results that we will need to fulfil our objectives, we devote Section 4 to a presentation of the proof strategy that we will apply to obtain Theorem 4. Sections 5–7 then present the formalisation of the ideas discussed in that section. Similarly, in Section 8 we give a high-level description of the approach that we will follow to prove Theorem 5. The technical development of the proof is then reported in Sections 9–10.

4 Proof strategy for Theorem 4

In this section we present the proof strategy we will apply to obtain Theorem 4.

Our proof follows the so-called *proof-theoretic approach* to non-finite-axiomatisability results, whose use in the field of process algebra stems from [30–32], where Moller proved that CCS modulo strong bisimilarity is not finitely based. In the proof-theoretic approach, the idea is to identify a specific property of terms parametric in $n \geq 0$, say \mathbb{P}_n , and show that if \mathcal{E} is an arbitrary finite axiom system that is sound with respect to \sim_{RBB} , then \mathbb{P}_n is preserved by provability from \mathcal{E} when n is “large enough”. Next, we exhibit an infinite family of equations $\{\epsilon_n \mid n \geq 0\}$ over closed terms that are all sound modulo \sim_{RBB} , but are such that only one side of ϵ_n satisfies \mathbb{P}_n , for each $n \geq 0$. In particular, this implies that whenever n is “large enough” then the sound equation ϵ_n cannot be proved from \mathcal{E} . Since \mathcal{E} is an arbitrary finite sound axiom system, it follows that no finite sound axiomatisation can prove all the equations in the family $\{\epsilon_n \mid n \geq 0\}$ and therefore no finite sound axiomatisation is ground complete for CCS modulo \sim_{RBB} .

The choice of \mathbb{P}_n and the family of equations. In [30–32] Moller applied the proof method sketched above to prove that strong bisimilarity has no finite, complete axiomatisation over CCS. The key idea underlying this result is that, since \parallel does not distribute over $+$ in either of its arguments modulo strong bisimilarity, then no finite, sound axiom system can “*expand*” the initial behaviour of process $a \parallel \sum_{i=1}^n a^i$ (where $a^i = aa^{i-1}$ for each $i = 1, \dots, n$, with $a^0 = \mathbf{0}$) when n is large.

Since, by definition, rooted branching bisimilarity behaves exactly like strong bisimilarity on the first step, and parallel composition does not distribute over choice in either of its arguments, modulo \sim_{RBB} , it is natural to exploit a similar strategy to prove Theorem 4. In detail, we will consider, for each $n \geq 2$, the process $p_n = \sum_{i=2}^n aa^{\leq i}$, where $a^{\leq i} = \sum_{j=1}^i a^j$ for each $i = 2, \dots, n$. Then, for each $n \geq 2$, the property \mathbb{P}_n will consist in having a summand rooted branching bisimilar to the process $a \parallel p_n$, and we will show that, when n is large enough, \mathbb{P}_n is an invariant under provability from an arbitrary finite, sound axiom system (Theorem 18). Hence, the sound equation $\epsilon_n : a \parallel p_n \approx ap_n + \sum_{i=2}^n a(a \parallel a^{\leq i})$ cannot be derived from \mathcal{E} because its right-hand side has no summand that is rooted branching bisimilar to $a \parallel p_n$, unlike its left-hand side. Therefore no finite sound axiom system can prove the infinite family of equations $\{\epsilon_n \mid n \geq 2\}$, yielding the desired negative result.

In proving that \mathbb{P}_n is invariant under provability, one pivotal ingredient will be the fact that processes p_n and $a^{\leq i}$, for $n \geq 2$ and $i \in \{2, \dots, n\}$, are *indecomposable*. The existence of a unique parallel decomposition into indecomposable processes modulo *branching bisimilarity* over CCS with *interleaving parallel composition* was studied in [26]. In Section 6, we extend the result from [26] to the full merge operator, thus including communication (Proposition 16).

The choice of n . The choice of a sufficiently large n plays a crucial role in proving that \mathbb{P}_n is an invariant under provability from a finite, sound axiom system \mathcal{E} (Theorem 18). The key step in that proof deals with the case in which $p \approx q$ is a substitution instance of an equation in \mathcal{E} (Proposition 20), i.e., $p = \sigma(t)$, $q = \sigma(u)$, and $t \approx u \in \mathcal{E}$ for some terms t, u and closed substitution σ . In this case, assuming that $n > \text{size}(t)$, we can prove that if $p = \sigma(t)$ satisfies \mathbb{P}_n then this is due to the behaviour of $\sigma(x)$ for some variable x . In order to reach this conclusion, in Section 5, we study how the behaviour of closed instances of terms may depend on the behaviour of the closed instances of variables occurring in them. Moreover, we one can show that if $t \approx u$ is sound modulo rooted branching bisimilarity and x occurs in t , then it occurs also in u . Hence, we can infer that $\sigma(x)$ triggers in $\sigma(u)$ the same behaviour that it induced in $\sigma(t)$, and thus that $q = \sigma(u)$ satisfies \mathbb{P}_n .

5 Decomposing the semantics of terms

In the proofs to follow, we shall sometimes need to establish a correspondence between the behaviour of open terms and that of their closed instances. In detail, we are interested in the correspondence between a transition $\sigma(t) \xrightarrow{\mu} p$, for some term t , closed substitution σ , action μ , and process p , and the behaviour of t and that of $\sigma(x)$, for each variable x occurring in t . The simplest case is a direct application of the operational semantics in Table 1.

► **Lemma 6.** *For all terms t, t' , substitution σ , and $\mu \in \mathcal{A}_\tau$, if $t \xrightarrow{\mu} t'$ then $\sigma(t) \xrightarrow{\mu} \sigma(t')$.*

Let us focus now on the role of variables. A transition $\sigma(t) \xrightarrow{\mu} p$ may also derive from the initial behaviour of some closed term $\sigma(x)$, provided that the collection of initial moves of $\sigma(t)$ depends, in some formal sense, on that of the closed term substituted for the variable

■ **Table 3** Inference rules for the transition relation $\xrightarrow{\ell}_\rho$ ($\mu \in \mathcal{A}_\tau$, $\alpha \in \mathcal{A} \cup \overline{\mathcal{A}}$).

$$\begin{array}{c}
(a_1) \frac{}{x \xrightarrow{(x)}_\mu x_\mu} \quad (a_2) \frac{t \xrightarrow{\ell}_\rho c}{t + u \xrightarrow{\ell}_\rho c} \quad (a_3) \frac{t \xrightarrow{\ell}_\rho c}{t \parallel u \xrightarrow{\ell}_\rho c \parallel u} \\
(a_4) \frac{t \xrightarrow{(x)}_\alpha c \quad u \xrightarrow{(y)}_{\overline{\alpha}} c'}{t \parallel u \xrightarrow{(x,y)}_\tau c \parallel c'} \quad (a_5) \frac{t \xrightarrow{(x)}_\alpha c \quad u \xrightarrow{\overline{\alpha}} u'}{t \parallel u \xrightarrow{(x)}_{\alpha,\tau} c \parallel u'} \quad (a_6) \frac{t \xrightarrow{\alpha} t' \quad u \xrightarrow{(x)}_{\overline{\alpha}} c}{t \parallel u \xrightarrow{(x)}_{\overline{\alpha},\tau} t' \parallel c}
\end{array}$$

x . In this case, we say that x *triggers the behaviour* of t . To fully describe this situation, we introduce an auxiliary transition relation over open terms. The notion of *configuration* over terms, which stems from [8], will play an important role in their definition.

The presence of communication in CCS entails a complex definition of the semantics of configurations. In particular, it is necessary to introduce a fresh set of variables $\mathcal{V}_{\mathcal{A}_\tau} = \{x_\mu \mid x \in \mathcal{V}, \mu \in \mathcal{A}_\tau\}$, disjoint from \mathcal{V} , and terms. Intuitively, the symbol x_μ denotes that the closed term substituted for an occurrence of variable x has begun its execution (expressed in terms of a μ -action), and it contributes thus to triggering the behaviour of the term in which x occurs (see Example 8 below). Moreover, we also need to introduce special labels and subscripts for the auxiliary transitions over configurations, which will be of the form $c \xrightarrow{\ell}_\rho c'$. Briefly, the label ℓ is used to keep track of the variables that trigger the transition $c \xrightarrow{\ell}_\rho c'$. The subscript ρ , instead, will allow us to correctly define the semantics of communication: it will allow us to distinguish a τ -action directly performed by (the term substituted for) a variable x (transition $c \xrightarrow{(x)}_\tau c'$, with $\rho = \tau$), from a τ -action resulting from the communication of x with a subterm of the configuration (transition $c \xrightarrow{(x)}_{\alpha,\tau} c'$, with $\rho = \alpha, \tau$, where α is the action performed by the term substituted for x).

CCS configurations are defined over the set of variables $\mathcal{V}_{\mathcal{A}_\tau}$ and CCS terms.

► **Definition 7.** *The collection of CCS configurations, denoted by \mathcal{C} , is given by:*

$$c ::= x_\mu \mid t \mid c \parallel c, \quad \text{where } t \text{ is a term, and } x_\mu \in \mathcal{V}_{\mathcal{A}_\tau}.$$

The auxiliary transitions of the form $\xrightarrow{\ell}_\rho$ are then formally defined via the inference rules in Table 3, where we omitted the symmetric rules to (a_2) , (a_4) , (a_5) and (a_6) . We have that $\rho \in \mathcal{A}_\tau \cup ((\mathcal{A} \cup \overline{\mathcal{A}}) \times \{\tau\})$, whereas the label ℓ can be either of the form (x) or (x, y) , for some variables $x, y \in \mathcal{V}$. Given a variable x and a label ℓ , we write $x \in \ell$ if x occurs in ℓ .

The distinguished variables x_μ allow us to keep track of which variable and action trigger the behaviour of the term, and they also allow us to present substitutions in an intuitive fashion. As explained in the following example, it is precisely because of substitutions (and communication) that we need to make the action μ explicit in x_μ .

► **Example 8.** Let $x \in \mathcal{V}$ and consider the term $x \parallel x$. By rules (a_1) and (a_4) in Table 3, we have that $x \parallel x \xrightarrow{(x,x)}_\tau x_\alpha \parallel x_{\overline{\alpha}}$ because $x \xrightarrow{(x)}_\alpha x_\alpha$ and $x \xrightarrow{(x)}_{\overline{\alpha}} x_{\overline{\alpha}}$. Hence, given any substitution σ such that $\sigma(x) \xrightarrow{\alpha} p_1$ and $\sigma(x) \xrightarrow{\overline{\alpha}} p_2$, for some terms p_1, p_2 , we want to be able to correctly infer that $\sigma(x) \parallel \sigma(x) \xrightarrow{\tau} p_1 \parallel p_2$. Since the two occurrences of x , x_α and $x_{\overline{\alpha}}$, can be distinguished by the subscripts, the substitution $\sigma[x_\alpha \mapsto p_1, x_{\overline{\alpha}} \mapsto p_2](x_\alpha \parallel x_{\overline{\alpha}}) = p_1 \parallel p_2$ is well-defined. Without the subscripts, it would not have been possible to correctly define the substitution σ on the configuration c that is the target of $x \parallel x \xrightarrow{(x,x)}_\tau c$.

► **Lemma 9.** *Let t be term and σ be a closed substitution. Let $x, y \in \mathcal{V}$.*

1. *For any $\mu \in \mathcal{A}_\tau$, if $\sigma(x) \xrightarrow{\mu} p$, for some process p , and $t \xrightarrow{(x)}_\mu c$, for some configuration $c \in \mathcal{C}$, then $\sigma(t) \xrightarrow{\mu} \sigma[x_\mu \mapsto p](c)$.*
2. *For any $\alpha \in \mathcal{A} \cup \bar{\mathcal{A}}$, if $\sigma(x) \xrightarrow{\alpha} p$, for some process p , and $t \xrightarrow{(x)}_{\alpha, \tau} c$, for some configuration $c \in \mathcal{C}$, then $\sigma(t) \xrightarrow{\tau} \sigma[x_\alpha \mapsto p](c)$.*
3. *For any $\alpha \in \mathcal{A} \cup \bar{\mathcal{A}}$, if $\sigma(x) \xrightarrow{\alpha} p_x$, $\sigma(y) \xrightarrow{\bar{\alpha}} p_y$, for some processes p_x, p_y , and $t \xrightarrow{(x, y)}_\tau c$, for some configuration c , then $\sigma(t) \xrightarrow{\tau} \sigma[x_\alpha \mapsto p_x, y_{\bar{\alpha}} \mapsto p_y](c)$.*

Lemma 9 shows how the auxiliary transitions can be used to derive the behaviour of $\sigma(t)$ from those of the variables in t . We are now interested in analysing the converse situation: we show how a transition $\sigma(t) \xrightarrow{\mu} p$ can stem from transitions of the term t and of the process $\sigma(x)$, for $x \in \text{var}(t)$. We limit ourselves to present the case of silent actions $\sigma(t) \xrightarrow{\tau} p$ as it requires a detailed analysis. The case of transitions labelled with observable actions is simpler and therefore omitted.

► **Lemma 10.** *Let t be a term, σ be a closed substitution, and p be a process. If $\sigma(t) \xrightarrow{\tau} p$, then one of the following holds:*

1. *There is a term t' s.t. $t \xrightarrow{\tau} t'$ and $\sigma(t') = p$.*
2. *There are a variable x , a process q , and a configuration c s.t. $\sigma(x) \xrightarrow{\tau} q$, $t \xrightarrow{(x)}_\tau c$, and $\sigma[x_\tau \mapsto q](c) = p$.*
3. *There are a variable x , a process q , and a configuration c s.t., for some $\alpha \in \mathcal{A} \cup \bar{\mathcal{A}}$, $\sigma(x) \xrightarrow{\alpha} q$, $t \xrightarrow{(x)}_{\alpha, \tau} c$, and $\sigma[x_\alpha \mapsto q](c) = p$.*
4. *There are variables x, y , processes q_x, q_y and a configuration c s.t., for some $\alpha \in \mathcal{A} \cup \bar{\mathcal{A}}$, $\sigma(x) \xrightarrow{\alpha} q_x$, $\sigma(y) \xrightarrow{\bar{\alpha}} q_y$, $t \xrightarrow{(x, y)}_\tau c$, and $\sigma[x_\alpha \mapsto q_x, y_{\bar{\alpha}} \mapsto q_y](c) = p$.*

6 Unique parallel decomposition

As explained in Section 4, our approach for establishing that \mathbb{P}_n is invariant under equational proofs relies on processes having a unique parallel decomposition modulo \sim_{BB} .

► **Definition 11** (Parallel decomposition modulo \sim_{BB}). *A process p is indecomposable if $p \not\sim_{\text{BB}} \mathbf{0}$ and $p \sim_{\text{BB}} p_1 \parallel p_2$ implies $p_1 \sim_{\text{BB}} \mathbf{0}$ or $p_2 \sim_{\text{BB}} \mathbf{0}$, for all processes p_1 and p_2 . A parallel decomposition of a process p is a finite multiset $\{p_1, \dots, p_k\}$ of indecomposable processes p_1, \dots, p_k such that $p \sim_{\text{BB}} p_1 \parallel \dots \parallel p_k$. We say that p has a unique parallel decomposition if p has a parallel decomposition $\{p_1, \dots, p_k\}$ and for every other parallel decomposition $\{p'_1, \dots, p'_\ell\}$ of p there exists a bijection $f : \{1, \dots, k\} \rightarrow \{1, \dots, \ell\}$ such that $p_i \sim_{\text{BB}} p'_{f(i)}$ for all $1 \leq i \leq k$.*

To prove that processes have a unique parallel decomposition we shall exploit a general result stating that a partial commutative monoid has unique decomposition if it can be endowed with a *weak decomposition order* that satisfies *power cancellation* [26]; we shall define and explain the notions below. Note that, in view of axioms P0–P2, which are (also) sound modulo \sim_{BB} , the set of processes \mathbf{P} modulo \sim_{BB} is a commutative monoid with respect to the binary operation naturally induced by \parallel on \sim_{BB} -equivalence classes and the \sim_{BB} -equivalence class of $\mathbf{0}$ as identity element. We permit ourselves a minor abuse in notation and use \rightarrow to (also) denote the binary relation $\{(p, q) \mid \exists \mu. p \xrightarrow{\mu} q\}$, and proceed to argue that \rightarrow induces a weak decomposition order satisfying power cancellation on the commutative monoid of processes modulo \sim_{BB} .

6:10 On the Axiomatisation of Branching Bisimulation Congruence over CCS

Given any process p and $n \geq 1$, let p^n denote the n -fold parallel composition $p \parallel p^{n-1}$, with $p^0 = \mathbf{0}$. We first state some properties of the reflexive-transitive closure \rightarrow^* of \rightarrow :

► **Proposition 12.** *The relation \rightarrow^* is an inversely well-founded partial order on processes satisfying the following properties:*

1. *For every process p there exists a process p' such that $p \rightarrow^* p' \sim_{\text{BB}} \mathbf{0}$.*
2. *For all processes p, p' and q , if $p \rightarrow^* p'$, then $p \parallel q \rightarrow^* p' \parallel q$ and $q \parallel p \rightarrow^* q \parallel p'$.*
3. *For all processes p, q and r , if $p \parallel q \rightarrow^* r$, then there exist p' and q' such that $p \rightarrow^* p'$, $q \rightarrow^* q'$ and $r = p' \parallel q'$.*
4. *For all processes p and q , if $p \rightarrow^* q^n$ for all $n \in \mathbb{N}$, then $q \sim_{\text{BB}} \mathbf{0}$.*

The following lemma is a direct consequence of the definition of branching bisimilarity.

► **Lemma 13.** *For all processes p, p' and q , if $p \sim_{\text{BB}} q$ and $p \rightarrow^* p'$, then there exists q' such that $q \rightarrow^* q'$ and $p' \sim_{\text{BB}} q'$.*

By this lemma we can define a binary relation \preceq on $\mathbf{P}/\sim_{\text{BB}}$, the set of \sim_{BB} -equivalence classes of processes, by stating that $[p]_{\sim_{\text{BB}}} \preceq [q]_{\sim_{\text{BB}}}$ if, and only if, there exists $p' \in [p]_{\sim_{\text{BB}}}$ such that $q \rightarrow^* p'$ (here $[p]_{\sim_{\text{BB}}}$ and $[q]_{\sim_{\text{BB}}}$ denote the \sim_{BB} -equivalence classes of p and q , respectively). The following result is then a straightforward corollary of Proposition 12.

► **Corollary 14.** *The relation \preceq is a weak decomposition order on $\mathbf{P}/\sim_{\text{BB}}$, namely:*

1. *it is well-founded, i.e., every non-empty subset of $\mathbf{P}/\sim_{\text{BB}}$ has a \preceq -minimal element;*
2. *the identity element $[\mathbf{0}]_{\sim_{\text{BB}}}$ of $\mathbf{P}/\sim_{\text{BB}}$ is the least element of $\mathbf{P}/\sim_{\text{BB}}$ with respect to \preceq , i.e., $[\mathbf{0}]_{\sim_{\text{BB}}} \preceq [p]_{\sim_{\text{BB}}}$ for all $p \in \mathbf{P}$;*
3. *it is compatible, i.e., for all $p, q, r \in \mathbf{P}$ if $[p]_{\sim_{\text{BB}}} \preceq [q]_{\sim_{\text{BB}}}$, then $[p \parallel r]_{\sim_{\text{BB}}} \preceq [q \parallel r]_{\sim_{\text{BB}}}$;*
4. *it is precompositional, i.e., for all $p, q, r \in \mathbf{P}$ we have that $[p]_{\sim_{\text{BB}}} \preceq [q \parallel r]_{\sim_{\text{BB}}}$ implies $[p]_{\sim_{\text{BB}}} = [q' \parallel r']_{\sim_{\text{BB}}}$ for some $[q']_{\sim_{\text{BB}}} \preceq [q]_{\sim_{\text{BB}}}$ and $[r']_{\sim_{\text{BB}}} \preceq [r]_{\sim_{\text{BB}}}$; and*
5. *it is Archimedean, i.e., for all $p, q \in \mathbf{P}$ we have that $[p^n]_{\sim_{\text{BB}}} \preceq [q]_{\sim_{\text{BB}}}$ for all $n \in \mathbb{N}$ implies that $[p]_{\sim_{\text{BB}}} = [\mathbf{0}]_{\sim_{\text{BB}}}$.*

According to [26, Theorem 34] it now remains to prove that \preceq satisfies power cancellation. The weak decomposition order \preceq on the commutative monoid of processes modulo \sim_{BB} satisfies *power cancellation* if for every indecomposable process p and for all processes q and r such that $[p]_{\sim_{\text{BB}}} \not\preceq [q]_{\sim_{\text{BB}}}, [r]_{\sim_{\text{BB}}}$, for all $k \in \mathbb{N}$, we have that $[p^k \parallel q]_{\sim_{\text{BB}}} = [p^k \parallel r]_{\sim_{\text{BB}}}$ implies $[q]_{\sim_{\text{BB}}} = [r]_{\sim_{\text{BB}}}$.

► **Proposition 15.** *The weak decomposition order \preceq on the commutative monoid of processes modulo \sim_{BB} satisfies power cancellation.*

We have now established that \preceq is a weak decomposition order on the commutative monoid of processes modulo \sim_{BB} that satisfies power cancellation. Thus, with an application of [26, Theorem 34] we get the following unique parallel decomposition result.

► **Proposition 16.** *Every process in \mathbf{P} has a unique parallel decomposition.*

In what follows, we shall make use of the following direct consequence of Proposition 16.

► **Corollary 17.** *If $p \parallel r \sim_{\text{BB}} q \parallel r$, then $p \sim_{\text{BB}} q$.*

7 Nonexistence of a finite axiomatisation

We devote this section to proving Theorem 4. Following the strategy sketched in Section 4, we introduce a particular family of equations on which we will build our negative result:

$$p_n = \sum_{i=2}^n aa^{\leq i} \quad (n \geq 2)$$

$$\epsilon_n: a \parallel p_n \approx ap_n + \sum_{i=2}^n a(a \parallel a^{\leq i}) \quad (n \geq 2).$$

It is easy to check that each equation ϵ_n , for $n \geq 2$, is sound modulo rooted branching bisimilarity (as, in particular, it is sound modulo strong bisimilarity).

In order to prove Theorem 4, we proceed to show that no finite collection of equations over CCS that are sound modulo rooted branching bisimilarity can prove all of the equations ϵ_n ($n \geq 2$) from the family given above. Formally, for each $n \geq 2$, we consider the property \mathbb{P}_n : *having a summand rooted branching bisimilar to $a \parallel p_n$* . Then, we prove the following:

► **Theorem 18.** *Let \mathcal{E} be a finite axiom system over CCS that is sound modulo \sim_{RBB} , let n be larger than the size of each term in the equations in \mathcal{E} , and let p, q be closed terms such that $p, q \sim_{\text{RBB}} a \parallel p_n$. If $\mathcal{E} \vdash p \approx q$ and p satisfies \mathbb{P}_n then so does q .*

The crucial step in the proof of Theorem 18 is delivered by the proposition below, which ensures that the property \mathbb{P}_n ($n \geq 2$) is preserved by the closure under substitutions of equations in a finite, sound axiom system. Proposition 20 is proved by means of the technical results provided so far, and the notion of **0**-factor of a term:

► **Definition 19.** *We say that a term t has a **0** factor if it contains a subterm of the form $t' \parallel t''$, and either $t' \sim_{\text{RBB}} \mathbf{0}$ or $t'' \sim_{\text{RBB}} \mathbf{0}$.*

► **Proposition 20.** *Let $t \approx u$ be an equation over CCS terms that is sound modulo \sim_{RBB} . Let σ be a closed substitution with $p = \sigma(t)$ and $q = \sigma(u)$. Suppose that p and q have neither **0** summands nor **0** factors, and $p, q \sim_{\text{RBB}} a \parallel p_n$ for some n larger than the sizes of t and u . If p satisfies \mathbb{P}_n , then so does q .*

Theorem 18 shows the property \mathbb{P}_n to be an invariant under provability from finite sound axiom systems. As the left-hand side of equation ϵ_n , i.e., the term $a \parallel p_n$, satisfies \mathbb{P}_n , whilst the right-hand side, i.e., the term $ap_n + \sum_{i=2}^n a(a \parallel a^{\leq i})$, does not, we can conclude that the infinite collection of equations ϵ_n ($n \geq 2$) cannot be derived from any finite, sound axiom system. Hence, Theorem 4 follows.

8 Towards a positive result

We now proceed to study the role of the auxiliary operators *left merge* (\mathbb{L}) and *communication merge* (\mathbb{I}) from [12] in the axiomatisation of parallel composition modulo \sim_{RBB} . We will show that by adding them to CCS we can obtain a complete axiomatisation of rooted branching bisimilarity over the new language. This axiomatisation is finite if so is \mathcal{A}_τ .

We denote the language obtained by enriching CCS with \mathbb{L} and \mathbb{I} by CCS_{LC} :

$$t ::= \mathbf{0} \mid x \mid \mu.t \mid t + t \mid t \parallel t \mid t \mathbb{L} t \mid t \mathbb{I} t, \quad (\text{CCS}_{\text{LC}})$$

where $x \in \mathcal{V}$, and $\mu \in \mathcal{A}_\tau$. The SOS rules for the CCS_{LC} operators are given by the rules in Table 1 plus those reported in Table 4.

■ **Table 4** Additional SOS rules for CCS_{LC} operators ($\mu \in \mathcal{A}_\tau$, $\alpha \in \mathcal{A} \cup \bar{\mathcal{A}}$).

$$\frac{t \xrightarrow{\mu} t'}{t \parallel u \xrightarrow{\mu} t' \parallel u} \quad \frac{t \xrightarrow{\alpha} t' \quad u \xrightarrow{\bar{\alpha}} u'}{t \mid u \xrightarrow{\tau} t' \parallel u'}$$

To obtain the desired completeness result, we consider the axiom system \mathcal{E}_{RBB} (see Table 7 in Section 10), obtained by extending the complete axiom system for strong bisimilarity over CCS_{LC} from [7] with axioms expressing the behaviour of \parallel and \mid in the presence of τ -actions (from [13]), and with the suitable τ -laws (from [20, 23]) necessary to deal with rooted branching bisimilarity. Then, we adjust the semantics of configurations given in Section 5 to the CCS_{LC} setting, and we use it to extend the definition of rooted branching bisimilarity to open CCS_{LC} terms (Definition 24). Usually, a behavioural equivalence \sim is defined over processes and is then possibly extended to open terms by saying that $t \sim u$ iff $\sigma(t) \sim \sigma(u)$ for all closed substitutions σ . However, we adopt the same approach of, e.g., [10, 16, 29], and present the definition of \sim_{RBB} directly over configurations. We will show in Section 9 that the two approaches yield the same equivalence relation over terms (Theorem 25). Finally, we apply the strategy used in [10] to obtain the completeness of the axiomatisation of prefix iteration with silent moves modulo rooted branching bisimilarity:

1. We identify *normal forms* for CCS_{LC} terms (Definition 27) and show that each term can be proven equal to a normal form using \mathcal{E}_{RBB} (Proposition 28).
2. We establish a relationship between \sim_{BB} and derivability in \mathcal{E}_{RBB} (Proposition 29).
3. We show that for all terms t, u , if $t \sim_{\text{RBB}} u$, then $\mathcal{E}_{\text{RBB}} \vdash t \approx u$ (Theorem 5).

9 Rooted branching bisimilarity over terms

In this section we discuss the decomposition of the semantics of CCS_{LC} terms, and the extension of the definition of (rooted) branching bisimilarity to open CCS_{LC} terms.

The first step towards our completeness result consists in providing a semantics for open CCS_{LC} terms. To this end, we need to extend the semantics of configurations given in Section 5. For the sake of readability, we present the syntax of CCS_{LC} configurations and the inference rules for variables and summations, even though they are identical to the corresponding ones presented in Section 5 for CCS. However, we omit the explanations on the roles of labels ℓ , ρ , and variables x_μ , as those can be found in Section 5. In particular, the use of variables $x_\mu \in \mathcal{V}_{\mathcal{A}_\tau}$ (as explained in Example 8) remains unchanged.

► **Definition 21** (CCS_{LC} configuration). *The collection of CCS_{LC} configurations, denoted by \mathcal{C}_{LC} , is given by:*

$$c ::= x_\mu \mid t \mid c \parallel c, \quad \text{where } t \text{ is a } \text{CCS}_{\text{LC}} \text{ term, and } x_\mu \in \mathcal{V}_{\mathcal{A}_\tau}.$$

The auxiliary transitions of the form $\xrightarrow{\ell}_\rho$ are formally defined via the inference rules in Table 5, where we omitted the rules (a'_1) and (a'_2) for prefixing and choice (which are identical to, respectively, rules (a_1) and (a_2) in Table 3) the symmetric rules to (a'_2) , (a'_4) , (a'_5) and (a'_6) , as well as the rules for \parallel . We remark that Lemma 10 can be easily extended to CCS_{LC} to show how a transition $\sigma(t) \xrightarrow{\mu} p$ can stem from transitions of the CCS_{LC} term t and of the process $\sigma(x)$, for $x \in \text{var}(t)$.

Since $\mathcal{V}_{\mathcal{A}_\tau}$ is disjoint from \mathcal{V} , we also need to introduce auxiliary rules for the special configuration $x_\mu \in \mathcal{V}_{\mathcal{A}_\tau}$. These are identified by a proper label x_μ on the transition and reported in Table 6 as rules (c_1) and (c_2) . To conclude our analysis of the decomposition

■ **Table 5** Inference rules for the transition relation $\xrightarrow{\ell}_{\rho}$ ($\mu \in \mathcal{A}_{\tau}$, $\alpha \in \mathcal{A} \cup \bar{\mathcal{A}}$).

$$(a'_3) \frac{t \xrightarrow{\ell}_{\rho} c}{t \parallel u \xrightarrow{\ell}_{\rho} c \parallel u}$$

$$(a'_4) \frac{t \xrightarrow{(x)}_{\alpha} c \quad u \xrightarrow{(y)}_{\bar{\alpha}} c'}{t \mid u \xrightarrow{(x,y)}_{\tau} c \parallel c'} \quad (a'_5) \frac{t \xrightarrow{(x)}_{\alpha} c \quad u \xrightarrow{\bar{\alpha}} u'}{t \mid u \xrightarrow{(x)}_{\alpha, \tau} c \parallel u'} \quad (a'_6) \frac{t \xrightarrow{\alpha} t' \quad u \xrightarrow{(x)}_{\bar{\alpha}} c}{t \mid u \xrightarrow{(x)}_{\bar{\alpha}, \tau} t' \parallel c}$$

■ **Table 6** Inference rules completing the operational semantics of CCS_{LC} configurations ($\mu \in \mathcal{A}_{\tau}$).

$$(c_1) \frac{}{x_{\mu} \xrightarrow{x_{\mu}} x_{\mu}} \quad (c_2) \frac{c_1 \xrightarrow{x_{\mu}} c'_1}{c_1 \parallel c_2 \xrightarrow{x_{\mu}} c'_1 \parallel c_2} \quad (c_3) \frac{c_1 \xrightarrow{\mu} c'_1}{c_1 \parallel c_2 \xrightarrow{\mu} c'_1 \parallel c_2} \quad (c_4) \frac{c_1 \xrightarrow{\ell}_{\rho} c'_1}{c_1 \parallel c_2 \xrightarrow{\ell}_{\rho} c'_1 \parallel c_2}$$

of the semantics of terms, we then need to extend the transition relations $\xrightarrow{\mu}$ and $\xrightarrow{\ell}_{\rho}$ to configurations. This is done by rules (c_3) and (c_4) in Table 6, where their symmetric counterparts have been omitted. Let $\xrightarrow{\xi}$ range over the possible transitions over configurations, i.e., $\xrightarrow{\xi}$ can be either $\xrightarrow{\mu}$, $\xrightarrow{\ell}_{\rho}$, or $\xrightarrow{x_{\mu}}$. The operational semantics of CCS_{LC} configurations is then given by the LTS whose states are configurations in \mathcal{C}_{LC} , whose actions are in $\mathcal{A}_{\tau} \cup \mathcal{V} \cup \mathcal{V}_{\mathcal{A}_{\tau}}$, and whose transitions are those that are provable from the rules in Tables 1, 4, 5, and 6.

Following the same approach of, e.g. [10,16,29], we now present the definitions of branching and rooted branching bisimulation equivalences directly over configurations.

► **Definition 22** (Branching bisimulation over configurations). *A symmetric relation \mathcal{R} over \mathcal{C}_{LC} is a branching bisimulation iff whenever $c_1 \mathcal{R} c_2$, if $c_1 \xrightarrow{\xi} c'_1$ then:*

- either $\xrightarrow{\xi} = \xrightarrow{\tau}$ and $c'_1 \mathcal{R} c_2$,
- or $c_2 \xrightarrow{\varepsilon} c'_2 \xrightarrow{\xi} c'_2$ for some c'_2, c'_2 such that $c_1 \mathcal{R} c'_2$ and $c'_1 \mathcal{R} c'_2$.

Two configurations c_1, c_2 are branching bisimilar, denoted by $c_1 \sim_{\text{BB}} c_2$, iff there exists a branching bisimulation \mathcal{R} such that $c_1 \mathcal{R} c_2$.

The definition of \sim_{BB} given in Definition 22 yields the same equivalence relation over configurations that we would have obtained with the standard approach, i.e., by defining $c_1 \sim_{\text{BB}} c_2$ iff $\sigma(c_1) \sim_{\text{BB}} \sigma(c_2)$ for all closed substitutions σ .

► **Theorem 23.** *For all configurations $c_1, c_2 \in \mathcal{C}_{\text{LC}}$ it holds that $c_1 \sim_{\text{BB}} c_2$ iff $\sigma(c_1) \sim_{\text{BB}} \sigma(c_2)$ for all closed substitutions σ .*

The approach for \sim_{BB} can be extended in a straightforward manner to \sim_{RBB} .

► **Definition 24** (Rooted branching bisimilarity over configurations). *Let $c_1, c_2 \in \mathcal{C}_{\text{LC}}$. We say that c_1 and c_2 are rooted branching bisimilar, denoted by $c_1 \sim_{\text{RBB}} c_2$, iff:*

- if $c_1 \xrightarrow{\xi} c'_1$ then $c_2 \xrightarrow{\xi} c'_2$ for some c'_2 such that $c'_1 \sim_{\text{BB}} c'_2$;
- if $c_2 \xrightarrow{\xi} c'_2$ then $c_1 \xrightarrow{\xi} c'_1$ for some c'_1 such that $c'_1 \sim_{\text{BB}} c'_2$.

► **Theorem 25.** *For all $c_1, c_2 \in \mathcal{C}_{\text{LC}}$ it holds that $c_1 \sim_{\text{RBB}} c_2$ iff $\sigma(c_1) \sim_{\text{RBB}} \sigma(c_2)$ for all closed substitutions σ .*

■ **Table 7** Equational basis modulo rooted branching bisimilarity.

Equational basis modulo strong bisimilarity: \mathcal{E}_B	
A0 $x + \mathbf{0} \approx x$	C0 $\mathbf{0} \mid x \approx \mathbf{0}$
A1 $x + y \approx y + x$	C1 $x \mid y \approx y \mid x$
A2 $(x + y) + z \approx x + (y + z)$	C2 $(x \mid y) \mid z \approx x \mid (y \mid z)$
A3 $x + x \approx x$	C3 $(x + y) \mid z \approx x \mid z + y \mid z$
L0 $\mathbf{0} \parallel x \approx \mathbf{0}$	C4 $\alpha x \mid \beta y \approx \tau(x \parallel y) \quad \text{if } \alpha = \bar{\beta}$
L1 $\mu x \parallel y \approx \mu(x \parallel y)$	C5 $\alpha x \mid \beta y \approx \mathbf{0} \quad \text{if } \alpha \neq \bar{\beta}$
L2 $(x \parallel y) \parallel z \approx x \parallel (y \parallel z)$	C6 $(x \parallel y) \mid z \approx (x \mid z) \parallel y$
L3 $x \parallel \mathbf{0} \approx x$	C7 $x \mid y \mid z \approx \mathbf{0}$
L4 $(x + y) \parallel z \approx x \parallel z + y \parallel z$	P $x \parallel y \approx x \parallel y + y \parallel x + x \mid y$
Additional axioms for \sim_{RBB} : $\mathcal{E}_{\text{RBB}} = \mathcal{E}_B \cup \{TB, TL\}$	
TB $\mu(\tau(x + y) + y) \approx \mu(x + y)$	TL $x \parallel \tau y \approx x \parallel y$
Derivable axioms	
D1 $x \parallel y \approx y \parallel x$	DT1 $\mu \tau x \approx \mu x$
D2 $(x \parallel y) \parallel z \approx x \parallel (y \parallel z)$	DT2 $x \parallel (\tau(y + z) + y) \approx x \parallel (y + z)$
D3 $(x \parallel y) \mid (z \parallel w) \approx (x \mid z) \parallel (y \parallel w)$	DT3 $\tau x \mid y \approx \mathbf{0}$
D4 $x \parallel \mathbf{0} \approx x$	

10

 The equational basis

We now present the complete axiomatisation for rooted branching bisimilarity over CCS_{LC} .

In [20] it was proved that if we consider the fragment BCCS of CCS (i.e., the fragment consisting only of $\mathbf{0}$, variables, prefixing, and choice), then a ground-complete axiomatisation of rooted branching bisimilarity over BCCS is given by $\mathcal{E}_0 \cup \{TB\}$, where $\mathcal{E}_0 = \{A0, A1, A2, A3\}$ from Table 2 (also reported in Table 7), and axiom TB is in Table 7. Informally, TB reflects that if executing a τ -step does not discard any observable behaviour, then it is redundant. In [7] it was proved that the axiom system \mathcal{E}_B given in Table 7, is a complete axiomatisation of bisimilarity over CCS_{LC} . Starting from these works, we now study a complete axiomatisation for \sim_{RBB} . Our aim is to show that the axiom system $\mathcal{E}_{\text{RBB}} = \mathcal{E}_B \cup \{TB, TL\}$ presented in Table 7 is a *complete axiomatisation of rooted branching bisimilarity* over CCS_{LC} .

If executing a τ -move does not resolve a choice within a parallel component, then it will also not resolve a choice of the parallel composition; axiom TL expresses a similar property of rooted branching bisimilarity for left merge. Interestingly, by combining TL and TB, it is possible to derive, as shown below, equation DT2 in Table 7, which is the equation for the left merge corresponding to TB.

$$x \parallel (\tau(y + z) + y) \stackrel{(TL)}{\approx} x \parallel \tau(\tau(y + z) + y) \stackrel{(TB)}{\approx} x \parallel \tau(y + z) \stackrel{(TL)}{\approx} x \parallel (y + z).$$

In Table 7 we report also some other equations that can be derived from \mathcal{E}_{RBB} , and that are useful in the technical development of our results. We refer the reader interested in the derivation proofs of D1–D3 and DT3 to [7]. Notice that DT1 corresponds essentially to the substitution instance of TB in which y is mapped to $\mathbf{0}$.

First of all, it is immediate to prove the soundness of \mathcal{E}_{RBB} modulo \sim_{RBB} .

► **Theorem 26 (Soundness).** *The axiom system \mathcal{E}_{RBB} is sound modulo \sim_{RBB} over CCS_{LC} .*

To obtain the desired completeness result, we apply the same strategy used in [10] that consists in the three steps discussed in Section 8.

Let us proceed to the first step: identifying normal forms for CCS_{LC} terms.

► **Definition 27 (Normal forms).** *The set of normal forms over CCS_{LC} is generated by the following grammar:*

$$\begin{aligned} S &::= \mu.N \quad | \quad x \ll N \quad | \quad (x \mid \alpha) \ll N \quad | \quad (x \mid y) \ll N \\ N &::= \mathbf{0} \quad | \quad S \quad | \quad N + N \end{aligned}$$

where $x, y \in \mathcal{V}$, $\mu \in \mathcal{A}_\tau$ and $\alpha \in \mathcal{A} \cup \overline{\mathcal{A}}$. Normal forms generated by S are also called simple normal forms and are characterised by the fact that they do not have $+$ as head operator.

► **Proposition 28.** *For every term t there is a normal form N such that $\mathcal{E}_{\text{RBB}} \vdash t \approx N$.*

We can then proceed to prove that branching bisimilar terms can be proven equal to rooted branching bisimilar terms using the axiom system \mathcal{E}_{RBB} .

► **Proposition 29.** *For CCS_{LC} terms t, u , if $t \sim_{\text{BB}} u$ then $\mathcal{E}_{\text{RBB}} \vdash \mu.t \approx \mu.u$, for any $\mu \in \mathcal{A}_\tau$.*

The completeness of the axiom system \mathcal{E}_{RBB} then follows from Proposition 28 and Proposition 29. Notice that axioms L1 and TB are actually axiom schemata that both generate $|\mathcal{A}_\tau|$ axioms. Similarly, the schema C4 generates $2|\mathcal{A}|$ axioms, and C5 generates $2|\mathcal{A}| \times (2|\mathcal{A}| - 1)$ axioms. Hence, \mathcal{E}_{RBB} is *finite* when so is the set of actions.

► **Theorem 5 (Completeness).** *Let t, u be CCS_{LC} terms. If $t \sim_{\text{RBB}} u$, then $\mathcal{E}_{\text{RBB}} \vdash t \approx u$.*

11 Concluding remarks

In this paper we have shown that the use of auxiliary operators, such as the left merge and communication merge, is crucial to obtain a finite, complete axiomatisation of the CCS parallel composition operator modulo rooted branching bisimilarity. Indeed, rooted branching bisimilarity does not afford a finite, complete axiomatisation over CCS without the auxiliary operators (our negative result), whereas CCS with the auxiliary operators added does have such a finite complete axiomatisation modulo rooted branching bisimilarity (our positive result).

A natural direction for future research is the extension of our results to other weak congruences from the spectrum [17]. The infinite family of equations used in the proof of our negative result (Theorem 4) is the same as that used by Moller to prove that CCS does not afford a finite complete axiomatisation of strong bisimilarity [32]. Our proof that the parametric property \mathbb{P}_n is preserved by provability from every collection of equations that are bounded in size by n and that are sound with respect to rooted branching bisimilarity refines Moller's proof that \mathbb{P}_n is preserved by provability if the equations are required to be sound with respect to strong bisimilarity. Our next goal will be to identify the weakest congruence \sim in the spectrum that includes strong bisimilarity and for which provability from a collection of sound equations that are sound with respect to \sim preserves \mathbb{P}_n . It will then follow that CCS does not afford a finite complete axiomatisation for all congruences including strong bisimilarity and included in \sim .

Regarding extensions of the positive result, we will focus on three weak congruences, namely *rooted η -bisimilarity* ($\sim_{\text{R}\eta\text{B}}$), *rooted delay bisimilarity* (\sim_{RDB}), and *rooted weak bisimilarity* (\sim_{RWB}), and provide complete axiomatisations for them. We are confident that the axiomatisation for $\sim_{\text{R}\eta\text{B}}$ can be obtained by exploiting a proof technique from [10] based on the notion of *saturation*. It should then be established that $\sim_{\text{R}\eta\text{B}}$ coincides with \sim_{RBB} on the class of η -saturated terms. Hence, if we can show that each term is provably equal to an η -saturated term using the axiom system for $\sim_{\text{R}\eta\text{B}}$, the completeness of the considered axiom system then directly follows from that for \sim_{RBB} we provided in this paper.

The quest for complete axiomatisations for \sim_{RDB} and \sim_{RWB} will require a different approach, as these equivalences are not preserved by the communication merge operator. For instance, we have that $\tau.a \sim_{\text{RWB}} \tau.a + a$, but $\tau.a \mid \bar{a}.b \not\sim_{\text{RWB}} (\tau.a + a) \mid \bar{a}.b$. Regarding \sim_{RDB} , similar observations can be made (see [18] for more details). The complete axiomatisation for observational congruence [23] (and thus rooted weak bisimilarity) over ACP_τ presented in [13] includes the axiom

$$\tau.x \mid y \approx x \mid y. \quad (\text{TC})$$

Similarly, in [1, 21] it was argued that in order to reason compositionally, and obtain an equational theory of CCS modulo observational congruence, it is necessary to define the operational semantics of communication merge in terms of inference rules of the form

$$\frac{t \xrightarrow{\alpha} t' \quad u \xrightarrow{\bar{\alpha}} u'}{t \mid u \xrightarrow{\tau} t' \parallel u'}$$

where we use $\xrightarrow{\mu}$ as a short-hand for the sequence of transitions $\xrightarrow{\varepsilon} \xrightarrow{\mu} \xrightarrow{\varepsilon}$. This means that in order for \mid to preserve \sim_{RWB} (and/or \sim_{RDB}), we need to consider a sequence of weak transitions as a single step. Clearly, since \mid is an auxiliary operator that we introduce specifically to obtain finite axiomatisations, its semantics can be defined in the most suitable way for our purposes, i.e., so that it is consistent with the considered congruence relation. However, it is also clear that if we modify the semantics of one operator in CCS_{LC} , then we are working with a new language. In particular, some axioms that are sound modulo *strong* bisimilarity (and thus also modulo \sim_{RBB}) over CCS_{LC} become unsound modulo rooted weak bisimilarity over the new language: this is the case of axioms C6 and C7 in Table 7. As a consequence, we cannot exploit the completeness of the axiomatisation for rooted branching bisimilarity to derive complete axiomatisations for rooted weak bisimilarity and rooted delay bisimilarity, but we must provide new axiomatisations for them and prove their completeness from scratch. Hence, we leave as future work the quest for complete axiomatisations for \sim_{RWB} and \sim_{RDB} over (recursion, relabelling, and restriction free) CCS with left merge and communication merge.

References

- 1 Luca Aceto. On “Axiomatising Finite Concurrent Processes”. *SIAM J. Comput.*, 23(4):852–863, 1994. doi:10.1137/S0097539793243600.
- 2 Luca Aceto, Elli Anastasiadi, Valentina Castiglioni, Anna Ingólfssdóttir, and Bas Luttik. In search of lost time: Axiomatising parallel composition in process algebras. In *Proceedings of LICS 2021*, pages 1–14. IEEE, 2021. doi:10.1109/LICS52264.2021.9470526.
- 3 Luca Aceto, Valentina Castiglioni, Wan Fokkink, Anna Ingólfssdóttir, and Bas Luttik. Are two binary operators necessary to finitely axiomatise parallel composition? In *Proceedings of CSL 2021*, volume 183 of *LIPICs*, pages 8:1–8:17, 2021. doi:10.4230/LIPICs.CSL.2021.8.

- 4 Luca Aceto, Valentina Castiglioni, Anna Ingólfssdóttir, and Bas Luttik. On the axiomatisation of branching bisimulation congruence over CCS, 2022. doi:10.48550/ARXIV.2206.13927.
- 5 Luca Aceto, Valentina Castiglioni, Anna Ingólfssdóttir, Bas Luttik, and Mathias R. Pedersen. On the axiomatisability of parallel composition: A journey in the spectrum. In *Proceedings of CONCUR 2020*, volume 171 of *LIPICs*, pages 18:1–18:22, 2020. doi:10.4230/LIPICs.CONCUR.2020.18.
- 6 Luca Aceto, David de Frutos-Escrig, Carlos Gregorio-Rodríguez, and Anna Ingólfssdóttir. Axiomatizing weak simulation semantics over BCCSP. *Theor. Comput. Sci.*, 537:42–71, 2014. doi:10.1016/j.tcs.2013.03.013.
- 7 Luca Aceto, Wan Fokkink, Anna Ingólfssdóttir, 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.
- 8 Luca Aceto, Wan Fokkink, Anna Ingólfssdóttir, and Sumit Nain. Bisimilarity is not finitely based over BPA with interrupt. *Theor. Comput. Sci.*, 366(1-2):60–81, 2006. doi:10.1016/j.tcs.2006.07.003.
- 9 Luca Aceto, Anna Ingólfssdóttir, Bas Luttik, and Paul van Tilburg. Finite equational bases for fragments of CCS with restriction and relabelling. In *Proceedings of IFIP TCS 2008*, volume 273 of *IFIP*, pages 317–332, 2008. doi:10.1007/978-0-387-09680-3_22.
- 10 Luca Aceto, Rob J. van Glabbeek, Wan Fokkink, and Anna Ingólfssdóttir. Axiomatizing prefix iteration with silent steps. *Inf. Comput.*, 127(1):26–40, 1996. doi:10.1006/inco.1996.0047.
- 11 Twan Basten. Branching bisimilarity is an equivalence indeed! *Inf. Process. Lett.*, 58(3):141–147, 1996. doi:10.1016/0020-0190(96)00034-8.
- 12 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.
- 13 Jan A. Bergstra and Jan Willem Klop. Algebra of communicating processes with abstraction. *Theor. Comput. Sci.*, 37:77–121, 1985. doi:10.1016/0304-3975(85)90088-X.
- 14 Taolue Chen, Wan Fokkink, and Rob J. van Glabbeek. Ready to preorder: The case of weak process semantics. *Inf. Process. Lett.*, 109(2):104–111, 2008. doi:10.1016/j.ip1.2008.09.003.
- 15 Rob J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In *Proceedings of CONCUR '90*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297, 1990. doi:10.1007/BFb0039066.
- 16 Rob J. van Glabbeek. A complete axiomatization for branching bisimulation congruence of finite-state behaviours. In *Proceedings of MFCS'93*, volume 711 of *Lecture Notes in Computer Science*, pages 473–484, 1993. doi:10.1007/3-540-57182-5_39.
- 17 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.
- 18 Rob J. van Glabbeek. On cool congruence formats for weak bisimulations. *Theor. Comput. Sci.*, 412(28):3283–3302, 2011. doi:10.1016/j.tcs.2011.02.036.
- 19 Rob J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). In *IFIP Congress*, pages 613–618, 1989.
- 20 Rob J. van Glabbeek and W. P. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, 1996. doi:10.1145/233551.233556.
- 21 Matthew Hennessy. Axiomatizing finite concurrent processes. *SIAM J. Comput.*, 17(5):997–1017, 1988. doi:10.1137/0217063.
- 22 Matthew Hennessy and Robin Milner. On observing nondeterminism and concurrency. In *Proceedings of ICALP 1980*, volume 85 of *Lecture Notes in Computer Science*, pages 299–309, 1980. doi:10.1007/3-540-10003-2_79.
- 23 Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985. doi:10.1145/2455.2460.
- 24 Tony Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

- 25 Robert M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):371–384, 1976. doi:10.1145/360248.360251.
- 26 Bas Luttik. Unique parallel decomposition in branching and weak bisimulation semantics. *Theor. Comput. Sci.*, 612:29–44, 2016. doi:10.1016/j.tcs.2015.10.013.
- 27 Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980. doi:10.1007/3-540-10235-3.
- 28 Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.
- 29 Robin Milner. A complete axiomatisation for observational congruence of finite-state behaviors. *Inf. Comput.*, 81(2):227–247, 1989. doi:10.1016/0890-5401(89)90070-9.
- 30 Faron Moller. *Axioms for Concurrency*. PhD thesis, Department of Computer Science, University of Edinburgh, July 1989. Report CST-59-89. Also published as ECS-LFCS-89-84.
- 31 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.
- 32 Faron Moller. The nonexistence of finite axiomatisations for CCS congruences. In *Proceedings of LICS '90*, pages 142–153, 1990. doi:10.1109/LICS.1990.113741.
- 33 Rocco De Nicola and Matthew Hennessy. Testing equivalence for processes. In *Proceedings of ICALP 1983*, volume 154 of *Lecture Notes in Computer Science*, pages 548–560, 1983. doi:10.1007/BFb0036936.
- 34 David M. R. Park. Concurrency and automata on infinite sequences. In *Proceedings of GI-Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183, 1981. doi:10.1007/BFb0017309.
- 35 Gordon D. Plotkin. A structural approach to operational semantics. Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.
- 36 Gordon D. Plotkin. A structural approach to operational semantics. *J. Log. Algebraic Methods Program.*, 60-61:17–139, 2004.

Non-Deterministic Abstract Machines

Małgorzata Biernacka

Institute of Computer Science, University of Wrocław, Poland

Dariusz Biernacki

Institute of Computer Science, University of Wrocław, Poland

Sergueï Lenglet

Université de Lorraine, Nancy, France

Alan Schmitt

INRIA, Rennes, France

Abstract

We present a generic design of abstract machines for non-deterministic programming languages, such as process calculi or concurrent lambda calculi, that provides a simple way to implement them. Such a machine traverses a term in the search for a redex, making non-deterministic choices when several paths are possible and backtracking when it reaches a dead end, i.e., an irreducible subterm. The search is guaranteed to terminate thanks to term annotations the machine introduces along the way.

We show how to automatically derive a non-deterministic abstract machine from a zipper semantics – a form of structural operational semantics in which the decomposition process of a term into a context and a redex is made explicit. The derivation method ensures the soundness and completeness of the machines w.r.t. the zipper semantics.

2012 ACM Subject Classification Theory of computation → Abstract machines

Keywords and phrases Abstract machines, non-determinism, lambda-calculus, process calculi

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.7

Related Version *Extended Version*: <https://hal.inria.fr/hal-03545768v2>

Funding This work is partially funded by PHC Polonium and by the National Science Centre of Poland under grant no. 2019/33/B/ST6/00289.

Acknowledgements We thank the anonymous reviewers for their comments.

1 Introduction

Abstract machines, i.e., first-order tail-recursive transition systems for term reduction, such as SECD [29], CEK [13], and the KAM [28], are a traditional and celebrated artifact in the area of programming languages based on the λ -calculus. They serve both as a form of operational semantics [12, 13, 29] and an implementation model [26, 33] of programming languages, but they also play a role in other areas, e.g., in proof theory [28], higher-order model checking [41], or cost models [1]. They are used as an implementation model also in concurrent languages [16, 18, 34, 37, 46], in particular to study distribution [4, 19–21, 24, 38].

Since in general designing a new abstract machine is a serious undertaking, several frameworks supporting mechanical or even automatic derivations of abstract machines from other forms of semantics have been developed [2, 7, 23, 44]. However, these frameworks assume a language that satisfies the unique decomposition property [7, 11], which entails that at each step one specific redex is selected, and thus the language follows a deterministic reduction strategy. This property does not hold in non-deterministic languages such as process calculi (or even in the λ -calculus without a fixed reduction strategy) and the existing methodology cannot be applied. Existing machines for non-deterministic languages are ad-hoc and may not be complete, i.e., not all reduction paths of the language can be simulated by the corresponding abstract machine [16, 18, 20, 34, 46].



© Małgorzata Biernacka, Dariusz Biernacki, Sergueï Lenglet, and Alan Schmitt; licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Sławomir Lasota, and Anca Muscholl; Article No. 7; pp. 7:1–7:24



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

This work presents a generic framework for the definition of complete abstract machines that implement a non-deterministic reduction relation in a systematic way. The idea is to go through a term to find a redex without following a specific strategy, picking arbitrarily a subterm when several are available – e.g., going left or right of an application in λ -calculus. The two main ideas are: (1) the machine should not remain stuck when it chooses a subterm which cannot reduce – in such a case we make it backtrack to its last choice; (2) the machine should not endlessly loop searching for redexes in subterms which cannot reduce – the machine annotates the subterms which are normal forms to prevent itself from visiting them again.

Non-deterministic machines designed in this way can be complex even for small languages, therefore we show how to generate them automatically from an intermediary *zipper semantics*. This semantics, inspired by Huet [25], is a form of structural operational semantics (SOS) [40] that remembers the current position in a term by building a context, i.e., a syntactic object that represents a term with a hole [14]. This format of semantics makes it explicit how a term is decomposed into a context and a redex, and thus it can be seen as a non-deterministic counterpart of the decomposition function in (deterministic) context-based reduction semantics [10,15]. While deterministic reduction semantics is directly implementable and the corresponding abstract machine can be viewed (roughly) as its optimization [11], non-deterministic reduction semantics, even when expressed as a zipper semantics, requires non-trivial instrumentation to become implementable in a complete way. Deriving the non-deterministic abstract machine (NDAM) from the zipper semantics consists exactly in such an instrumentation with the backtracking mechanism and normal-form annotations. We show how to derive an NDAM from an arbitrary zipper semantics that satisfies minimal conditions, and we prove that the resulting NDAM is sound and complete w.r.t. the semantics. Our approach applies in particular to process calculi, for which the abstract machines defined so far were ad-hoc and usually not complete.

The contributions of this paper are: (1) a generic design of sound and complete, non-deterministic abstract machines which cannot get stuck or infinitely loop in a redex search, (2) with a systematic derivation procedure from an intermediary format, called zipper semantics. The resulting machine is an implementation of the non-deterministic source language.

We illustrate our method on the λ -calculus without a fixed reduction strategy and on a minimal process calculus HOcore [31], respectively in Sections 2 and 3. We then give a derivation procedure of an NDAM from an arbitrary zipper semantics in Section 4. We discuss related work in Section 5 and future work in Section 6. The appendix contains further examples, including the zipper semantics and abstract machine of HO π [42] that extends HOcore with name restriction. An extended version [6] contains the proofs missing from the paper. An implementation of the derivation procedure is also available [5].

2 Lambda-calculus

As a warm-up example, we present the zipper semantics and the corresponding NDAM for the λ -calculus with no fixed reduction strategy.

2.1 Syntax and Context-based Reduction Semantics

We let t, s range over λ -terms. We denote application with an explicit operator $@$ to annotate it later on. We represent a context \mathbb{E} as a list of elementary contexts called *frames* \mathfrak{F} .

$$t, s ::= x \mid \lambda x.t \mid t@s \quad \mathfrak{F} ::= \lambda x \mid @t \mid t@ \quad \mathbb{E}, \mathbb{F}, \mathbb{G} ::= \bullet \mid \mathfrak{F}::\mathbb{E}$$

$$\begin{array}{c}
\begin{array}{ccccc}
\text{init} & \text{appL} & \text{appR} & \text{app}\lambda & \text{app}\beta \\
\frac{t \xrightarrow{\bullet} t'}{t \rightarrow_{\text{zs}} t'} & \frac{t \xrightarrow{@ s :: \mathbb{E}} t'}{t @ s \xrightarrow{\mathbb{E}} t'} & \frac{s \xrightarrow{t @ :: \mathbb{E}} s'}{t @ s \xrightarrow{\mathbb{E}} s'} & \frac{t \xrightarrow{\lambda x :: \mathbb{E}} t'}{\lambda x.t \xrightarrow{\mathbb{E}} t'} & \frac{t \xrightarrow{s, \mathbb{E}} t'}{t @ s \xrightarrow{\mathbb{E}} t'} \\
\end{array} \\
\text{lamb}\beta \\
\hline
\lambda x.t \xrightarrow{s, \mathbb{E}}_{\text{lamb}} \mathbb{E}[t\{s/x\}]
\end{array}$$

■ **Figure 1** Zipper semantics for the λ -calculus.

Because it is more convenient for the definition of the machine, we interpret contexts inside-out [12]: the head of the context is the innermost frame. The definition of plugging a term in a context $\mathbb{E}[t]$ is therefore as follows:

$$\bullet[t] \triangleq t \quad (\lambda x :: \mathbb{E})[t] \triangleq \mathbb{E}[\lambda x.t] \quad (@ s :: \mathbb{E})[t] \triangleq \mathbb{E}[t @ s] \quad (s @ :: \mathbb{E})[t] \triangleq \mathbb{E}[s @ t]$$

We write $t\{s/x\}$ for the capture-avoiding substitution of x by s in t , and define the context-based reduction semantics \rightarrow_{rs} of the λ -calculus by the following rule

$$\mathbb{E}[(\lambda x.t) @ s] \rightarrow_{\text{rs}} \mathbb{E}[t\{s/x\}]$$

which can be read declaratively: if we find a redex in a context \mathbb{E} built according to the given grammar of contexts, then we can reduce. This format of semantics does not make it apparent how to *decompose* a term to find a redex. On the other hand, structural operational semantics offers another common semantic format that makes it more explicit how to navigate in a term to find a redex, but it does not store the traversed path.

2.2 Zipper Semantics

A first step towards an abstract machine is to make explicit the step-by-step decomposition of a term into a context and a redex. To this end, we propose zipper semantics, a combination of SOS and reduction semantics. Like a regular SOS, a zipper semantics goes through a term looking for a redex using structural rules, except the current position in the term is made explicit with a context as in reduction semantics.

The zipper semantics for the λ -calculus is defined in Figure 1. It looks for a β -redex while constructing the surrounding context \mathbb{E} at the same time. The decomposition happens in the rules **appL**, **appR**, and **app λ** , where we search for a redex by descending into the appropriate subterm of a given term. Each of these rules corresponds to a frame, with **init** initiating the search by setting the context to \bullet .

These rules actually look for the application at the root of the β -redex; checking that an application $t @ s$ is indeed a β -redex is done by the rule **app β** . It relies on an auxiliary transition $t \xrightarrow{s, \mathbb{E}}_{\text{lamb}} t'$, which checks that its source is indeed a λ -abstraction. In that case, we can β -reduce with rule **lamb β** . We can see that computation only occurs in the axiom; the other rules are simply propagating the result unchanged.

One may wonder why we need the rules **app β** and **lamb β** while a single axiom $(\lambda x.t) @ s \xrightarrow{\mathbb{E}}_{\text{app}} \mathbb{E}[t\{s/x\}]$ is enough to recognize a β -redex. The reason is that we restrict ourselves to patterns discriminating only the head constructor of a term, to remain close to an abstract machine where the decomposition of a term occurs only one operator at a time.

We prove that the zipper semantics and reduction semantics coincide in Appendix A.

► **Example 1.** To illustrate further how to recognize a redex one operator at a time, suppose we restrict the argument of the β -redex to a value $v ::= x \mid \lambda x.t$, so that $\mathbb{E}[(\lambda x.t) @ v] \rightarrow_{rs} \mathbb{E}[t\{v/x\}]$. In such a case, we would need an extra transition $s \xrightarrow{x,t,\mathbb{E}}_v t'$, checking that s is a value. The rule $\text{lam}\beta$ would be replaced by the rule $\text{lam}\beta^v$ below.

$$\frac{\text{lam}\beta^v \quad s \xrightarrow{x,t,\mathbb{E}}_v t'}{\lambda x.t \xrightarrow{s,\mathbb{E}}_{\text{lam}} t'} \quad \frac{\text{var}^v \quad y \xrightarrow{x,t,\mathbb{E}}_v \mathbb{E}[t\{y/x\}]}{y \xrightarrow{x,t,\mathbb{E}}_v \mathbb{E}[t\{y/x\}]} \quad \frac{\text{lam}^v \quad \lambda y.s \xrightarrow{x,t,\mathbb{E}}_v \mathbb{E}[t\{\lambda y.s/x\}]}{\lambda y.s \xrightarrow{x,t,\mathbb{E}}_v \mathbb{E}[t\{\lambda y.s/x\}]}$$

2.3 Non-Deterministic Abstract Machine

Design principles. Zipper semantics describes how to decompose a term into a redex and a context, but it is not yet an implementation, as it does not explain what to do when several rules can be applied, like appL , appR , and $\text{app}\beta$. The NDAM simply picks one of them, and backtracks if it reaches a dead-end. We present how we implement this backtracking and how it can be derived from the zipper rules, before giving the formal definition of the NDAM.

The decomposition at work in the zipper semantics rules can be turned into machine steps: we see the change of focus occurring in the source term between the conclusion and the premise. We introduce a machine mode for each transition kind (here, app and lam), and the rules appL , appR , and $\text{app}\beta$ are translated to the following *forward* machine steps, with $|$ separating the term from the context:

$$\langle t @ s \mid \mathbb{E} \rangle_{\text{app}} \mapsto \langle t \mid @ s :: \mathbb{E} \rangle_{\text{app}} \quad \langle t @ s \mid \mathbb{E} \rangle_{\text{app}} \mapsto \langle s \mid t @ :: \mathbb{E} \rangle_{\text{app}} \quad \langle t @ s \mid \mathbb{E} \rangle_{\text{app}} \mapsto \langle t \mid s, \mathbb{E} \rangle_{\text{lam}}$$

We see why interpreting the context inside-out is convenient: focusing on t in $\mathbb{E}[t @ s]$ amounts to pushing the frame $@ s$ on top of \mathbb{E} . It is the same as decomposing the term as $(@ s :: \mathbb{E})[t]$: the innermost constructor becomes the topmost one in the context.

The resulting machine is non-deterministic as three different steps can be taken from the configuration $\langle t @ s \mid \mathbb{E} \rangle_{\text{app}}$. Unlike typical deterministic machines, it does not implement a strategy and does not choose, e.g., to always go left of an application as in the KAM [28]. A consequence is that the machine can make a wrong choice, i.e., focus on a term which cannot reduce, like a variable. In such cases, we want the machine to backtrack to the last configuration for which a choice had to be made, and no further. To do so, we record the applied rules in a stack π . When we reach a term which cannot reduce, we switch to a backtracking mode (here, bapp) where we can “unapply” a rule.

$$\begin{aligned} \langle t @ s ; \pi \mid \mathbb{E} \rangle_{\text{app}} &\mapsto \langle t ; \text{appL} :: \pi \mid @ s :: \mathbb{E} \rangle_{\text{app}} \\ \langle x ; \pi \mid \mathbb{E} \rangle_{\text{app}} &\mapsto \langle \pi ; x \mid \mathbb{E} \rangle_{\text{bapp}} \\ \langle \text{appL} :: \pi ; t \mid @ s :: \mathbb{E} \rangle_{\text{bapp}} &\mapsto \langle t @ s ; \pi \mid \mathbb{E} \rangle_{\text{app}} \end{aligned}$$

The machine may try other rules on $t @ s$, e.g., to find a redex in s . However, it should not try appL again, as the backtracking step implies there is no redex in t . We refer to backtracking steps like the last one as *backward*, and to steps like the middle one as *switching*. The backward step is simply the reverse of the corresponding forward step.

We prevent the machine from choosing a previously explored path by annotating the root operator of an already tested subterm. An annotation $t @^{\text{app}} s$ means that $t @ s$ has already been tried for $\xrightarrow{\mathbb{E}}_{\text{app}}$ transitions and is a normal form for it. Similarly, a term annotated lam is a normal form w.r.t. $\xrightarrow{s,\mathbb{E}}_{\text{lam}}$ (it is not a λ -abstraction). A term can be annotated with both app and lam , for instance if it is a variable.

The machine can take a forward step only if the term in focus has not been already tested. For $t@s$, we try `appL` (resp. `appR`) only if t (resp. s) is not annotated `app`, and `appβ` only if t is not annotated `lam`. If none of the steps applies because of the annotations, then all possible rules have been tried and $t@s$ is a normal form for `app`: the machine backtracks and annotates the term accordingly. In what follows, Σ represents an annotation set.

$$\begin{aligned} \langle x^\Sigma; \pi | \mathbb{E} \rangle_{\text{app}} &\mapsto \langle \pi; x^{\Sigma \cup \{\text{app}\}} | \mathbb{E} \rangle_{\text{bapp}} \\ \langle t@s; \pi | \mathbb{E} \rangle_{\text{app}} &\mapsto \langle \pi; t@s^{\Sigma \cup \{\text{app}\}} | \mathbb{E} \rangle_{\text{bapp}} \text{ if no other step applies} \end{aligned}$$

Switching steps are of two kinds: either the language construct does not have a forward step for a given mode (like a variable in the `app` mode), or all possible rules have been tried for the construct. They both can be derived from the zipper semantics by looking at which rule can be applied to each construct. This derivation is made easier by the constraint that the decomposition occurs one operator at a time in zipper rules. If we allowed for more complex patterns such as $(\lambda x.t)@s$, we would have to create a switching step for the terms not fitting this pattern, like $x@s$, and enumerating these anti-patterns would be more difficult [27].

Finally, because we store the annotations of a term in its root operator, we need to remember them when a forward step removes the operator, to be able to restore them when we backtrack. We do so in the stack π .

$$\begin{aligned} \langle t@s; \pi | \mathbb{E} \rangle_{\text{app}} &\mapsto \langle t; (\text{appL}, \Sigma) :: \pi | @s :: \mathbb{E} \rangle_{\text{app}} \\ \langle (\text{appL}, \Sigma) :: \pi; t | @s :: \mathbb{E} \rangle_{\text{bapp}} &\mapsto \langle t@s; \pi | \mathbb{E} \rangle_{\text{app}} \end{aligned}$$

In this simple example we could do without the stack because the contexts encode precisely the rules that have been applied along the way. In general, however, a single context cannot always reflect the derivation tree, as we can see in the $\text{HO}\pi$ example (Appendix C).

The next example illustrates how annotations work, and also that they may no longer hold after reduction. Therefore they should be erased before searching for the next redex.

► **Example 2.** Let $\Omega \triangleq (\lambda^\emptyset x.x^\emptyset @^\emptyset x^\emptyset) @^\emptyset (\lambda^\emptyset x.x^\emptyset @^\emptyset x^\emptyset)$. We show a possible machine run for this term, where we label forward and backward steps with the rule they apply or unapply, and switching steps with a constant τ . For readability, we write only the term under focus.

The machine may first go left and under the λ -abstraction.

$$\langle \Omega | \dots \rangle_{\text{app}} \xrightarrow{\text{appL}} \xrightarrow{\text{app}\lambda} \langle x^\emptyset @^\emptyset x^\emptyset | \dots \rangle_{\text{app}}$$

At that point, it may test whether the application is a β -redex. Since it is not the case, it backtracks, annotating the variable in function position.

$$\langle x^\emptyset @^\emptyset x^\emptyset | \dots \rangle_{\text{app}} \xrightarrow{\text{app}\beta} \langle x^\emptyset | \dots \rangle_{\text{lam}} \xrightarrow{\tau} \xrightarrow{-\text{app}\beta} \langle x^{\text{lam}} @^\emptyset x^\emptyset | \dots \rangle_{\text{app}}$$

From there, it necessarily tests the other possibilities `appL` and `appR` (in no predefined order), and fails in both cases.

$$\langle x^{\text{lam}} @^\emptyset x^\emptyset | \dots \rangle_{\text{app}} \xrightarrow{\text{appL}} \xrightarrow{\tau} \xrightarrow{-\text{appL}} \xrightarrow{\text{appR}} \xrightarrow{\tau} \xrightarrow{-\text{appR}} \langle x^{\{\text{app}, \text{lam}\}} @^\emptyset x^{\text{app}} | \dots \rangle_{\text{app}}$$

Then it can only backtrack to reconstruct the λ -abstraction on the left, and then the whole term.

$$\begin{aligned} \langle x^{\{\text{app}, \text{lam}\}} @^\emptyset x^{\text{app}} | \dots \rangle_{\text{app}} &\xrightarrow{\tau} \xrightarrow{-\text{app}\lambda} \langle \lambda^\emptyset x.x^{\{\text{app}, \text{lam}\}} @^{\text{app}} x^{\text{app}} | \dots \rangle_{\text{app}} \\ &\xrightarrow{\tau} \xrightarrow{-\text{appL}} \langle (\lambda^{\text{app}} x.x^{\{\text{app}, \text{lam}\}} @^{\text{app}} x^{\text{app}}) @^\emptyset (\lambda^\emptyset x.x^\emptyset @^\emptyset x^\emptyset) | \dots \rangle_{\text{app}} \end{aligned}$$

$$\begin{aligned}
 \langle t \rangle_{\text{zs}} &\mapsto \langle t; \text{init} \mid \bullet \rangle_{\text{app}} \\
 \langle t @^{\Sigma} s; \pi \mid \mathbb{E} \rangle_{\text{app}} &\mapsto \langle t; (\text{appL}, \Sigma) :: \pi \mid @ s :: \mathbb{E} \rangle_{\text{app}} && \text{if } \text{app} \notin \text{an}(t) \\
 \langle t @^{\Sigma} s; \pi \mid \mathbb{E} \rangle_{\text{app}} &\mapsto \langle s; (\text{appR}, \Sigma) :: \pi \mid t @ :: \mathbb{E} \rangle_{\text{app}} && \text{if } \text{app} \notin \text{an}(s) \\
 \langle t @^{\Sigma} s; \pi \mid \mathbb{E} \rangle_{\text{app}} &\mapsto \langle t; (\text{app}\beta, \Sigma) :: \pi \mid s, \mathbb{E} \rangle_{\text{lam}} && \text{if } \text{lam} \notin \text{an}(t) \\
 \langle \lambda^{\Sigma} x.t; \pi \mid \mathbb{E} \rangle_{\text{app}} &\mapsto \langle t; (\text{app}\lambda, \Sigma) :: \pi \mid \lambda x :: \mathbb{E} \rangle_{\text{app}} && \text{if } \text{app} \notin \text{an}(t) \\
 \langle t; \pi \mid \mathbb{E} \rangle_{\text{app}} &\mapsto \langle \pi; t^{\cup \text{app}} \mid \mathbb{E} \rangle_{\text{bapp}} && \text{otherwise} \\
 \\
 \langle \text{init}; t \mid \bullet \rangle_{\text{bapp}} &\mapsto \langle t \rangle_{\text{nf}} \\
 \langle (\text{appL}, \Sigma) :: \pi; t \mid @ s :: \mathbb{E} \rangle_{\text{bapp}} &\mapsto \langle t @^{\Sigma} s; \pi \mid \mathbb{E} \rangle_{\text{app}} \\
 \langle (\text{appR}, \Sigma) :: \pi; s \mid t @ :: \mathbb{E} \rangle_{\text{bapp}} &\mapsto \langle t @^{\Sigma} s; \pi \mid \mathbb{E} \rangle_{\text{app}} \\
 \langle (\text{app}\beta, \Sigma) :: \pi; t \mid s, \mathbb{E} \rangle_{\text{blam}} &\mapsto \langle t @^{\Sigma} s; \pi \mid \mathbb{E} \rangle_{\text{app}} \\
 \langle (\text{app}\lambda, \Sigma) :: \pi; t \mid \lambda x :: \mathbb{E} \rangle_{\text{bapp}} &\mapsto \langle \lambda^{\Sigma} x.t; \pi \mid \mathbb{E} \rangle_{\text{app}} \\
 \\
 \langle \lambda^{\Sigma} x.t; \pi \mid s, \mathbb{E} \rangle_{\text{lam}} &\mapsto \langle |\mathbb{E}[t\{s/x\}]| \rangle_{\text{zs}} \\
 \langle t; \pi \mid s, \mathbb{E} \rangle_{\text{lam}} &\mapsto \langle \pi; t^{\cup \text{lam}} \mid s, \mathbb{E} \rangle_{\text{blam}} && \text{otherwise}
 \end{aligned}$$

■ **Figure 2** Non-Deterministic Abstract Machine for the λ -calculus.

The machine can then look for a redex in the λ -abstraction on the right, and it would result in the same annotations as for the one on the left, not necessarily generated in the same order. It can also rightfully recognize the term as a β -redex, with the sequence $\xrightarrow{\text{app}\beta} \xrightarrow{\text{lam}\beta}$, the last step performing the reduction. After the reduction, we should also erase the remaining annotations. If we do not erase them, the result of the reduction would be

$$\langle (\lambda^{\emptyset} x.x^{\emptyset} @^{\emptyset} x^{\emptyset}) @^{\{\text{app}\}} (\lambda^{\emptyset} x.x^{\emptyset} @^{\emptyset} x^{\emptyset}) \mid \dots \rangle_{\text{app}}$$

and the **app** annotation would wrongfully signal the term as a normal-form, preventing it from being reduced. Erasing all the remaining annotations ensures the machine finds the next redex, but a finer, language-specific analysis would erase only the problematic annotations. We leave such an optimization as a future work. \lrcorner

Formal definition. We let α range over annotations, Σ over annotation sets, and denote the empty set by \emptyset . We extend the λ -calculus syntax as follows:

$$\alpha ::= \text{app} \mid \text{lam} \quad t, s ::= x^{\Sigma} \mid \lambda^{\Sigma} x.t \mid t @^{\Sigma} s$$

We write $\text{an}(t)$ for the annotation set at the root of t , e.g., $\text{an}(t @^{\Sigma} s) \triangleq \Sigma$. We write $t^{\cup \alpha}$ for its extension with α so that $\text{an}(t^{\cup \alpha}) = \text{an}(t) \cup \{\alpha\}$. We write $|t|$ for the erasure of t , where all the annotation sets in t are made empty.

The syntax of contexts uses annotated terms, and plugging returns an annotated term where the annotation sets of the context operators are empty: e.g., $(\lambda x :: \mathbb{E})[t] \triangleq \mathbb{E}[\lambda^{\emptyset} x.t]$. Plugging is used only after a reduction step, where all the annotation sets are erased anyway.

We let ρ range over rule names and π over rule stacks, defined as $\pi ::= \text{init} \mid (\rho, \Sigma) :: \pi$. The definition of the machine for the λ -calculus is given in Figure 2.

A *forward* configuration $\langle t; \pi | \mathbb{E} \rangle_m$ (with $m \in \{\text{app}, \text{lam}\}$) discriminates on (the root operator of) t to apply a rule of the zipper semantics. For an inductive rule, it results in a change of focus and an extension of the stack, on which we record the applied rule and the annotation set of the root operator. Taking such a step is possible only if the new term under focus is not a normal form. A special case of forward step is the *initial* one from $\langle t \rangle_{\text{zs}}$ which does not have a side-condition, as we assume the annotation sets of t to be empty.

The β -reduction happens in the first transition of the lam mode. Backtracking is no longer necessary so we drop the stack. We reconstruct the entire term, and switch to the initial mode to search for a new redex starting from the root of the new term. We erase all annotations, as they may no longer be valid, as illustrated by Example 2.

If a forward configuration cannot apply a rule, we switch to the corresponding *backward* mode, annotating t in the process: these are the two “otherwise” steps. A backward configuration $\langle \pi; t | \mathbb{E} \rangle_{\text{bm}}$ inspects the stack π to unapply the rule at its top. While a backward step restores the configuration of the corresponding forward step, the term contains more annotations after a backward step than before taking the forward step: in $\langle \pi; t | \mathbb{E} \rangle_{\text{bm}}$, we have $m \in \text{an}(t)$ by construction. The annotations prevent the machine from reapplying a rule it just unapplied. The *normal form* mode $\langle t \rangle_{\text{nf}}$ signals that the term cannot reduce.

A machine run starts with an initial configuration $\langle t \rangle_{\text{zs}}$ where all the annotation sets of t are empty. The semantics of the machine is given by these configurations: if $\langle t \rangle_{\text{zs}} \mapsto^+ \langle t' \rangle_{\text{zs}}$ such that the sequence \mapsto^+ does not go through another initial configuration, then $t \rightarrow_{\text{zs}} t'$. Similarly, if $\langle t \rangle_{\text{zs}} \mapsto^+ \langle t' \rangle_{\text{nf}}$, then $|t'| = t$ and t is a normal form. We state the correspondence and termination theorems independently from the source zipper semantics in Section 4.

3 HOcore

We consider a minimal process calculus called HOcore [31], which can be seen as an extension of the λ -calculus with parallel composition.

3.1 Syntax and Semantics

We let a, b range over *channel names*, X, Y over *process variables*, and we define the syntax of processes as follows.

$$P, Q, R ::= X \mid \mathbf{0} \mid P \parallel Q \mid a(X).P \mid \bar{a}\langle P \rangle$$

The process $\mathbf{0}$ is the inactive process, $P \parallel Q$ runs P and Q in parallel, and a communication may happen between an input $a(X).P$ and an output $\bar{a}\langle Q \rangle$ that run in parallel. The communication is asynchronous because a message output does not have a continuation [43]; we discuss the synchronous case in Remark 3. In spite of its minimal number of constructors, HOcore is Turing-complete [31].

The semantics of process calculi is usually presented either with a structural congruence relation which reorders terms to make redexes appear, bringing input and output processes together, or with a labeled transition system which preserves the structure of the term [43]. Instead, we present it first as a reduction semantics with explicit contexts, as in Section 2.1, which makes it easier to come up with (or translate into) the corresponding zipper semantics.

We define frames as $\mathfrak{F} ::= \parallel P \mid P \parallel$ and plugging as follows.

$$\bullet[P] \triangleq P \quad (\parallel Q :: \mathbb{E})[P] \triangleq \mathbb{E}[P \parallel Q] \quad (Q \parallel :: \mathbb{E})[P] \triangleq \mathbb{E}[Q \parallel P]$$

$$\begin{array}{c}
\text{init} \\
\frac{P \xrightarrow{\bullet}_{\text{par}} P'}{P \rightarrow_{\text{zs}} P'} \\
\\
\text{parL} \\
\frac{P \parallel Q \xrightarrow{\mathbb{E}}_{\text{par}} P'}{P \parallel Q \xrightarrow{\mathbb{E}}_{\text{par}} P'} (s) \\
\\
\text{parOutL} \\
\frac{P \xrightarrow{\bullet, \mathcal{L}, \mathbb{E}, Q}_{\text{out}} P'}{P \parallel Q \xrightarrow{\mathbb{E}}_{\text{par}} P'} \\
\\
\text{parOutR} \\
\frac{Q \xrightarrow{\bullet, \mathcal{R}, \mathbb{E}, P}_{\text{out}} P'}{P \parallel Q \xrightarrow{\mathbb{E}}_{\text{par}} P'} \\
\\
\text{outParL} \\
\frac{P \parallel Q \xrightarrow{\mathbb{F}, \mathcal{S}, \mathbb{E}, R}_{\text{out}} P'}{P \parallel Q \xrightarrow{\mathbb{F}, \mathcal{S}, \mathbb{E}, R}_{\text{out}} P'} (s) \\
\\
\text{outIn} \\
\frac{R \xrightarrow{\bullet, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}}_{\text{in}} P'}{\bar{a}\langle P \rangle \xrightarrow{\mathbb{F}, \mathcal{S}, \mathbb{E}, R}_{\text{out}} P'} \\
\\
\text{inParL} \\
\frac{R \parallel Q \xrightarrow{\mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}}_{\text{in}} P'}{R \parallel Q \xrightarrow{\mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}}_{\text{in}} P'} (s) \\
\\
\text{inComL} \\
\frac{b = a}{b(X).R \xrightarrow{\mathbb{G}, \mathcal{L}, a, P, \mathbb{E}, \mathbb{F}}_{\text{in}} \mathbb{E}[\mathbb{F}[\mathbf{0}] \parallel \mathbb{G}[R\{P/X\}]]} (s)
\end{array}$$

■ **Figure 3** Output-first Zipper Semantics for HOcore.

A redex is a parallel composition with an input on one side and an output on the same name on the other side, both surrounded with contexts. The general formulation of such communication sites in a program can be expressed with the following reduction semantics, where we write $P\{Q/X\}$ for the capture-avoiding substitution of X by Q in P :

$$\begin{array}{l}
\mathbb{E}[\mathbb{F}[\bar{a}\langle Q \rangle] \parallel \mathbb{G}[a(X).P]] \rightarrow_{\text{rs}} \mathbb{E}[\mathbb{F}[\mathbf{0}] \parallel \mathbb{G}[P\{Q/X\}]] \\
\mathbb{E}[\mathbb{G}[a(X).P] \parallel \mathbb{F}[\bar{a}\langle Q \rangle]] \rightarrow_{\text{rs}} \mathbb{E}[\mathbb{G}[P\{Q/X\}] \parallel \mathbb{F}[\mathbf{0}]]
\end{array}$$

3.2 Zipper Semantics

Finding an HOcore redex requires us to recognize three constructs (parallel composition along with output and input on a shared name) and build the contexts \mathbb{E} , \mathbb{F} , and \mathbb{G} . The first step is to find the parallel composition; once the communicating processes $P \parallel Q$ are found, the communication rules of typical LTSs for process calculi [31, 42, 43] have two premises looking for the output P and the input in P and Q respectively. To be closer to an abstract machine, we sequentialize the search by looking for the output first (while constructing \mathbb{F}) and then the input (with \mathbb{G}) – the opposite choice would produce a completely symmetric semantics. Figure 3 presents such an output-first zipper semantics, where we omit the symmetric versions of the rules marked with the symbol (s). The resulting semantics is close to complementary semantics [32], where the communication is also sequentialized.

The transition $\xrightarrow{\mathbb{E}}_{\text{par}}$ is looking for the parallel composition while building \mathbb{E} : it proceeds as $\xrightarrow{\mathbb{E}}_{\text{app}}$ in the λ -calculus. Once we find the parallel composition, we look for the output either on the left or on the right with respectively rules **parOutL** and **parOutR**. We record the side we pick with a parameter $\mathcal{S} ::= \mathcal{L} \mid \mathcal{R}$. For example, in rule **parOutL**, we look for an output in P on the left (\mathcal{L}), remembering that we should later search for a corresponding input in Q . We also initialize the context \mathbb{F} surrounding the output with \bullet and remember \mathbb{E} as the context enclosing the whole redex.

The transition $\xrightarrow{\mathbb{F}, \mathcal{S}, \mathbb{E}, R}_{\text{out}}$ decomposes its source process to find an output, building \mathbb{F} at the same time: the other parameters \mathcal{S} , \mathbb{E} , and R remain unchanged during the search. When we find the output $\bar{a}\langle P \rangle$ (rule **outIn**), we look for a corresponding input in R using $\xrightarrow{\mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}}_{\text{in}}$, which builds the context \mathbb{G} during the search. Once we find an input on a , we compute the result of the communication, which depends whether the output is on the left (rule **inComL**) or on the right (omitted rule **inComR**).

$$\begin{array}{l}
\langle P \rangle_{\text{zs}} \mapsto \langle P ; \text{init} \mid \bullet \rangle_{\text{par}} \\
\langle P \parallel^{\Sigma} Q ; \pi \mid \mathbb{E} \rangle_{\text{par}} \mapsto \langle P ; (\text{parL}, \Sigma) :: \pi \mid \parallel Q :: \mathbb{E} \rangle_{\text{par}} \quad \text{if } \text{par} \notin \text{an}(P) \\
\langle P \parallel^{\Sigma} Q ; \pi \mid \mathbb{E} \rangle_{\text{par}} \mapsto \langle P ; (\text{parOutL}, \Sigma) :: \pi \mid \bullet, \mathcal{L}, \mathbb{E}, Q \rangle_{\text{out}} \quad \text{if } (\text{out}, |Q|) \notin \text{an}(P) \\
\langle P \parallel^{\Sigma} Q ; \pi \mid \mathbb{E} \rangle_{\text{par}} \mapsto \langle Q ; (\text{parOutR}, \Sigma) :: \pi \mid \bullet, \mathcal{R}, \mathbb{E}, P \rangle_{\text{out}} \quad \text{if } (\text{out}, |P|) \notin \text{an}(Q) \\
\langle P ; \pi \mid \mathbb{E} \rangle_{\text{par}} \mapsto \langle \pi ; P^{\cup \text{par}} \mid \mathbb{E} \rangle_{\text{bpar}} \quad \text{otherwise} \\
\langle \text{init} ; P \mid \bullet \rangle_{\text{bpar}} \mapsto \langle P \rangle_{\text{nf}} \\
\langle (\text{parL}, \Sigma) :: \pi ; P \mid \parallel Q :: \mathbb{E} \rangle_{\text{bpar}} \mapsto \langle P \parallel^{\Sigma} Q ; \pi \mid \mathbb{E} \rangle_{\text{par}} \\
\langle P \parallel^{\Sigma} Q ; \pi \mid \mathbb{F}, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} \mapsto \langle P ; (\text{outParL}, \Sigma) :: \pi \mid \parallel Q :: \mathbb{F}, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} \quad \text{if } (\text{out}, |R|) \notin \text{an}(P) \\
\langle \bar{a}^{\Sigma} \langle P \rangle ; \pi \mid \mathbb{F}, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} \mapsto \langle R ; (\text{outIn}, \Sigma) :: \pi \mid \bullet, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{in}} \quad \text{if } (\text{in}, a) \notin \text{an}(R) \\
\langle P ; \pi \mid \mathbb{F}, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} \mapsto \langle \pi ; P^{\cup (\text{out}, |R|)} \mid \mathbb{F}, \mathcal{S}, \mathbb{E}, R \rangle_{\text{bout}} \quad \text{otherwise} \\
\langle (\text{parOutL}, \Sigma) :: \pi ; P \mid \bullet, \mathcal{L}, \mathbb{E}, Q \rangle_{\text{bout}} \mapsto \langle P \parallel^{\Sigma} Q ; \pi \mid \mathbb{E} \rangle_{\text{par}} \\
\langle (\text{parOutR}, \Sigma) :: \pi ; Q \mid \bullet, \mathcal{R}, \mathbb{E}, P \rangle_{\text{bout}} \mapsto \langle P \parallel^{\Sigma} Q ; \pi \mid \mathbb{E} \rangle_{\text{par}} \\
\langle (\text{outParL}, \Sigma) :: \pi ; P \mid \parallel Q :: \mathbb{F}, \mathcal{S}, \mathbb{E}, R \rangle_{\text{bout}} \mapsto \langle P \parallel^{\Sigma} Q ; \pi \mid \mathbb{F}, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} \\
\langle R \parallel^{\Sigma} Q ; \pi \mid \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{in}} \mapsto \langle R ; (\text{inParL}, \Sigma) :: \pi \mid \parallel Q :: \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{in}} \quad \text{if } (\text{in}, a) \notin \text{an}(R) \\
\langle b^{\Sigma}(X).R ; \pi \mid \mathbb{G}, \mathcal{L}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{in}} \mapsto \langle \mathbb{E}[\mathbb{F}[\mathbf{0}] \parallel \mathbb{G}[R\{P/X\}]] \rangle_{\text{zs}} \quad \text{if } a = b \\
\langle b^{\Sigma}(X).R ; \pi \mid \mathbb{G}, \mathcal{R}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{in}} \mapsto \langle \mathbb{E}[\mathbb{G}[R\{P/X\}] \parallel \mathbb{F}[\mathbf{0}]] \rangle_{\text{zs}} \quad \text{if } a = b \\
\langle R ; \pi \mid \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{in}} \mapsto \langle \pi ; R^{\cup (\text{in}, a)} \mid \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{bin}} \quad \text{otherwise} \\
\langle (\text{outIn}, \Sigma) :: \pi ; R \mid \bullet, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{bin}} \mapsto \langle \bar{a}^{\Sigma} \langle P \rangle ; \pi \mid \mathbb{F}, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} \\
\langle (\text{inParL}, \Sigma) :: \pi ; R \mid \parallel Q :: \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{bin}} \mapsto \langle R \parallel^{\Sigma} Q ; \pi \mid \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{in}}
\end{array}$$

■ **Figure 4** Non-Deterministic Abstract Machine for HOcore.

We prove the correspondence between the two semantics in Appendix B.

► **Remark 3 (Synchronous communication).** For a synchronous calculus with an output $\bar{a}\langle P \rangle Q$, the rule `outIn` would pass the continuation Q as an argument of the input transition `in`. The continuation Q would then be plugged into \mathbb{F} in the axioms `inComL` and `inComR`.

► **Remark 4 (Left-first search).** After finding the communicating processes $P \parallel Q$, we could always go left (in P). When we find an output or input in P , we look for its complement in Q . A right-first search is also possible. We present the left-first zipper semantics and its machine in Appendix B; such an approach does not scale to $\text{HO}\pi$, as explained in Remark 22.

3.3 Non-Deterministic Abstract Machine

We derive the HOcore NDAM from its zipper semantics along the same principles as for the λ -calculus: each rule of the semantics corresponds to a forward step and a backward step, and when no forward step applies to a configuration, we switch to a backward configuration. The difference is in the normal-form annotations: in λ -calculus, to be a normal form w.r.t. $\xrightarrow{s, \mathbb{E}}_{\text{lam}}$ or $\xrightarrow{\mathbb{E}}_{\text{app}}$ does not depend on the arguments s and \mathbb{E} . In HOcore, being a normal form depends on some of the arguments in the input and output transitions.

For example, in a process $(\bar{a}\langle\mathbf{0}\rangle \parallel \bar{b}\langle\mathbf{0}\rangle) \parallel Q$, we may look into Q for an input on a or on b . If Q does not contain an input on a , then annotating it with the mode in would prevent from searching in Q for an input on b . We therefore include the name in the annotation, marking the root operator of Q with (in, a) , meaning that Q cannot do an input on a . If it also cannot do an input on b , then its root operator will be annotated with both (in, a) and (in, b) .

With outputs the problem is similar, but not completely symmetric. Let $P_{a,b} = \bar{a}\langle\mathbf{0}\rangle \parallel \bar{b}\langle\mathbf{0}\rangle$, and consider a process $(P_{a,b} \parallel Q) \parallel R$. We may try to find a communication between $P_{a,b}$ and Q first. If Q does not contain an input on a or b , then $P_{a,b}$ is a normal form w.r.t. the output search transition $\xrightarrow{\bullet, \mathcal{L}, \parallel R :: \bullet, Q}_{\text{out}}$, but a communication between $P_{a,b}$ and R is still possible. As a result, we annotate the root operator of $P_{a,b}$ with (out, Q) , meaning that the outputs of $P_{a,b}$ are not complemented by the inputs in Q . Such an annotation does not prevent trying to make $P_{a,b}$ and R communicate, which would correspond to the transition $\xrightarrow{\parallel Q :: \bullet, \mathcal{L}, \bullet, R}_{\text{out}}$.

As before, Σ ranges over annotation sets, and $|P|$ is the erasure of P , the annotated process with empty annotation sets. The syntax of annotations and processes is as follows.

$$\alpha ::= \text{par} \mid (\text{out}, |P|) \mid (\text{in}, a) \quad P, Q, R ::= X^\Sigma \mid \mathbf{0}^\Sigma \mid P \parallel^\Sigma Q \mid a^\Sigma(X).P \mid \bar{a}^\Sigma\langle P \rangle$$

Substitution and plugging are extended to annotated processes as expected. The definition of the machine is given in Figure 4. The process P in an annotation $(\text{out}, |P|)$ – as in the side conditions in the **par**-transitions – is erased, because normal forms are defined with respect to the zipper semantics transitions, where processes are not annotated. Apart from richer annotations, the definition of the machine follows the principles of Section 2.3. Note that the “otherwise” step for the input mode includes the operators that are not parsed in that mode, but also the inputs on a name distinct from a .

4 Derivation of the Abstract Machine

We show how to derive an abstract machine from a zipper semantics under some conditions. To this end, we specify zipper semantics as a transition system [22], a framework used to describe rule formats.

4.1 Zipper Semantics as a Transition System

Given an entity e , we write \tilde{e} for a possibly empty sequence (e_1, \dots, e_n) for some n . We assume a set \mathcal{S} of sorts ranged over by s , denoting the entities of the language (contexts, names, etc), and which includes the sort t of terms that are reduced. For each sort s , let \mathcal{O}_s be the signature of s , i.e., a set of operators, each having a typing $\tilde{s} \rightarrow s$. In particular, we let op range over the operators of the terms \mathcal{O}_t . We also assume a set \mathcal{F} of auxiliary functions that are used to build terms, like term substitution or context plugging, each of type $\tilde{s} \rightarrow t$.

For each s , we assume an infinite set \mathcal{V}_s of *rule variables*, denoted by v_s, w_s , or v, w if the sort does not matter. The set \mathfrak{E}_s of *rule entities* of sort s , ranged over by e_s, f_s (or e, f if we ignore the sort), are the entities built out of the signature \mathcal{O}_s extended with rule variables. We define \mathfrak{E}_s inductively so that $\mathcal{V}_s \subseteq \mathfrak{E}_s$, and for all $o \in \mathcal{O}_s$ of signature $(s_1, \dots, s_n) \rightarrow s$ and $(e_{s_i} \in \mathfrak{E}_{s_i})_{i \in 1 \dots n}$ for some n , we have $o(e_{s_1}, \dots, e_{s_n}) \in \mathfrak{E}_s$. A special case are term entities e_t , which can also be built out of auxiliary functions in \mathcal{F} . We write $\text{rv}(e_s)$ for the set of rule variables of e_s ; e_s is ground if $\text{rv}(e_s) = \emptyset$.

A rule substitution σ is a sort-respecting mapping from rule variables to rule entities. It should not be confused with the substitution $\{\cdot/\cdot\}$ which may exist for terms and is considered an auxiliary function in \mathcal{F} . We write $v\sigma$ for the application of σ to v , and $e\sigma$ – for its extension to rule entities, defined in the expected way. A ground entity e is an instance of e' if there exists σ such that $e'\sigma = e$.

$$\begin{array}{c}
\text{toy} \\
\frac{P \xrightarrow{a, \mathbb{E}} P'}{P \xrightarrow{\mathbb{E}} P'} \\
\text{outInL} \\
\frac{R \xrightarrow{\mathbb{F}[0] \parallel :: \mathbb{E}, a, P} \text{in} P'}{\bar{a}\langle P \rangle \xrightarrow{\mathbb{F}, \mathcal{L}, \mathbb{E}, R} \text{out} P'} \\
\text{choiceBad} \\
\frac{P \xrightarrow{\mathbb{E}} P'}{P + Q \xrightarrow{\mathbb{E}} P'} \\
\text{choiceOk} \\
\frac{P \xrightarrow{\mathbb{E}, Q :: \theta} P'}{P + Q \xrightarrow{\mathbb{E}, \theta} P'} \\
\text{rec} \\
\frac{P\{\mu X.P/X\} \xrightarrow{\mathbb{E}} P'}{\mu X.P \xrightarrow{\mathbb{E}} P'}
\end{array}$$

■ **Figure 5** Rules for variants of HOCore.

Given some rule variables \tilde{v} , we write $\mathcal{P}(\tilde{v})$ for a decidable predicate on \tilde{v} . We assume a set \mathcal{M} of modes, denoted by \mathfrak{m} , such that each mode is associated with a sequence $\tilde{s}_{\mathfrak{m}}$ giving the sorts of its arguments. The set \mathcal{M} includes the initial mode \mathfrak{zs} with no argument.

A *transition* is a predicate $e_i \xrightarrow{\tilde{e}}_{\mathfrak{m}} e_o$, where e_i and e_o are respectively the source and the target. We consider only three kinds of rule: inductive (whose names are ranged over with ρ), axiom, and initial, of the following respective shapes.

$$\begin{array}{ccc}
\frac{e_t \xrightarrow{\tilde{f}}_{\mathfrak{m}'} v_t \quad \mathcal{P}(\tilde{w}) \quad \rho}{op(\tilde{v}) \xrightarrow{\tilde{e}}_{\mathfrak{m}} v_t} & \frac{\mathcal{P}(\tilde{w})}{op(\tilde{v}) \xrightarrow{\tilde{e}}_{\mathfrak{m}} e_t} & \frac{v_t \xrightarrow{\tilde{f}}_{\mathfrak{m}} w_t}{v_t \rightarrow_{\mathfrak{zs}} w_t} \text{ init}
\end{array}$$

We extend the notion of set of rule variables \mathfrak{rv} and the application of a substitution to transitions and rules.

An inductive rule has only one premise, and may have side-conditions, represented by \mathcal{P} , on some of the variables \tilde{w} occurring in the rule. The modes \mathfrak{m} and \mathfrak{m}' may be distinct or not, and the sequences \tilde{e} and \tilde{f} should be rule entities of sorts respectively $\tilde{s}_{\mathfrak{m}}$ and $\tilde{s}_{\mathfrak{m}'}$. The sources and targets of the transitions are terms; in the conclusion, the source term is of the form $op(\tilde{v})$, enforcing that a rule can only pattern-match the head operator of the term. Both targets should be the same term variable, meaning that an inductive rule is simply passing along the result. Computation occurs in axioms, where the target can be any term.

An initial rule defines the initial mode \mathfrak{zs} . The source of the conclusion is a variable, so an initial rule does not perform any pattern-matching. An initial rule is just a means to set up the arguments of another mode \mathfrak{m} (such that $\mathfrak{m} \neq \mathfrak{zs}$). A zipper semantics is a triple $(\mathcal{S}, \mathcal{O}, \mathcal{R})$ where \mathcal{R} is a finite set of zipper rules with exactly one initial rule. The associated semantics on terms is defined by $\rightarrow_{\mathfrak{zs}}$.

4.2 Derivable Zipper Semantics

Not every zipper semantics can be turned into an NDAM. Some conditions have to be satisfied for the transformation to be possible and to ensure termination.

The first one is that the rules of the semantics must be constructive w.r.t. the machine, meaning that the entities in its premise are constructed from the ones in the conclusion. Indeed, the abstract machine searches for redexes with forward steps by going from the conclusion to the premise of a rule. As a result, a rule like **toy** in Figure 5 cannot be turned into a machine step, as the machine would have to guess the name a . We forbid such a rule by requiring that in each inductive rule of the zipper semantics, the rule variables of the premise are included in the rule variables of the conclusion.

► **Definition 5.** $\frac{e_t \xrightarrow{\tilde{f}}_{\mathfrak{m}'} v_t \quad \mathcal{P}(\tilde{w})}{op(\tilde{v}) \xrightarrow{\tilde{e}}_{\mathfrak{m}} v_t}$ is machine constructive if $\mathfrak{rv}(e_t \xrightarrow{\tilde{f}}_{\mathfrak{m}'} v_t) \cup \tilde{w} \subseteq \mathfrak{rv}(op(\tilde{v}) \xrightarrow{\tilde{e}}_{\mathfrak{m}} v_t)$.

The other constraint is that the rules must be *reversible* to allow for backtracking: it should be possible to reconstruct the entities in the conclusion from the ones in the premise. We say a rule is reversible if it cannot have two different instances with the same premise. For example, we could make the input search in HOCORE less verbose, by combining the contexts \mathbb{E} and \mathbb{F} in a single context, like in the rule `outInL` in Figure 5. In $\mathbb{E}[R \parallel \mathbb{F}[\mathbf{0}]]$, the input process is plugged into the context $\parallel \mathbb{F}[\mathbf{0}] :: \mathbb{E}$, that we build in rule `outInL`, instead of keeping \mathbb{E} and \mathbb{F} separate as in Figure 3. However, to unapply the rule `outInL`, we need to uniquely decompose a context as $\parallel \mathbb{F}[\mathbf{0}] :: \mathbb{E}$, which is not possible as soon as there are several occurrences of $\mathbf{0}$ in $\mathbb{F}[\mathbf{0}]$: the rule `outInL` is not reversible. We give a simple sufficient criterion for a rule to be reversible.

► **Lemma 6.** $\frac{e_t \xrightarrow{\tilde{f}}_{m'} v_t \quad \mathcal{P}(\tilde{w})}{op(\tilde{v}) \xrightarrow{\tilde{e}}_m v_t}$ is reversible if we have $\text{rv}(op(\tilde{v}) \xrightarrow{\tilde{e}}_m) \subseteq \text{rv}(e_t \xrightarrow{\tilde{f}}_{m'})$, and the auxiliary functions used to build the entities in e_t and \tilde{f} are injective.

The first condition states that the rules variables of the conclusion have to be included in those of the premise. Indeed, if we forget an entity between the conclusion and the premise, like Q in the rule for choice `choiceBad` in Figure 5, then we have no information to restore Q when backtracking. Instead, it should be kept in an extra argument of the zipper semantics, like the stack θ in the rule `choiceOk` in Figure 5. The stack θ is useful only for backtracking and not to define the semantics of the language, as it is simply thrown away when we apply an axiom. Any rule forgetting entities between its conclusion and premise can be made reversible using this principle [39].

Finally, we want the machine to always terminate when searching for a redex. Consider for instance the `rec` rule for a recursion operator in Figure 5. The corresponding machine would infinitely loop with $\mu X.X$. Indeed, the forward step of this rule changes focus from the source of the conclusion to the source of the premise, but these two terms are equal when $P = X$. To avoid this, we require the zipper semantics to be well-founded.

► **Definition 7.** A zipper semantics is well-founded if there exists a well-founded size ζ such that for all inductive rules $\frac{e'_t \xrightarrow{\tilde{f}}_{m'} v_t \quad \mathcal{P}(\tilde{w})}{e_t \xrightarrow{\tilde{e}}_m v_t}$, we have $\zeta(e'_t \xrightarrow{\tilde{f}}_{m'} v_t) < \zeta(e_t \xrightarrow{\tilde{e}}_m v_t)$.

In the calculi of this paper, each rule either focuses on a subterm or it changes mode (like in rule `outIn` in HOCORE). We therefore define an ordering on modes such that $m > m'$ if the derivation of m depends on m' ; e.g., we have `zs` > `app` > `lam` in λ -calculus, and `zs` > `par` > `out` > `in` in HOCORE and HO π . The size we consider is then the lexicographic ordering composed of the ordering on modes followed by the subterm ordering on the source term of the transition. This size works as long as we have no cyclic dependencies in modes and only congruence rules within each mode. It rules out unconstrained recursion, but we can still adapt it for guarded recursion, where the recursion variable occurs only after an input, as in $\mu X.a(Y).(X \parallel Y)$. In the premise of the `rec` rule, the μ operator itself becomes guarded, so the number of recursion operators at toplevel strictly decreases.

The semantics of Figures 1, 3, and 9 are machine constructive well-founded, and reversible (they satisfy Lemma 6). Henceforth, we assume the zipper semantics to be machine constructive, reversible, and well-founded.

$$\begin{array}{c}
\frac{e_t \xrightarrow{\tilde{f}}_{m'} v_t \quad \mathcal{P}(\tilde{w})}{op(\tilde{v}) \xrightarrow{\tilde{e}}_m v_t} \rho \qquad \langle op(v_\Sigma, \tilde{v}); \pi \mid \tilde{e} \rangle_m \mapsto \langle \|e_t\|; (\rho, v_\Sigma) :: \pi \mid \tilde{f} \rangle_{m'} \\
\qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \qquad \text{if } \phi(m', \tilde{f}) \notin \text{an}(\|e_t\|) \text{ and } \mathcal{P}(\tilde{w}) \\
\langle (\rho, v_\Sigma) :: \pi; \|e_t\| \mid \tilde{f} \rangle_{bm'} \mapsto \langle op(v_\Sigma, \tilde{v}); \pi \mid \tilde{e} \rangle_m \\
\\
\frac{v_t \xrightarrow{\tilde{f}}_m w_t \quad \text{init}}{v_t \rightarrow_{zs} w_t} \text{init} \qquad \langle v_t \rangle_{zs} \mapsto \langle v_t; \text{init} \mid \tilde{f} \rangle_m \\
\qquad \qquad \qquad \qquad \qquad \langle \text{init}; v_t \mid \tilde{f} \rangle_{bm} \mapsto \langle v_t \rangle_{nf} \\
\\
\frac{\mathcal{P}(\tilde{w})}{op(\tilde{v}) \xrightarrow{\tilde{e}}_m e_t} \qquad \langle op(v_\Sigma, \tilde{v}); \pi \mid \tilde{e} \rangle_m \mapsto \langle \|e_t\| \rangle_{zs} \text{ if } \mathcal{P}(\tilde{w})
\end{array}$$

■ **Figure 6** Forward and backward steps generated from a zipper semantics rule.

4.3 Machine Derivation

Annotations. The machine annotates terms which cannot do certain transitions, to forbid repeated tries which would lead to an infinite loop. The arguments of the transition may play a role in whether the term is a normal form or not: in HOCore an output $\bar{a}\langle P \rangle$ is a normal form w.r.t. the output transition $\xrightarrow{\mathbb{F}, \mathcal{S}, \mathbb{E}, R}_{\text{out}}$ if R cannot receive the message on a , so the annotation is $(\text{out}, |R|)$. Similarly an input $\xrightarrow{\mathbb{G}, \mathcal{S}, a, \mathbb{E}, \mathbb{F}}_{\text{in}}$ depends on the name a .

The arguments kept in the annotation are the ones either taking part in the reduction, like R in the output case, or in side-conditions, like a in the input case. Given a mode m with arguments \tilde{e} , its annotation $\phi(m, \tilde{e})$ is defined as (m, \tilde{f}) where $\tilde{f} \subseteq \tilde{e}$ are the arguments occurring either in side-conditions or source terms of the rules defining m . Repeating this for each mode of a zipper semantics, we define the *annotation function* ϕ of the semantics.

Annotated terms. Let $(\mathcal{S}, \mathcal{O}, \mathcal{R})$ be a zipper semantics with annotation function ϕ . We extend \mathcal{S} with the sort of annotation sets s_Σ , for which we assume the usual operators on sets. The machine is built on a signature \mathcal{A} which replaces the signature for terms \mathcal{O}_t with *annotated terms*, so that for all $op \in \mathcal{O}_t$ of type $(s_1, \dots, s_n) \rightarrow t$ for some n , we have a corresponding operator $op \in \mathcal{A}_t$ of type $(s_\Sigma, s_1, \dots, s_n) \rightarrow t$.

We let a range over annotated terms \mathfrak{A}_t , built out of \mathcal{A} , \mathcal{V}_t , and a single rule variable for annotation sets v_Σ : one variable is enough, as at most one annotation set occurs in a given machine step. Given an annotated term $a = op(e_\Sigma, \tilde{e})$, we write $\text{an}(a)$ for its annotation set e_Σ . Given a term $e_t \in \mathfrak{E}_t$, its annotated version, written $\|e_t\|$, is inductively defined so that $\|v_s\| = v_s$ and $\|op(\tilde{e})\| = op(v_\Sigma, \|\tilde{e}\|)$. Given an annotated term $a \in \mathfrak{A}_t$, its erasure $|a|$ produces a term with empty annotation sets, inductively defined so that $|v_s| = v_s$ and $|op(e_\Sigma, \tilde{e})| = op(\emptyset, \|\tilde{e}\|)$.

Machine steps. The syntax of rule stacks π is given by $\pi ::= \text{init} \mid (\rho, \Sigma) :: \pi$. We denote configurations $\langle a; \pi \mid \tilde{e} \rangle_m$ as *forward*, a special case being *initial* ones $\langle a \rangle_{zs}$. *Backward* configurations are of the form $\langle \pi; a \mid \tilde{e} \rangle_{bm}$ with *normal-form* ones $\langle a \rangle_{nf}$ as a subcase.

Figure 6 presents the forward and backward steps generated from an inductive rule ρ , an initial rule init , and an axiom. The forward step for an inductive rule goes from the conclusion to the premise, while the backward step goes in the opposite direction. Terms are extended with the rule variable for annotated sets v_Σ . The initial rule case is the same as the inductive one but simpler, as there is no side-condition: the annotated sets of the term v_t in $\langle v_t \rangle_{zs}$ are assumed to be empty. We can see that the annotations are erased after applying an axiom, as we end up with $\langle \|e_t\| \rangle_{zs}$. There is no backward step associated to axioms.

What remains are the switching steps when we realize that the current mode m does not apply to the term $op(v_\Sigma, \tilde{v})$ we reduce. These are the “otherwise” steps in Figures 2 and 4, which actually cover different cases. The first possibility is that op does not have a rule applying to it in the mode m . For such cases, we add a step

$$\langle op(v_\Sigma, \tilde{v}); \pi | \tilde{e} \rangle_m \mapsto \langle \pi; op(v_\Sigma \cup \{\phi(m, \tilde{e})\}, \tilde{v}) | \tilde{e} \rangle_{bm}$$

When going to a backward configuration, we extend the annotation set of the operator with the current annotation.

The other case is that no rule $\frac{e_t^i \xrightarrow{\tilde{f}_i}_{m_i} v_t}{op(\tilde{v}) \xrightarrow{\tilde{e}}_m v_t} \rho_i$ for op in the mode m applies, because either the premise or the side condition do not hold. If the machine has already checked that the premise fails, then e_t^i has been annotated with $\phi(m_i, \tilde{f}_i)$. The corresponding switching step is therefore

$$\langle op(v_\Sigma, \tilde{v}); \pi | \tilde{e} \rangle_m \mapsto \langle \pi; op(v_\Sigma \cup \{\phi(m, \tilde{e})\}, \tilde{v}) | \tilde{e} \rangle_{bm} \text{ if } \bigwedge_i \left(\phi(m_i, \tilde{f}_i) \in \text{an}(\|e_t^i\|) \vee \neg \mathcal{P}_i(\tilde{w}) \right)$$

Equivalence. The equivalence between the zipper semantics and its derived NDAM is proved in the research report [6]; we state here the main results. We let T (resp. A) range over (resp. annotated) ground terms. For all T , we write $\|T\|^\emptyset$ for the corresponding annotated term with empty annotations sets. For all A , we write $|A|$ for A where all annotations sets are made empty; there exists a unique T such that $|A| = \|T\|^\emptyset$. We call a *search path* a sequence of machine steps \mapsto^+ which does not go through an initial configuration. Search paths are finite, and result either in an initial or a normal-form configuration.

► **Theorem 8.** *For all T , there exists n such that any search path starting from $\langle \|T\|^\emptyset \rangle_{zs}$ is of size at most n . For all maximal search paths $\langle \|T\|^\emptyset \rangle_{zs} \mapsto^+ c$, either $c = \langle \|T'\|^\emptyset \rangle_{zs}$ for some T' , or $c = \langle A \rangle_{nf}$ for some A with $|A| = \|T\|^\emptyset$.*

We write $\vdash T \rightarrow_{zs} T'$ when there exists a zipper semantics derivation ended with $T \rightarrow_{zs} T'$. Search paths correspond to derivations in the following way.

- **Theorem 9.** *For all T, T' , and A ,*
 - $\vdash T \rightarrow_{zs} T'$ iff there exists a search path $\langle \|T\|^\emptyset \rangle_{zs} \mapsto^+ \langle \|T'\|^\emptyset \rangle_{zs}$;
 - T is a normal form iff there exists a search path $\langle \|T\|^\emptyset \rangle_{zs} \mapsto^+ \langle A \rangle_{nf}$ with $|A| = \|T\|^\emptyset$.

5 Related Work

The zipper semantics of the process calculi are inspired by complementary semantics [32], a format dedicated to bisimulation proofs. In both semantics, the derivation tree of two communicating processes is sequentialized. The difference is in the transition labels, which should be as minimal as possible in complementary semantics to keep the bisimulation proofs simple, while ours are detailed enough to be able to reconstruct the whole term.

Typical abstract machines for deterministic languages based on the λ -calculus are in refocused form [11]; such machines continue term decomposition from the contraction site. They have been shown to be uniformly derivable from the underlying reduction semantics by a refocusing method [7, 44], and the correctness of the derivation hinges on the unique decomposition property. NDAMs do not have this property, and after contracting a redex they completely reconstruct the term. An optimization similar to refocusing for non-deterministic languages appears more challenging in general. Another common feature of abstract machines for the λ -calculus is an efficient implementation of substitution with environments [8]. The

use of environments is orthogonal to the derivation of NDAMs: if the source zipper semantics uses environments, then so does its derived NDAM. We consider substitution-based zipper semantics in this paper because they are simpler than environment-based ones.

Process algebras have been implemented in various frameworks ranging from rewriting logic [45] to biological systems [35], including dedicated implementations and abstract machines [4, 16, 18–21, 24, 34, 37, 38, 46]. These implementations are ad-hoc and calculus-specific, and only some of them are complete [4, 19, 21, 24, 37, 38]. We believe we can handle most of these calculi in our framework in a uniform and complete way. However, the resulting implementation would be “single-threaded”, while the distribution of processes is a concern of previous machines [19], especially for calculi with localities [4, 20, 21, 24, 38]. Considering the many different models of distribution, making our machine distributed requires significantly more work, especially if we want to remain generic and complete.

Our use of backtracking evokes reversible calculi [9, 47], where one can revert communication steps, not necessarily in the order they were taken, as long as the causality between them is preserved. The concerns are different, though: in reversible calculi it is to keep enough information to track causality [30, 39], while here it is to control backtracking to avoid infinite searches. As a result, we store less information in machine configurations, but the annotations we use to prevent loops would not be typically needed in the other setting.

6 Conclusion

We present a generic design of abstract machines for non-deterministic languages. The machine looks for a redex in the term, making arbitrary choices when several paths are possible, and backtracks when it reaches a subterm which cannot reduce. The machine annotates such subterms to avoid trying them again, preventing infinite search. An NDAM is automatically derived from zipper semantics, a form of SOS in which the decomposition process of a term into a context and a redex is made explicit. The machine is sound and complete w.r.t. the zipper semantics. The derivation procedure has been implemented in OCaml [5]. The presented methodology is readily applicable to other non-deterministic calculi not shown in this paper, such as concurrent lambda calculi, with communication via channels or via futures [3, 17, 36].

An improvement of the current design would be to keep as many annotations as possible after reducing, in order to prune redundant search. Another optimization would be to find a way to manage annotations that would generically enable refocusing.

Finally, we would like to derive the zipper semantics from a more commonly used format, such as reduction semantics or SOS. An appropriate starting point should be able to express the different families of non-deterministic languages, such as concurrent λ -calculi or process calculi. A multi-hole context-based reduction semantics could be such a starting point.

References

- 1 Beniamino Accattoli and Giulio Guerrieri. Abstract machines for open call-by-value. *Sci. Comput. Program.*, 184, 2019.
- 2 Mads Sig Ager, Dariusz Biernacki, Olivier Danvy, and Jan Midtgaard. A functional correspondence between evaluators and abstract machines. In *Proceedings of the 5th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, 27-29 August 2003, Uppsala, Sweden*, pages 8–19. ACM, 2003.
- 3 Federico Aschieri, Agata Ciabattoni, and Francesco A. Genco. On the concurrent computational content of intermediate logics. *Theor. Comput. Sci.*, 813:375–409, 2020. doi:10.1016/j.tcs.2020.01.022.

- 4 Philippe Bidinger, Alan Schmitt, and Jean-Bernard Stefani. An abstract machine for the kell calculus. In Martin Steffen and Gianluigi Zavattaro, editors, *Formal Methods for Open Object-Based Distributed Systems, 7th IFIP WG 6.1 International Conference, FMOODS 2005, Athens, Greece, June 15-17, 2005, Proceedings*, volume 3535 of *Lecture Notes in Computer Science*, pages 31–46. Springer, 2005.
- 5 Małgorzata Biernacka, Dariusz Biernacki, Sergueï Lenglet, and Alan Schmitt. Non-deterministic abstract machines. Implementation available at <https://gitlab.inria.fr/skeletons/ndam/>.
- 6 Małgorzata Biernacka, Dariusz Biernacki, Sergueï Lenglet, and Alan Schmitt. Non-deterministic abstract machines. Technical Report 9475, Inria, 2022. available at <https://hal.inria.fr/hal-03545768>.
- 7 Malgorzata Biernacka, Witold Charatonik, and Klara Zielinska. Generalized refocusing: From hybrid strategies to abstract machines. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction, FSCD 2017, September 3-9, 2017, Oxford, UK*, volume 84 of *LIPICs*, pages 10:1–10:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
- 8 Malgorzata Biernacka and Olivier Danvy. A concrete framework for environment machines. *ACM Trans. Comput. Log.*, 9(1):6, 2007.
- 9 Vincent Danos and Jean Krivine. Reversible communicating systems. In Philippa Gardner and Nobuko Yoshida, editors, *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings*, volume 3170 of *Lecture Notes in Computer Science*, pages 292–307. Springer, 2004.
- 10 Olivier Danvy. From reduction-based to reduction-free normalization. In Pieter W. M. Koopman, Rinus Plasmeijer, and S. Doaitse Swierstra, editors, *Advanced Functional Programming, 6th International School, AFP 2008, Heijen, The Netherlands, May 2008, Revised Lectures*, volume 5832 of *Lecture Notes in Computer Science*, pages 66–164. Springer, 2008.
- 11 Olivier Danvy and Lasse R. Nielsen. Syntactic theories in practice. *Electron. Notes Theor. Comput. Sci.*, 59(4):358–374, 2001.
- 12 Matthias Felleisen, Robert Bruce Findler, and Matthew Flatt. *Semantics Engineering with PLT Redex*. The MIT Press, 2009.
- 13 Matthias Felleisen and Daniel P. Friedman. Control operators, the SECD-machine, and the λ -calculus. In Martin Wirsing, editor, *Formal Description of Programming Concepts - III: Proceedings of the IFIP TC 2/WG 2.2 Working Conference on Formal Description of Programming Concepts - III, Ebberup, Denmark, 25-28 August 1986*, pages 193–222. North-Holland, 1987.
- 14 Matthias Felleisen and Robert Hieb. The revised report on the syntactic theories of sequential control and state. *Theor. Comput. Sci.*, 103(2):235–271, 1992.
- 15 Matthias Felleisen and Robert Hieb. The revised report on the syntactic theories of sequential control and state. *Theor. Comput. Sci.*, 103(2):235–271, 1992.
- 16 Fabrice Le Fessant. *JoCaml: conception et implémentation d'un langage à agents mobiles*. PhD thesis, École polytechnique, 2001.
- 17 Cormac Flanagan and Matthias Felleisen. The semantics of future and an application. *J. Funct. Program.*, 9(1):1–31, 1999. doi:10.1017/s0956796899003329.
- 18 Cédric Fournet and Georges Gonthier. The reflexive CHAM and the join-calculus. In Hans-Juergen Boehm and Guy L. Steele Jr., editors, *Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, St. Petersburg Beach, Florida, USA, January 21-24, 1996*, pages 372–385. ACM Press, 1996.
- 19 Philippa Gardner, Cosimo Laneve, and Lucian Wischik. The fusion machine. In Lubos Brim, Petr Jancar, Mojmir Kretínský, and Antonín Kucera, editors, *CONCUR 2002 - Concurrency Theory, 13th International Conference, Brno, Czech Republic, August 20-23, 2002, Proceedings*, volume 2421 of *Lecture Notes in Computer Science*, pages 418–433. Springer, 2002.

- 20 Florence Germain, Marc Lacoste, and Jean-Bernard Stefani. An abstract machine for a higher-order distributed process calculus. *Electron. Notes Theor. Comput. Sci.*, 66(3):145–169, 2002.
- 21 Paola Giannini, Davide Sangiorgi, and Andrea Valente. Safe ambients: Abstract machine and distributed implementation. *Sci. Comput. Program.*, 59(3):209–249, 2006.
- 22 Jan Friso Groote and Frits W. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Inf. Comput.*, 100(2):202–260, 1992.
- 23 John Hannan and Dale Miller. From operational semantics for abstract machines. *Math. Struct. Comput. Sci.*, 2(4):415–459, 1992.
- 24 Daniel Hirschhoff, Damien Pous, and Davide Sangiorgi. An efficient abstract machine for safe ambients. *J. Log. Algebraic Methods Program.*, 71(2):114–149, 2007.
- 25 Gérard P. Huet. The zipper. *J. Funct. Program.*, 7(5):549–554, 1997.
- 26 Simon L. Peyton Jones. Implementing lazy functional languages on stock hardware: The spineless tagless G-machine. *J. Funct. Program.*, 2(2):127–202, 1992.
- 27 Claude Kirchner, Radu Kopetz, and Pierre-Etienne Moreau. Anti-pattern matching. In Rocco De Nicola, editor, *Programming Languages and Systems, 16th European Symposium on Programming, ESOP 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 4421 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2007.
- 28 Jean-Louis Krivine. A call-by-name lambda-calculus machine. *Higher-Order and Symbolic Computation*, 20(3):199–207, 2007.
- 29 Peter J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, 1964.
- 30 Ivan Lanese and Dorian Medić. A general approach to derive uncontrolled reversible semantics. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPICs*, pages 33:1–33:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 31 Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness and decidability of higher-order process calculi. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 145–155. IEEE Computer Society, 2008.
- 32 Sergueï Lenglet, Alan Schmitt, and Jean-Bernard Stefani. Characterizing contextual equivalence in calculi with passivation. *Inf. Comput.*, 209(11):1390–1433, 2011.
- 33 Xavier Leroy. The ZINC experiment: an economical implementation of the ML language. Technical report 117, INRIA, 1990.
- 34 Luís M. B. Lopes, Fernando M. A. Silva, and Vasco Thudichum Vasconcelos. A virtual machine for a process calculus. In Gopalan Nadathur, editor, *Principles and Practice of Declarative Programming, International Conference PPDP'99, Paris, France, September 29 - October 1, 1999, Proceedings*, volume 1702 of *Lecture Notes in Computer Science*, pages 244–260. Springer, 1999.
- 35 Urmi Majumder and John H. Reif. Design of a biomolecular device that executes process algebra. *Nat. Comput.*, 10(1):447–466, 2011.
- 36 Joachim Niehren, Jan Schwinghammer, and Gert Smolka. A concurrent lambda calculus with futures. *Theor. Comput. Sci.*, 364(3):338–356, 2006. doi:10.1016/j.tcs.2006.08.016.
- 37 Andrew Phillips and Luca Cardelli. A correct abstract machine for the stochastic pi-calculus. In *Concurrent Models in Molecular Biology*, 2004.
- 38 Andrew Phillips, Nobuko Yoshida, and Susan Eisenbach. A distributed abstract machine for boxed ambient calculi. In David A. Schmidt, editor, *Programming Languages and Systems, 13th European Symposium on Programming, ESOP 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 2986 of *Lecture Notes in Computer Science*, pages 155–170. Springer, 2004.
- 39 Iain C. C. Phillips and Irek Ulidowski. Reversing algebraic process calculi. *J. Log. Algebraic Methods Program.*, 73(1-2):70–96, 2007.

- 40 Gordon D. Plotkin. A structural approach to operational semantics. Technical Report FN-19, DAIMI, Department of Computer Science, Aarhus University, Aarhus, Denmark, September 1981.
- 41 Sylvain Salvati and Igor Walukiewicz. Krivine machines and higher-order schemes. *Inf. Comput.*, 239:340–355, 2014.
- 42 Davide Sangiorgi. Bisimulation in higher-order process calculi. In Ernst-Rüdiger Olderog, editor, *Programming Concepts, Methods and Calculi, Proceedings of the IFIP TC2/WG2.1/WG2.2/WG2.3 Working Conference on Programming Concepts, Methods and Calculi (PROCOMET '94) San Miniato, Italy, 6-10 June, 1994*, volume A-56 of *IFIP Transactions*, pages 207–224. North-Holland, 1994.
- 43 Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press, 2001.
- 44 Filip Sieczkowski, Malgorzata Biernacka, and Dariusz Biernacki. Automating derivations of abstract machines from reduction semantics: - A generic formalization of refocusing in Coq. In Jurriaan Hage and Marco T. Morazán, editors, *Implementation and Application of Functional Languages - 22nd International Symposium, IFL 2010, Alphen aan den Rijn, The Netherlands, September 1-3, 2010, Revised Selected Papers*, volume 6647 of *Lecture Notes in Computer Science*, pages 72–88. Springer, 2010.
- 45 Prasanna Thati, Koushik Sen, and Narciso Martí-Oliet. An executable specification of asynchronous pi-calculus semantics and may testing in maude 2.0. *Electron. Notes Theor. Comput. Sci.*, 71:261–281, 2002.
- 46 David Turner. *The Polymorphic Pi-calculus: Theory and Implementation*. PhD thesis, University of Edinburgh, 1995.
- 47 Irek Ulidowski, Ivan Lanese, Ulrik Pagh Schultz, and Carla Ferreira, editors. *Reversible Computation: Extending Horizons of Computing - Selected Results of the COST Action IC1405*, volume 12070 of *Lecture Notes in Computer Science*. Springer, 2020.

A Lambda-calculus

We prove the correspondence between the zipper and reduction semantics.

► **Lemma 10.** *For all $t \xrightarrow{s, \mathbb{E}}_{\text{lam}} t'$, we have $\mathbb{E}[t @ s] \rightarrow_{\text{rs}} t'$.*

For all $t \xrightarrow{\mathbb{E}}_{\text{app}} t'$, we have $\mathbb{E}[t] \rightarrow_{\text{rs}} t'$.

Proof. The first item is by definition, and the second one is proved by induction on the derivation of $t \xrightarrow{\mathbb{E}}_{\text{app}} t'$. The base case is $\text{app}\beta$, where we conclude using the first item.

For the recursive case, suppose we apply appL : we have $t @ s \xrightarrow{\mathbb{E}}_{\text{app}} t'$ because $t \xrightarrow{@ s :: \mathbb{E}}_{\text{app}} t'$. By induction, we have $(@ s :: \mathbb{E})[t] \rightarrow_{\text{rs}} t'$, i.e., $\mathbb{E}[t @ s] \rightarrow_{\text{rs}} t'$, as wished. The other cases are similar. ◀

► **Theorem 11.** *For all $t \rightarrow_{\text{zs}} t'$, we have $t \rightarrow_{\text{rs}} t'$.*

For completeness, we need $\mathbb{E}[(\lambda x.t'') @ s] \rightarrow_{\text{rs}} \mathbb{E}[t''\{s/x\}]$ implies $\mathbb{E}[(\lambda x.t'') @ s] \rightarrow_{\text{zs}} \mathbb{E}[t''\{s/x\}]$. First, we notice that $\lambda x.t'' \xrightarrow{s, \mathbb{E}}_{\text{lam}} \mathbb{E}[t''\{s/x\}]$ holds by definition of $\xrightarrow{s, \mathbb{E}}_{\text{lam}}$. With $\text{app}\beta$, we get $(\lambda x.t'') @ s \xrightarrow{\mathbb{E}}_{\text{app}} \mathbb{E}[t''\{s/x\}]$. To conclude, we use the following result.

► **Lemma 12.** *For all $t \xrightarrow{\mathbb{E}}_{\text{app}} t'$, we have $\mathbb{E}[t] \xrightarrow{\bullet}_{\text{app}} t'$.*

Proof. We proceed by induction on \mathbb{E} . There is nothing to prove for \bullet . If $\mathbb{E} = \lambda x :: \mathbb{E}'$, then $t \xrightarrow{\lambda x :: \mathbb{E}'}_{\text{app}} t'$ implies $\lambda x.t \xrightarrow{\mathbb{E}'}_{\text{app}} t'$ by $\text{app}\lambda$, from which we deduce $\mathbb{E}'[\lambda x.t] \xrightarrow{\bullet}_{\text{app}} t'$ by the induction hypothesis, i.e., $\mathbb{E}[t] \xrightarrow{\bullet}_{\text{app}} t'$, as wished. The proof is similar in the remaining cases. ◀

► **Theorem 13.** *For all $t \rightarrow_{\text{rs}} t'$, we have $t \rightarrow_{\text{zs}} t'$.*

$$\begin{array}{c}
\text{init} \\
\frac{P \xrightarrow{\bullet}_{\text{par}} P'}{P \rightarrow_{\text{zs}} P'} \\
\\
\text{parL} \\
\frac{P \parallel Q \xrightarrow{\mathbb{E}}_{\text{par}} P'}{P \parallel Q \xrightarrow{\mathbb{E}}_{\text{par}} P'} (s) \\
\\
\text{parCom} \\
\frac{P \xrightarrow{\bullet, \mathbb{E}, Q}_{\text{left}} P'}{P \parallel Q \xrightarrow{\mathbb{E}}_{\text{par}} P'} \\
\\
\text{leftParL} \\
\frac{P \parallel Q \xrightarrow{\mathbb{F}, \mathbb{E}, R}_{\text{left}} P'}{P \parallel Q \xrightarrow{\mathbb{F}, \mathbb{E}, R}_{\text{left}} P'} (s) \\
\\
\text{leftOut} \\
\frac{R \xrightarrow{\bullet, a, P, \mathbb{E}, \mathbb{F}}_{\text{in}} P'}{\bar{a}\langle P \rangle \xrightarrow{\mathbb{F}, \mathbb{E}, R}_{\text{left}} P'} \\
\\
\text{leftIn} \\
\frac{R \xrightarrow{\bullet, a, X, P, \mathbb{E}, \mathbb{F}}_{\text{out}} P'}{a(X).P \xrightarrow{\mathbb{F}, \mathbb{E}, R}_{\text{left}} P'} \\
\\
\text{inParL} \\
\frac{R \parallel Q \xrightarrow{\mathbb{G}, a, P, \mathbb{E}, \mathbb{F}}_{\text{in}} P'}{R \parallel Q \xrightarrow{\mathbb{G}, a, P, \mathbb{E}, \mathbb{F}}_{\text{in}} P'} (s) \\
\\
\text{inCom} \\
\frac{}{a(X).R \xrightarrow{\mathbb{G}, a, P, \mathbb{E}, \mathbb{F}}_{\text{in}} \mathbb{E}[\mathbb{F}[\mathbf{0}] \parallel \mathbb{G}[R\{P/X\}]]} \\
\\
\text{outParL} \\
\frac{R \parallel Q \xrightarrow{\mathbb{G}, a, X, P, \mathbb{E}, \mathbb{F}}_{\text{out}} P'}{R \parallel Q \xrightarrow{\mathbb{G}, a, X, P, \mathbb{E}, \mathbb{F}}_{\text{out}} P'} (s) \\
\\
\text{outCom} \\
\frac{}{\bar{a}\langle R \rangle \xrightarrow{\mathbb{G}, a, X, P, \mathbb{E}, \mathbb{F}}_{\text{out}} \mathbb{E}[\mathbb{F}[P\{R/X\}] \parallel \mathbb{G}[\mathbf{0}]]}
\end{array}$$

■ **Figure 7** Left-first Zipper Semantics for HOcore.

B HOcore

The output-first zipper semantics is equivalent to reduction semantics in the following way.

► **Lemma 14.** *For all $R \xrightarrow{\mathbb{G}, a, S, P, \mathbb{E}, \mathbb{F}}_{\text{in}} R'$, there exists R'' such that either we have $R' = \mathbb{E}[\mathbb{F}[\mathbf{0}] \parallel \mathbb{G}[R''\{P/X\}]]$ if $S = \mathcal{L}$ or $R' = \mathbb{E}[\mathbb{G}[R''\{P/X\}] \parallel \mathbb{F}[\mathbf{0}]]$ if $S = \mathcal{R}$.*

For all $P \xrightarrow{\mathbb{F}, S, \mathbb{E}, R}_{\text{out}} P'$, we have either $\mathbb{E}[\mathbb{F}[P] \parallel R] \rightarrow_{\text{rs}} P'$ if $S = \mathcal{L}$ or $\mathbb{E}[R \parallel \mathbb{F}[P]] \rightarrow_{\text{rs}} P'$ if $S = \mathcal{R}$.

For all $P \xrightarrow{\mathbb{E}}_{\text{par}} P'$, we have $\mathbb{E}[P] \rightarrow_{\text{rs}} P'$.

Each result is proved by induction on the zipper derivation.

► **Theorem 15.** *For all $P \rightarrow_{\text{zs}} P'$, we have $P \rightarrow_{\text{rs}} P'$.*

The reverse implication relies on the following results about contexts in zipper semantics.

► **Lemma 16.** *For all $R \xrightarrow{\mathbb{G}, S, a, P, \mathbb{E}, \mathbb{F}}_{\text{in}} R'$, we have $\mathbb{G}[R] \xrightarrow{\bullet, S, a, P, \mathbb{E}, \mathbb{F}}_{\text{in}} R'$.*

For all $P \xrightarrow{\mathbb{F}, S, \mathbb{E}, R}_{\text{out}} P'$, we have $\mathbb{F}[P] \xrightarrow{\bullet, S, \mathbb{E}, R}_{\text{out}} P'$.

For all $P \xrightarrow{\mathbb{E}}_{\text{par}} P'$, we have $\mathbb{E}[P] \xrightarrow{\bullet}_{\text{par}} P'$.

Suppose $R \rightarrow_{\text{rs}} R'$ with $R = \mathbb{E}[\mathbb{F}[\bar{a}\langle Q \rangle] \parallel \mathbb{G}[a(X).P]]$; the proof is similar in the symmetric case. We have $a(X).P \xrightarrow{\mathbb{G}, \mathcal{L}, a, Q, \mathbb{E}, \mathbb{F}}_{\text{in}} R'$, and by the first item of Lemma 16, we deduce $\mathbb{G}[a(X).P] \xrightarrow{\bullet, \mathcal{L}, a, Q, \mathbb{E}, \mathbb{F}}_{\text{in}} R'$. We get $\bar{a}\langle Q \rangle \xrightarrow{\mathbb{F}, \mathcal{L}, \mathbb{E}, \mathbb{G}[a(X).P]}_{\text{out}} R'$ by rule `outIn`, i.e., $\mathbb{F}[\bar{a}\langle Q \rangle] \xrightarrow{\bullet, \mathcal{L}, \mathbb{E}, \mathbb{G}[a(X).P]}_{\text{out}} R'$ with the second item. With rule `parOutL`, we obtain $\mathbb{F}[\bar{a}\langle Q \rangle] \parallel \mathbb{G}[a(X).P] \xrightarrow{\mathbb{E}}_{\text{par}} R'$, from which we can conclude using the last item.

► **Theorem 17.** *For all $P \rightarrow_{\text{rs}} P'$, we have $P \rightarrow_{\text{zs}} P'$.*

$$\begin{aligned}
\langle P \rangle_{\text{zs}} &\mapsto \langle P ; \text{init} \mid \bullet \rangle_{\text{par}} \\
\langle P \parallel^\Sigma Q ; \pi \mid \mathbb{E} \rangle_{\text{par}} &\mapsto \langle P ; (\text{parL}, \Sigma) :: \pi \mid \parallel Q :: \mathbb{E} \rangle_{\text{par}} && \text{if } \text{par} \notin \text{an}(P) \\
\langle P \parallel^\Sigma Q ; \pi \mid \mathbb{E} \rangle_{\text{par}} &\mapsto \langle Q ; (\text{parR}, \Sigma) :: \pi \mid P \parallel :: \mathbb{E} \rangle_{\text{par}} && \text{if } \text{par} \notin \text{an}(Q) \\
\langle P \parallel^\Sigma Q ; \pi \mid \mathbb{E} \rangle_{\text{par}} &\mapsto \langle P ; (\text{parCom}, \Sigma) :: \pi \mid \bullet, \mathbb{E}, Q \rangle_{\text{left}} && \text{if } (\text{left}, |Q|) \notin \text{an}(P) \\
\langle P ; \pi \mid \mathbb{E} \rangle_{\text{par}} &\mapsto \langle \pi ; P^{\cup \text{par}} \mid \mathbb{E} \rangle_{\text{bpar}} && \text{otherwise} \\
\langle \text{init} ; P \mid \bullet \rangle_{\text{bpar}} &\mapsto \langle P \rangle_{\text{nf}} \\
\langle (\text{parL}, \Sigma) :: \pi ; P \mid \parallel Q :: \mathbb{E} \rangle_{\text{bpar}} &\mapsto \langle P \parallel^\Sigma Q ; \pi \mid \mathbb{E} \rangle_{\text{par}} \\
\langle (\text{parR}, \Sigma) :: \pi ; Q \mid P \parallel :: \mathbb{E} \rangle_{\text{bpar}} &\mapsto \langle P \parallel^\Sigma Q ; \pi \mid \mathbb{E} \rangle_{\text{par}} \\
\langle P \parallel^\Sigma Q ; \pi \mid \mathbb{F}, \mathbb{E}, R \rangle_{\text{left}} &\mapsto \langle P ; (\text{leftParL}, \Sigma) :: \pi \mid \parallel Q :: \mathbb{F}, \mathbb{E}, R \rangle_{\text{left}} && \text{if } (\text{left}, |R|) \notin \text{an}(P) \\
\langle P \parallel^\Sigma Q ; \pi \mid \mathbb{F}, \mathbb{E}, R \rangle_{\text{left}} &\mapsto \langle Q ; (\text{leftParR}, \Sigma) :: \pi \mid P \parallel :: \mathbb{F}, \mathbb{E}, R \rangle_{\text{left}} && \text{if } (\text{left}, |R|) \notin \text{an}(Q) \\
\langle \bar{a}^\Sigma \langle P \rangle ; \pi \mid \mathbb{F}, \mathbb{E}, R \rangle_{\text{left}} &\mapsto \langle R ; (\text{leftOut}, \Sigma) :: \pi \mid \bullet, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{in}} && \text{if } (\text{in}, a) \notin \text{an}(R) \\
\langle a^\Sigma(X).P ; \pi \mid \mathbb{F}, \mathbb{E}, R \rangle_{\text{left}} &\mapsto \langle R ; (\text{leftIn}, \Sigma) :: \pi \mid \bullet, a, \mathbb{F}, \mathbb{E}, X, P \rangle_{\text{out}} && \text{if } (\text{out}, a) \notin \text{an}(R) \\
\langle P ; \pi \mid \mathbb{F}, \mathbb{E}, R \rangle_{\text{left}} &\mapsto \langle \pi ; P^{\cup (\text{left}, R)} \mid \mathbb{F}, \mathbb{E}, R \rangle_{\text{bleft}} && \text{otherwise} \\
\langle (\text{parCom}, \Sigma) :: \pi ; P \mid \bullet, \mathbb{E}, Q \rangle_{\text{bleft}} &\mapsto \langle P \parallel^\Sigma Q ; \pi \mid \mathbb{E} \rangle_{\text{par}} \\
\langle (\text{leftParL}, \Sigma) :: \pi ; P \mid \parallel Q :: \mathbb{F}, \mathbb{E}, R \rangle_{\text{bleft}} &\mapsto \langle P \parallel^\Sigma Q ; \pi \mid \mathbb{F}, \mathbb{E}, R \rangle_{\text{left}} \\
\langle (\text{leftParR}, \Sigma) :: \pi ; Q \mid P \parallel :: \mathbb{F}, \mathbb{E}, R \rangle_{\text{bleft}} &\mapsto \langle P \parallel^\Sigma Q ; \pi \mid \mathbb{F}, \mathbb{E}, R \rangle_{\text{left}} \\
\langle R \parallel^\Sigma Q ; \pi \mid \mathbb{G}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{in}} &\mapsto \langle R ; (\text{inParL}, \Sigma) :: \pi \mid \parallel Q :: \mathbb{G}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{in}} && \text{if } (\text{in}, a) \notin \text{an}(R) \\
\langle R \parallel^\Sigma Q ; \pi \mid \mathbb{G}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{in}} &\mapsto \langle Q ; (\text{inParR}, \Sigma) :: \pi \mid R \parallel :: \mathbb{G}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{in}} && \text{if } (\text{in}, a) \notin \text{an}(Q) \\
\langle a^\Sigma(X).R ; \pi \mid \mathbb{G}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{in}} &\mapsto \langle \mathbb{E}[\mathbb{F}[\mathbf{0}]] \parallel \mathbb{G}[R\{P/X\}] \rangle_{\text{zs}} \\
\langle R ; \pi \mid \mathbb{G}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{in}} &\mapsto \langle \pi ; R^{\cup (\text{in}, a)} \mid \mathbb{G}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{bin}} && \text{otherwise} \\
\langle (\text{leftOut}, \Sigma) :: \pi ; R \mid \bullet, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{bin}} &\mapsto \langle \bar{a}^\Sigma \langle P \rangle ; \pi \mid \mathbb{F}, \mathbb{E}, R \rangle_{\text{left}} \\
\langle (\text{inParL}, \Sigma) :: \pi ; R \mid \parallel Q :: \mathbb{G}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{bin}} &\mapsto \langle R \parallel^\Sigma Q ; \pi \mid \mathbb{G}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{in}} \\
\langle (\text{inParR}, \Sigma) :: \pi ; Q \mid R \parallel :: \mathbb{G}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{bin}} &\mapsto \langle R \parallel^\Sigma Q ; \pi \mid \mathbb{G}, a, P, \mathbb{E}, \mathbb{F} \rangle_{\text{in}}
\end{aligned}$$

■ **Figure 8** Left-first NDAM for HOcore.

The left-first semantics for HOcore is given in Figure 7. The `par` transition is going through the process to find the parallel composition at the root of the communication redex, building the context \mathbb{E} surrounding the redex at the same time. Finding the parallel composition triggers the $\xrightarrow{\mathbb{F}, \mathbb{E}, R}_{\text{left}}$ transition, which looks for an input or an output in the process on the left, while building the context \mathbb{F} and remembering \mathbb{E} and the process on the right R . If we find an output, we look for an input on the same name in R using $\xrightarrow{\mathbb{G}, a, P, \mathbb{E}, \mathbb{F}}_{\text{in}}$ (rule `leftOut`), otherwise we look for an output using $\xrightarrow{\mathbb{G}, a, X, P, \mathbb{E}, \mathbb{F}}_{\text{out}}$ (rule `leftIn`). These two transitions are building the context \mathbb{G} and use the remaining arguments to compute the results of the communication (rules `inCom` and `outCom`).

The corresponding NDAM is in Figure 8, except for the `out` and `bout` modes, which are symmetric to the `in` and `bin` modes.

C HO π

We present the zipper semantics of HO π , an extension of HOcore with name restriction. The main difficulty is that the evaluation contexts surrounding the communicating processes can be themselves modified by the reduction.

C.1 Syntax and Semantics

We add name restriction to HOcore processes and frames.

$$P, Q, R ::= \dots \mid \nu a.P \quad \mathfrak{F} ::= \dots \mid \nu a$$

To remain close to HOcore, the calculus of this section is asynchronous: outputs $\bar{a}\langle P \rangle$ do not have a continuation, unlike the original HO π [42]. Adding continuations would not be an issue as pointed out in Remark 3.

The scope of a in $\nu a.P$ is restricted to P , so that a communication on a is possible inside P only. For instance, the process $a(X).X \parallel \nu a.\bar{a}\langle \mathbf{0} \rangle$ cannot reduce, because the name a is restricted to the process on the right. In general, a process $\mathbb{E}[\bar{a}\langle P \rangle]$ or $\mathbb{E}[a(X).P]$ cannot communicate on a if \mathbb{E} captures a . To check this, we compute the set of names bound by \mathbb{E} , written $\text{bn}(\mathbb{E})$, as follows.

$$\begin{aligned} \text{bn}(\bullet) &\triangleq \emptyset & \text{bn}(\parallel P :: \mathbb{E}) &\triangleq \text{bn}(\mathbb{E}) \\ \text{bn}(\nu a :: \mathbb{E}) &\triangleq \{a\} \cup \text{bn}(\mathbb{E}) & \text{bn}(P \parallel :: \mathbb{E}) &\triangleq \text{bn}(\mathbb{E}) \end{aligned}$$

Name restriction does not forbid the communication on unrestricted names, but the scope of restricted names has to be enlarged to prevent them from escaping their delimiter. For example, we have

$$b(X).(X \parallel \bar{c}\langle \mathbf{0} \rangle) \parallel \nu a.(\bar{b}\langle a(Y).Y \rangle \parallel \bar{a}\langle \mathbf{0} \rangle) \rightarrow_{rs} \nu a.(a(Y).Y \parallel \bar{c}\langle \mathbf{0} \rangle \parallel \mathbf{0} \parallel \bar{a}\langle \mathbf{0} \rangle)$$

The scope of a has been extended to include the receiving process on b . This phenomenon is known as *scope extrusion*. To reflect it at the level of contexts, we define an operation $\text{extr}(\mathbb{E})$ which returns a pair of contexts $(\mathbb{E}_1, \mathbb{E}_2)$ such that \mathbb{E}_2 contains the binding frames, while \mathbb{E}_1 contains the remaining frames. We assume free names to be distinct from bound names using α -conversion if necessary, to avoid capture during extrusion.

$$\begin{aligned} \text{extr}(\bullet) &\triangleq (\bullet, \bullet) & \frac{\text{extr}(\mathbb{E}) = (\mathbb{E}_1, \mathbb{E}_2)}{\text{extr}(\nu a :: \mathbb{E}) \triangleq (\mathbb{E}_1, \nu a :: \mathbb{E}_2)} & \frac{\text{extr}(\mathbb{E}) = (\mathbb{E}_1, \mathbb{E}_2)}{\text{extr}(\parallel P :: \mathbb{E}) \triangleq (\parallel P :: \mathbb{E}_1, \mathbb{E}_2)} \\ & & \frac{\text{extr}(\mathbb{E}) = (\mathbb{E}_1, \mathbb{E}_2)}{\text{extr}(P \parallel :: \mathbb{E}) \triangleq (P \parallel :: \mathbb{E}_1, \mathbb{E}_2)} \end{aligned}$$

We define the reduction semantics \rightarrow_{rs} of HO π as follows, assuming $a \notin \text{bn}(\mathbb{F}) \cup \text{bn}(\mathbb{G})$ and $\text{extr}(\mathbb{F}) = (\mathbb{F}_1, \mathbb{F}_2)$.

$$\begin{aligned} \mathbb{E}[\mathbb{F}[\bar{a}\langle Q \rangle] \parallel \mathbb{G}[a(X).P]] &\rightarrow_{rs} \mathbb{E}[\mathbb{F}_2[\mathbb{F}_1[\mathbf{0}] \parallel \mathbb{G}[P\{Q/X\}]]] \\ \mathbb{E}[\mathbb{G}[a(X).P] \parallel \mathbb{F}[\bar{a}\langle Q \rangle]] &\rightarrow_{rs} \mathbb{E}[\mathbb{F}_2[\mathbb{G}[P\{Q/X\}] \parallel \mathbb{F}_1[\mathbf{0}]]] \end{aligned}$$

$$\begin{array}{c}
\text{init} \\
\frac{P \xrightarrow{\bullet} \text{par} P'}{P \rightarrow_{\text{zs}} P'} \\
\\
\text{parNu} \\
\frac{P \xrightarrow{\nu a :: \mathbb{E}} \text{par} P'}{\nu a.P \xrightarrow{\mathbb{E}} \text{par} P'} \\
\\
\text{parL} \\
\frac{P \parallel Q \xrightarrow{\mathbb{E}} \text{par} P'}{P \parallel Q \xrightarrow{\mathbb{E}} \text{par} P'} \quad (s) \\
\\
\text{parOutL} \\
\frac{P \xrightarrow{\bullet, \bullet, \mathcal{L}, \mathbb{E}, Q} \text{out} P'}{P \parallel Q \xrightarrow{\mathbb{E}} \text{par} P'} \\
\\
\text{parOutR} \\
\frac{Q \xrightarrow{\bullet, \bullet, \mathcal{R}, \mathbb{E}, P} \text{out} P'}{P \parallel Q \xrightarrow{\mathbb{E}} \text{par} P'} \\
\\
\text{outParL} \\
\frac{P \parallel Q \xrightarrow{\mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R} \text{out} P'}{P \parallel Q \xrightarrow{\mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R} \text{out} P'} \quad (s) \\
\\
\text{outNu} \\
\frac{P \xrightarrow{\mathbb{F}_1, \nu b :: \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R} \text{out} P'}{\nu b.P \xrightarrow{\mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R} \text{out} P'} \\
\\
\text{outIn} \\
\frac{R \xrightarrow{\bullet, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2} \text{in} P' \quad a \notin \text{bn}(\mathbb{F}_2)}{\bar{a}\langle P \rangle \xrightarrow{\mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R} \text{out} P'} \\
\\
\text{inParL} \\
\frac{R \parallel Q \xrightarrow{\mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2} \text{in} P'}{R \parallel Q \xrightarrow{\mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2} \text{in} P'} \quad (s) \\
\\
\text{inNu} \\
\frac{R \xrightarrow{\nu b :: \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2} \text{in} P' \quad a \neq b}{\nu b.R \xrightarrow{\mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2} \text{in} P'} \\
\\
\text{inComL} \\
\frac{a = b}{b(X).R \xrightarrow{\mathbb{G}, \mathcal{L}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2} \text{in} \mathbb{E}[\mathbb{F}_2[\mathbb{F}_1[\mathbf{0}] \parallel \mathbb{G}[R\{P/X\}]]]} \quad (s)
\end{array}$$

■ **Figure 9** Zipper Semantics for $\text{HO}\pi$.

C.2 Zipper Semantics and NDAM

We present the zipper semantics of $\text{HO}\pi$ in Figure 9. The out and in transitions differ from HOcore as they carry two contexts \mathbb{F}_1 and \mathbb{F}_2 : as in the reduction semantics, \mathbb{F}_1 collects the parallel compositions (rules **outParL** and **outParR**) while \mathbb{F}_2 collects the name restrictions (rule **outNu**).

Checking that the name a on which the communication happens is not captured by \mathbb{F}_2 or \mathbb{G} is not done the same way in the out and in transitions, because the transitions themselves are not completely symmetric. In the input transition, we already know the name a , so we simply verify that the names bound by \mathbb{G} differ from a on the fly in rule **inNu**. We cannot do the same in rule **outNu**, because we do yet not know a at this point. We know a when we find the output (rule **outIn**), so we check here that \mathbb{F}_2 does not capture it.

We first prove that zipper semantics implies reduction semantics.

► **Lemma 18.** *For all transitions $R \xrightarrow{\mathbb{G}, a, \mathcal{S}, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2} \text{in} R'$, there exists R'' such that $R' = \mathbb{E}[\mathbb{F}_2[\mathbb{F}_1[\mathbf{0}] \parallel \mathbb{G}[R''\{P/X\}]]]$ if $\mathcal{S} = \mathcal{L}$ and $R' = \mathbb{E}[\mathbb{F}_2[\mathbb{G}[R''\{P/X\}] \parallel \mathbb{F}_1[\mathbf{0}]]]$ if $\mathcal{S} = \mathcal{R}$.*

For all transitions $P \xrightarrow{\mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R} \text{out} P'$ and \mathbb{F} such that $\text{extr}(\mathbb{F}) = (\mathbb{F}_1, \mathbb{F}_2)$, we have $\mathbb{E}[\mathbb{F}[P] \parallel R] \rightarrow_{\text{rs}} P'$ if $\mathcal{S} = \mathcal{L}$ and $\mathbb{E}[R \parallel \mathbb{F}[P]] \rightarrow_{\text{rs}} P'$ if $\mathcal{S} = \mathcal{R}$.

For all $P \xrightarrow{\mathbb{E}} \text{par} P'$, we have $\mathbb{E}[P] \rightarrow_{\text{rs}} P'$.

$$\begin{aligned}
\langle P \rangle_{zs} &\mapsto \langle P; \text{init} \mid \bullet \rangle_{\text{par}} \\
\langle P \parallel^\Sigma Q; \pi \mid \mathbb{E} \rangle_{\text{par}} &\mapsto \langle P; (\text{parL}, \Sigma) :: \pi \mid \parallel Q :: \mathbb{E} \rangle_{\text{par}} && \text{if } \text{par} \notin \text{an}(P) \\
\langle P \parallel^\Sigma Q; \pi \mid \mathbb{E} \rangle_{\text{par}} &\mapsto \langle Q; (\text{parR}, \Sigma) :: \pi \mid P \parallel :: \mathbb{E} \rangle_{\text{par}} && \text{if } \text{par} \notin \text{an}(Q) \\
\langle \nu^\Sigma a.P; \pi \mid \mathbb{E} \rangle_{\text{par}} &\mapsto \langle P; (\text{parNu}, \Sigma) :: \pi \mid \nu a :: \mathbb{E} \rangle_{\text{par}} && \text{if } \text{par} \notin \text{an}(P) \\
\langle P \parallel^\Sigma Q; \pi \mid \mathbb{E} \rangle_{\text{par}} &\mapsto \langle P; (\text{parOutL}, \Sigma) :: \pi \mid \bullet, \bullet, \mathcal{L}, \mathbb{E}, Q \rangle_{\text{out}} && \text{if } (\text{out}, |Q|, \bullet) \notin \text{an}(P) \\
\langle P \parallel^\Sigma Q; \pi \mid \mathbb{E} \rangle_{\text{par}} &\mapsto \langle Q; (\text{parOutR}, \Sigma) :: \pi \mid \bullet, \bullet, \mathcal{R}, \mathbb{E}, P \rangle_{\text{out}} && \text{if } (\text{out}, |P|, \bullet) \notin \text{an}(Q) \\
\langle P; \pi \mid \mathbb{E} \rangle_{\text{par}} &\mapsto \langle \pi; P^{\cup \text{par}} \mid \mathbb{E} \rangle_{\text{bpar}} && \text{otherwise} \\
\langle \text{init}; P \mid \bullet \rangle_{\text{bpar}} &\mapsto \langle P \rangle_{\text{nf}} \\
\langle (\text{parL}, \Sigma) :: \pi; P \mid \parallel Q :: \mathbb{E} \rangle_{\text{bpar}} &\mapsto \langle P \parallel^\Sigma Q; \pi \mid \mathbb{E} \rangle_{\text{par}} \\
\langle (\text{parR}, \Sigma) :: \pi; Q \mid P \parallel :: \mathbb{E} \rangle_{\text{bpar}} &\mapsto \langle P \parallel^\Sigma Q; \pi \mid \mathbb{E} \rangle_{\text{par}} \\
\langle (\text{parNu}, \Sigma) :: \pi; P \mid \nu a :: \mathbb{E} \rangle_{\text{bpar}} &\mapsto \langle \nu^\Sigma a.P; \pi \mid \mathbb{E} \rangle_{\text{par}} \\
\langle P \parallel^\Sigma Q; \pi \mid \mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} &\mapsto \langle P; (\text{outParL}, \Sigma) :: \pi \mid \parallel Q :: \mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} && \text{if } (\text{out}, |R|, \mathbb{F}_2) \notin \text{an}(P) \\
\langle P \parallel^\Sigma Q; \pi \mid \mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} &\mapsto \langle Q; (\text{outParR}, \Sigma) :: \pi \mid P \parallel :: \mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} && \text{if } (\text{out}, |R|, \mathbb{F}_2) \notin \text{an}(Q) \\
\langle \nu^\Sigma a.P; \pi \mid \mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} &\mapsto \langle P; (\text{outNu}, \Sigma) :: \pi \mid \mathbb{F}_1, \nu a :: \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} && \text{if } (\text{out}, |R|, \nu a. \mathbb{F}_2) \notin \text{an}(P) \\
\langle \bar{a}^\Sigma \langle P \rangle; \pi \mid \mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} &\mapsto \langle R; (\text{outIn}, \Sigma) :: \pi \mid \bullet, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{in}} && \text{if } (\text{in}, a) \notin \text{an}(R), a \notin \text{bn}(\mathbb{F}_2) \\
\langle P; \pi \mid \mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} &\mapsto \langle \pi; P^{\cup (\text{out}, |R|)} \mid \mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R \rangle_{\text{bout}} && \text{otherwise} \\
\langle (\text{parOutL}, \Sigma) :: \pi; P \mid \bullet, \bullet, \mathcal{L}, \mathbb{E}, Q \rangle_{\text{bout}} &\mapsto \langle P \parallel^\Sigma Q; \pi \mid \mathbb{E} \rangle_{\text{par}} \\
\langle (\text{parOutR}, \Sigma) :: \pi; Q \mid \bullet, \bullet, \mathcal{R}, \mathbb{E}, P \rangle_{\text{bout}} &\mapsto \langle P \parallel^\Sigma Q; \pi \mid \mathbb{E} \rangle_{\text{par}} \\
\langle (\text{outParL}, \Sigma) :: \pi; P \mid \parallel Q :: \mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R \rangle_{\text{bout}} &\mapsto \langle P \parallel^\Sigma Q; \pi \mid \mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} \\
\langle (\text{outParR}, \Sigma) :: \pi; Q \mid P \parallel :: \mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R \rangle_{\text{bout}} &\mapsto \langle P \parallel^\Sigma Q; \pi \mid \mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} \\
\langle (\text{outNu}, \Sigma) :: \pi; P \mid \mathbb{F}_1, \nu a :: \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R \rangle_{\text{bout}} &\mapsto \langle \nu^\Sigma a.P; \pi \mid \mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}}
\end{aligned}$$

■ **Figure 10** Non-Deterministic Abstract Machine for HO π - parallel and output modes.

Proof. We sketch the proof of the second item, the others are easy. The proof is by induction on the derivation of the `out` transition. We assume $\mathcal{S} = \mathcal{L}$, the case $\mathcal{S} = \mathcal{R}$ is similar. In the base case (rule `outIn`), we have $P = \bar{a} \langle P'' \rangle$ and $R \xrightarrow{\bullet, a, \mathcal{S}, P'', \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2}_{\text{in}} P'$, which implies $P' = \mathbb{E}[\mathbb{F}_2[\mathbb{F}_1[\mathbf{0}] \parallel \mathbb{G}[R''\{P''/X\}]]]$ for some R'' by the first item.

Suppose we are in the case of rule `outNu`, and let \mathbb{F} such that $\text{extr}(\mathbb{F}) = (\mathbb{F}_1, \mathbb{F}_2)$. Then $P = \nu a.P''$ and $P'' \xrightarrow{\mathbb{F}_1, \nu a. \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R}_{\text{out}} P'$. By induction, for all \mathbb{F}' such that $\text{extr}(\mathbb{F}') = (\mathbb{F}_1, \nu a :: \mathbb{F}_2)$, we have $\mathbb{E}[\mathbb{F}'[P'']] \rightarrow_{\text{rs}} P'$. But since $\text{extr}(\mathbb{F}) = (\mathbb{F}_1, \mathbb{F}_2)$, we

$$\begin{aligned}
& \langle R \parallel^\Sigma Q; \pi \mid \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{in}} \mapsto \langle R; (\text{inParL}, \Sigma) :: \pi \parallel Q :: \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{in}} \\
& \quad \text{if } (\text{in}, a) \notin \text{an}(R) \\
& \langle R \parallel^\Sigma Q; \pi \mid \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{in}} \mapsto \langle Q; (\text{inParR}, \Sigma) :: \pi \mid R \parallel :: \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{in}} \\
& \quad \text{if } (\text{in}, a) \notin \text{an}(Q) \\
& \langle \nu^\Sigma b.R; \pi \mid \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{in}} \mapsto \langle R; (\text{inNu}, \Sigma) :: \pi \mid \nu b :: \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{in}} \\
& \quad \text{if } (\text{in}, a) \notin \text{an}(R), b \neq a \\
& \langle b^\Sigma(X).R; \pi \mid \mathbb{G}, \mathcal{L}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{in}} \mapsto \langle \mathbb{E}[\mathbb{F}_2[\mathbb{F}_1[\mathbf{0}] \parallel \mathbb{G}[R\{P/X\}]]] \rangle_{\text{zs}} \\
& \quad \text{if } a = b \\
& \langle b^\Sigma(X).R; \pi \mid \mathbb{G}, \mathcal{R}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{in}} \mapsto \langle \mathbb{E}[\mathbb{F}_2[\mathbb{G}[R\{P/X\}] \parallel \mathbb{F}_1[\mathbf{0}]]] \rangle_{\text{zs}} \\
& \quad \text{if } a = b \\
& \langle R; \pi \mid \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{in}} \mapsto \langle \pi; R^{\cup(\text{in}, a)} \mid \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{bin}} \\
& \quad \text{otherwise} \\
& \langle (\text{outIn}, \Sigma) :: \pi; R \mid \bullet, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{bin}} \mapsto \langle \bar{a}^\Sigma \langle P \rangle; \pi \mid \mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R \rangle_{\text{out}} \\
& \langle (\text{inParL}, \Sigma) :: \pi; R \parallel Q :: \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{bin}} \mapsto \langle R \parallel^\Sigma Q; \pi \mid \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{in}} \\
& \langle (\text{inParR}, \Sigma) :: \pi; Q \parallel R \parallel :: \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{bin}} \mapsto \langle R \parallel^\Sigma Q; \pi \mid \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{in}} \\
& \langle (\text{inNu}, \Sigma) :: \pi; R \mid \nu b :: \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{bin}} \mapsto \langle \nu^\Sigma b.R; \pi \mid \mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2 \rangle_{\text{in}}
\end{aligned}$$

■ **Figure 11** Non-Deterministic Abstract Machine for HO π - input mode.

have $\text{extr}(\nu a :: \mathbb{F}) = (\mathbb{F}_1, \nu a :: \mathbb{F}_2)$ by definition, so by the induction hypothesis, we obtain $\mathbb{E}[(\nu a.\mathbb{F})[P''] \parallel R] \rightarrow_{\text{rs}} P'$. This is the same as $\mathbb{E}[\mathbb{F}[\nu a.P''] \parallel R] \rightarrow_{\text{rs}} P'$, but $\nu a.P'' = P$, so we get the expected result. The cases of rules `outParL` and `outParR` are similar. ◀

► **Theorem 19.** *For all $P \rightarrow_{\text{zs}} P'$, we have $P \rightarrow_{\text{rs}} P'$.*

The proof of the reverse implication follows the same strategy as in HOcore, using the following result.

► **Lemma 20.** *For all $R \xrightarrow{\mathbb{G}, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2}_{\text{in}} R'$, we have $\mathbb{G}[R] \xrightarrow{\bullet, \mathcal{S}, a, P, \mathbb{E}, \mathbb{F}_1, \mathbb{F}_2}_{\text{in}} R'$.
For all $P \xrightarrow{\mathbb{F}_1, \mathbb{F}_2, \mathcal{S}, \mathbb{E}, R}_{\text{out}} P'$ and \mathbb{F} such that $\text{extr}(\mathbb{F}) = (\mathbb{F}_1, \mathbb{F}_2)$, we have $\mathbb{F}[P] \xrightarrow{\bullet, \bullet, \mathcal{S}, \mathbb{E}, R}_{\text{out}} P'$.
For all $P \xrightarrow{\mathbb{E}}_{\text{par}} P'$, we have $\mathbb{E}[P] \xrightarrow{\bullet}_{\text{par}} P'$.*

► **Theorem 21.** *For all $P \rightarrow_{\text{rs}} P'$, we have $P \rightarrow_{\text{zs}} P'$.*

► **Remark 22.** The zipper semantics for HO π cannot be written in the left-first style (Remark 4) because of scope extrusion. After finding the communicating processes $P \parallel Q$, we search for an output or input in P . Because we do not know the operator in advance, we do not know if we should decompose the context surrounding it to account for scope extrusion.

While writing the zipper semantics for HO π requires some care, the corresponding NDAM is as expected (cf. Figures 10 and 11). A difference with HOcore is the side-conditions in the `outIn` and `inNu` rules, which are added to the step. If the side-condition is not met, the “otherwise” step applies and we switch to the backward mode `bout`. The side-condition also makes the output mode annotation become $(\text{out}, |R|, \mathbb{F}_2)$: a process $\bar{a} \langle P \rangle$ is a normal form w.r.t. output if \mathbb{F}_2 captures a , so being a normal form in this mode depends on \mathbb{F}_2 .

Slimming down Petri Boxes: Compact Petri Net Models of Control Flows

Victor Khomenko ✉ 

School of Computing, Newcastle University, Newcastle upon Tyne, UK

Maciej Koutny ✉ 

School of Computing, Newcastle University, Newcastle upon Tyne, UK

Alex Yakovlev ✉ 

School of Engineering, Newcastle University, Merz Court, Newcastle upon Tyne, UK

Abstract

We look at the construction of compact Petri net models corresponding to process algebra expressions supporting sequential, choice, and parallel compositions. If “silent” transitions are disallowed, a construction based on Cartesian product is traditionally used to construct places in the target Petri net, resulting in an exponential explosion in the net size. We demonstrate that this exponential explosion can be avoided, by developing a link between this construction problem and the problem of finding an edge clique cover of a graph that is guaranteed to be complement-reducible (i.e., a cograph). It turns out that the exponential number of places created by the Cartesian product construction can be reduced down to polynomial (quadratic) even in the worst case, and to logarithmic in the best (non-degraded) case. As these results affect the “core” modelling techniques based on Petri nets, eliminating a source of an exponential explosion, we hope they will have applications in Petri net modelling and translations of various formalisms to Petri nets.

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases Petri net, Petri box, cograph, edge clique cover, control flow, static construction, local construction, interface graph, Burst automata, composition

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.8

1 Introduction

Petri nets have a special place among modelling formalisms due to their simplicity of semantics, intuitive graphical notation, and the possibility of capturing behaviours concisely without making subsequent processing (e.g., formal verification or synthesis) undecidable. This has led to the abundance of software tools for Petri nets, and to extensive use of Petri nets both as a modelling formalism and as an intermediate representation to which a model that was initially expressed in a different formalism is translated, e.g., to utilise efficient formal verification techniques and tools. In fact, developing translations from various process algebras and other formalisms to Petri nets has been a hot research topic for the past four decades, see e.g., [2, 5, 9, 12].

The possibility to create concise models is often the key advantage of Petri nets over simpler formalisms like Finite State Machines (FSMs). Indeed, it is generally accepted that one is likely to encounter the exponential *state space explosion* [13] during, e.g., formal verification – this problem is believed to be fundamental (unless $P=PSPACE$), and mitigating this explosion using heuristics has been a hot research topic for many years. However, encountering an exponential explosion already during the modelling stage would be unfortunate and indicative of problems in modelling techniques or even the formalism itself.

However, as we observed in [8], a naïve translation of even simple control flows to Petri nets may lead to an exponential explosion in the Petri net size. As a motivating example, [8] considers *Burst Automata* (BA) [3] – a formalism with applications in the area of asynchronous



© Victor Khomenko, Maciej Koutny, and Alex Yakovlev;
licensed under Creative Commons License CC-BY 4.0

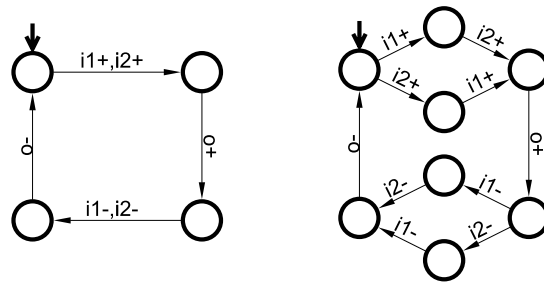
33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 8; pp. 8:1–8:16



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



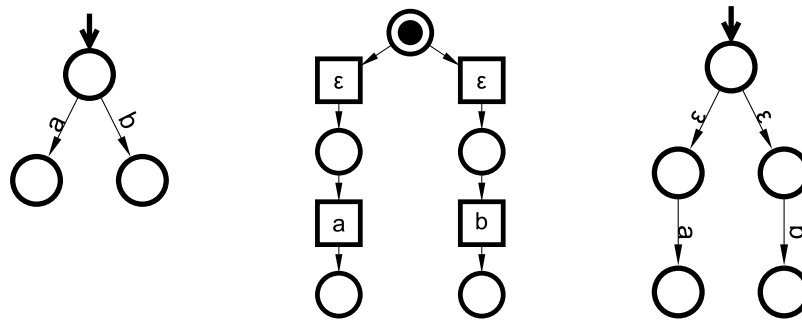
■ **Figure 1** A Burst Automata specification of the C-element and an FSM expressing its interleaving semantics. A C-element waits for both inputs to switch to 1 (actions i_1^+ and i_2^+) before switching its output to 1 (action o^+), and then waits for both inputs to switch to 0 (actions i_1^- and i_2^-) before switching its output to 0 (action o^-). It is assumed that the environment fulfills its part of the contract, i.e. each input switches only once before the output switches.

circuits design. Intuitively, Burst Automata are similar to FSMs, except that their arcs are labelled not by single actions but by sets of actions (“bursts”) which fire concurrently. One can define the interleaving semantics of Burst Automata by allowing the actions in a burst to fire in any order, which results in the usual FSM, see the example in Figure 1. When developing a translation between two formalisms, some kind of behavioural equivalence between the models is required, e.g., language equivalence or bisimulation of the corresponding FSMs. As Burst Automata are a very simple FSM-like formalism, it would be reasonable to expect that translating them to Petri nets would be quite simple and efficient.

However, developing a compact translation from Burst Automata to Petri nets is more complicated than one could expect – in particular, efficiently expressing a choice between several bursts of concurrent transitions is not trivial in Petri nets. In [3] a language-preserving linear size translation is proposed, that prefixes each burst with a silent “fork” transition and then uses another “join” transition after the burst to detect completion. Unfortunately, there are situations when this translation is unacceptable. First of all, silent transitions turn a deterministic model into a non-deterministic one, which is often undesirable (e.g., non-determinism cannot be directly implemented physically, say in an asynchronous logic circuit [4]). Second, language equivalence may be too weak (e.g., it does not preserve branching time temporal properties or even deadlocks), and prefixing bursts with silent transitions breaks not only strong but also weak bisimulation, see Figure 2.

To preserve strong bisimulation, the following *Cartesian Product Construction* (\times -construction) is traditionally used, see e.g., [1, 2, 5, 14]. To express a choice between several bursts (i.e., sets of concurrent transitions) B_1, B_2, \dots, B_n , this construction would create a set of places corresponding to tuples in $B_1 \times B_2 \times \dots \times B_n$, so that a place corresponding to a tuple (b_1, \dots, b_n) is connected to each transition b_i occurring in the tuple. This means that the number of created places is $|B_1| \cdot |B_2| \cdot \dots \cdot |B_n|$, i.e., the Petri net size is exponential in the number of bursts. For example, [3] developed two translations from Burst Automata to Petri nets based on \times -construction, which preserve either weak or strong bisimulation. However, in contrast to the linear size translation of [3] (that does not preserve even weak bisimulation), they may result in an exponentially large Petri net.

In [8] we proposed an alternative to \times -construction, that uses at most quadratic (in the total size of all bursts) number of places to express a choice between bursts, thereby reducing the size of Burst Automata to Petri net translation from exponential [3] down to polynomial. Furthermore, in some cases a logarithmic number of places is sufficient, yielding a double-exponential reduction compared with \times -construction. The technique was based



■ **Figure 2** A Burst Automaton with singleton bursts, so coinciding with the FSM expressing its interleaving semantics (left); its Petri net translation prefixing each burst with a silent “fork” transition (middle); the reachability graph (FSM) of this Petri net (right). Note that the two FSMs are language-equivalent but not weakly bisimilar.

on showing the equivalence between the modelling problem of expressing a choice between bursts of concurrent events and the problem of finding an edge clique cover of a complete multipartite graph.

In this paper, we generalise the technique of [8] to arbitrary control flows which are built from atomic actions using choice, concurrency, and sequencing operators. In particular, a polynomial translation of such control flows to Petri nets is possible, that preserves strong bisimulation (in fact, it guarantees the isomorphism of reachability graphs, which is an even stronger equivalence). The developed technique allowed us to further improve the translation of Burst Automata to Petri nets proposed in [8] by handling the sequence operator better, see Figure 3. The developed Petri net translation is compositional – this is ensured by augmenting Petri Box Algebra [2] with the notion of *interface graphs* (superseding the entry and exit places of Petri boxes) in a way that allowed us to import many results from Petri Box Algebra into the new framework.

Since the proposed construction affects the “core” modelling techniques for Petri nets and because the choice, concurrency, and sequencing operators are included in most process algebras and other formalisms for behavioural modelling, we believe it will have many applications. In particular, translations from various formalisms to Petri nets relying on the \times -construction can be significantly improved by using the proposed construction instead, eliminating thus a source of an exponential explosion.

The proposed construction is based on the observation that the problem of “gluing” two Petri boxes sequentially is equivalent to finding an *edge clique cover* of a certain *complement-reducible graph (cograph)* where some of the edges are already considered as “covered”, with the number of created places corresponding to the number of cliques in the cover. This results in an interesting optimisation problem that is in NP and likely NP-complete. In practice, the optimality is usually not required, and one can use simple approximations which yield useful lower and upper bounds – it is easy to see that at most polynomial (quadratic) number of cliques are always sufficient, which yields a polynomial Petri net.

2 Setting the scene

In this section, we describe a “bare bones” process algebra for expressing control flows, which can be regarded as a core fragment shared by many existing process algebras. We also discuss some basic notions related to Petri nets and clique covers of undirected graphs.

2.1 Models of concurrency

2.1.1 “Bare bones” process algebra

Consider a “bare bones” process algebra, where expressions are constructed from a finite alphabet of actions using operators “ \square ” (choice), “ \parallel ” concurrency, and “ $;$ ” (sequencing). This algebra allows one to model acyclic control flows. It is very simple and, in fact, most existing process algebras build on it, by adding more features and operators (e.g., communication and recursion).

One can then define the semantics of such expressions, e.g., using Finite State Machines, which can be exponential in the size of the expression due to concurrency, e.g., the FSMs for expressions of the form $a_1 \parallel a_2 \parallel \dots \parallel a_n$ would contain 2^n states.

One might hope that using Petri nets instead of FSMs would cope with the exponential explosion, i.e., that a polynomial translation from this process algebra to Petri nets is possible. However, as observed in [8], this is not trivial. In fact, the traditional \times -construction for modelling a choice between several “bursts” of concurrent actions creates an exponential number of places. For example, consider expressions of the form

$$(a_{11} \parallel \dots \parallel a_{1n}) \square \dots \square (a_{m1} \parallel \dots \parallel a_{mn})$$

representing a choice between m “bursts” each containing n concurrent actions. The \times -construction creates m^n places to express this in a Petri net, which is exponential in the length of the above expression.

In this paper, we demonstrate that a polynomial bisimulation-preserving translation to Petri nets is indeed possible for any “bare bones” process algebra expressions. In fact, the isomorphism of reachability graphs (which is a stronger equivalence than strong bisimulation) holds for the developed translation.

2.1.2 Petri nets

We focus on *safe* (i.e., at most one token per place) Petri nets, which are often used for modelling control flows. For a safe Petri net, the total number of tokens in its initial marking cannot exceed the number of places, so we can define its size as the total number of places, transitions, and arcs, disregarding the initial marking. Note that the size of a Petri net is dominated by its arcs, except the uninteresting degraded case when there are many isolated nodes.

In this paper, the set of transitions is usually given (e.g., when translating a model from some other formalism to Petri nets, the transitions often correspond to the occurrences of actions in that model), and the objective is to express the intended behaviour using small numbers of places and arcs. Note that having a small number of places is often desirable for formal verification as they correspond to state variables, and having a small number of arcs is desirable as they dominate the Petri net size.

2.2 Graphs

We consider undirected graphs with no parallel edges and no self-loops. For simplicity, a graph $(\{v\}, \emptyset)$ comprising a single vertex v and no edges will be denoted just by v .

2.2.1 Edge clique covers

A *clique* in a graph is a set of vertices which are pairwise connected by edges. A clique is called *maximal* (or max-clique) if it is not a subset of any other clique. In what follows, $\text{maxCL}(G)$ is the set of all the max-cliques of an undirected graph G .

A set of cliques in a graph form an *edge clique cover* (ECC) if, for every edge, there is at least one clique that contains both endpoints of this edge. The number of cliques in an ECC is called its *size*. Note that, given an ECC, one can expand each clique in it to some maximal one, without increasing the size of the ECC. The minimum possible size of an ECC of a graph G is the *edge clique cover number* (a.k.a. *intersection number*) of G , and will be denoted $\text{ecc}(G)$.

2.2.2 Complete multipartite graphs

A graph is called *multipartite* if its vertices are partitioned into several sets in such a way that there are no edges between vertices in the same part. A multipartite graph is *complete* if for every pair of vertices from different parts there is an edge connecting them. A complete multipartite graph with the parts of sizes $t_1 \leq t_2 \leq \dots \leq t_n$ will be denoted K_{t_1, t_2, \dots, t_n} .

2.2.3 Cographs

Complement-reducible graphs (cographs) [10] can be recursively defined as follows: (i) a single vertex graph is a cograph; (ii) the complement of a cograph is a cograph; (iii) the disjoint union of cographs is a cograph. Intuitively, for every cograph G with more than one vertex, either G or its complement \overline{G} is not connected. One can easily show that any complete multipartite graph is a cograph.

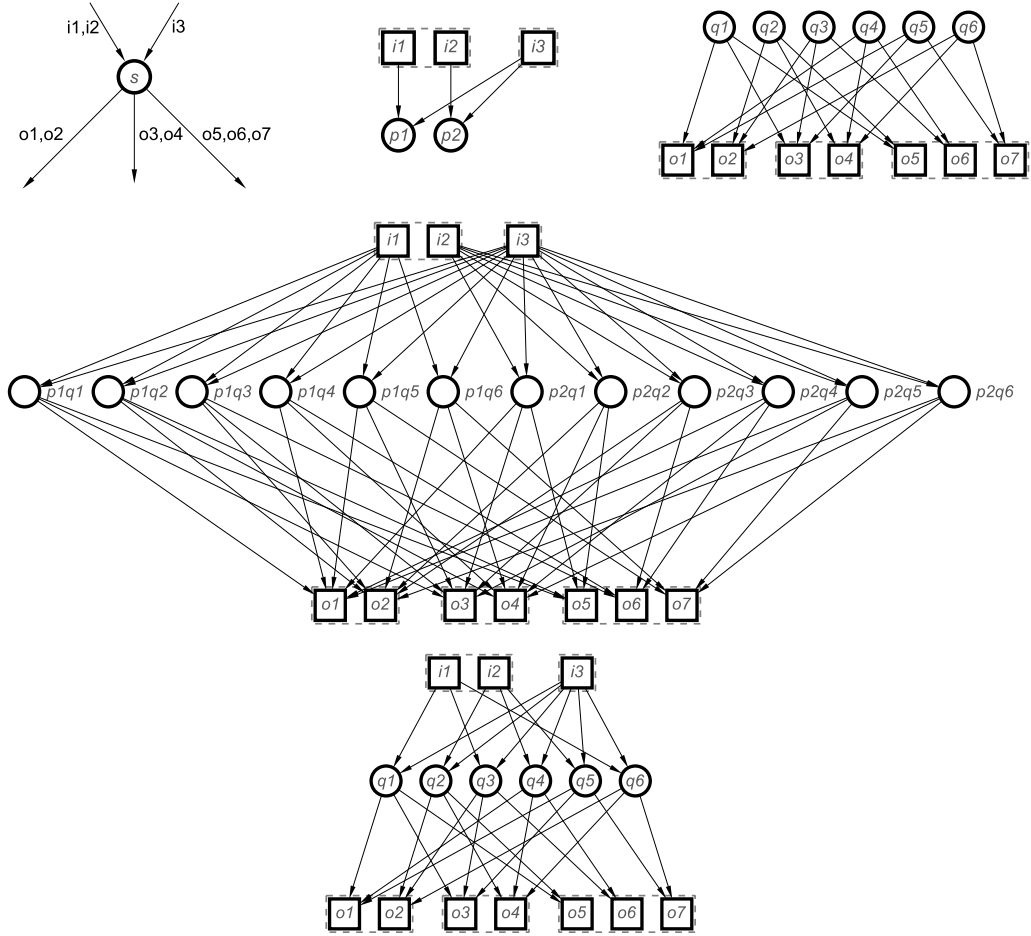
The *join* operation $G_1 \text{---} G_2$ consists of forming the disjoint union $G_1 \uplus G_2$ and then adding an edge between every vertex of G_1 and every vertex of G_2 . One can see that one can reformulate the above definition of cographs so that it uses the join instead of the complement. Indeed, it is easy to see that $G_1 \text{---} G_2 = \overline{\overline{G_1} \uplus \overline{G_2}}$, i.e., join can be expressed via disjoint union and complementation. Similarly, one can express the complementation via disjoint union and join for cographs by repeatedly applying the following rewriting rules:

$$\begin{aligned} \overline{\overline{v}} &= v \\ \overline{\overline{G}} &= G \\ \overline{G_1 \uplus G_2} &= \overline{G_1} \text{---} \overline{G_2}. \end{aligned}$$

3 Sequential composition and edge clique covers

In this section we informally present the underlying idea of the proposed construction, and illustrate it on a simple example from [8] of modelling a fragment of a BA shown in Figure 3(top-left) as a Petri net. There, a BA state with incoming bursts $\{i_1, i_2\}$ and $\{i_3\}$ and outgoing bursts $\{o_1, o_2\}$, $\{o_3, o_4\}$, and $\{o_5, o_6, o_7\}$ should be modelled as a set of Petri net places, assuming that the actions occurring in bursts are modelled as transitions with the corresponding names. More generally, we consider control flows constructed from atomic actions using sequential, choice, and parallel compositions. Such control flows can be naturally represented by acyclic safe Petri nets (though there are interesting “unnatural” Petri net representations with cycles, see e.g., Section 4.3). The technique illustrated by the translation of this BA fragment can be applied recursively, and thus naturally generalises to such control flows.

In Petri nets modelling control flows, places have the dual role of enforcing both sequencing of transitions and choice between transitions. Indeed, let t_1 and t_2 be two transitions connected to the same place p :



■ **Figure 3** An example of bisimulation-preserving BA to Petri net translation: **(top-left)** a BA state with its incoming and outgoing bursts; **(top-center)** Petri net translation [8] of incoming bursts – the maximal incoming burst size is two, so two places are created; **(top-right)** Petri net translation [8] of outgoing bursts – $\text{ecc}(K_{2,2,3}) = 6$ places are created; **(middle)** the combined Petri net [8] – the 12 places correspond to pairs in $\{p_1, p_2\} \times \{q_1, \dots, q_6\}$; **(bottom)** the improved construction presented in this paper – the 6 places correspond to an ECC of $(i_1 \uplus i_2) \text{---} i_3 \text{---} (o_1 \uplus o_2) \text{---} (o_3 \uplus o_4) \text{---} (o_5 \uplus o_6 \uplus o_7)$ with the edges of $(i_1 \uplus i_2) \text{---} i_3$ being optional to cover.

- If the connections are of the form $t_1 \rightarrow p \rightarrow t_2$ then p enforces sequencing of these transitions, as t_2 can fire only after t_1 .
- If the connections are of the form $p \rightarrow t_1$ and $p \rightarrow t_2$ then p enforces the choice between these transitions, as only one of them can fire.
- If the connections are of the form $t_1 \rightarrow p$ and $t_2 \rightarrow p$ then there is a choice between these transitions (as otherwise the Petri net would be unsafe), but this choice is enforced not by p but by some other place.

Note that t_1 and t_2 are not concurrent if they are connected to the same place.

In our example, there is a choice between the incoming bursts $\{i_1, i_2\}$ and $\{i_3\}$ but it is enforced not by state s of the BA (or the places corresponding to s in the Petri net translation) but elsewhere. There is a choice between outgoing bursts $\{o_1, o_2\}$, $\{o_3, o_4\}$, and $\{o_5, o_6, o_7\}$, and it is enforced by s (and the corresponding places in the Petri net translation).

Moreover, the sequencing between these incoming and outgoing bursts is enforced by s (and the corresponding places in the Petri net translation). Note also that there is sequencing within neither incoming nor outgoing bursts, which leads to the following notion.

A set of transitions T is called *non-sequential* if no two distinct transitions in it are sequential. In other words, there may be choices between some transitions in such a set, and the transitions in any subset of T are concurrent as long as this subset contains no two transitions which are in the choice relationship. In our example, the set of incoming transitions $\{i_1, i_2, i_3\}$ and the set of outgoing transitions $\{o_1, o_2, o_3, o_4, o_5, o_6, o_7\}$ are non-sequential. The behaviour of T can be viewed as a collection of maximal such sets of concurrent transitions. One can represent the choice relation between transitions in T as a graph $G(T)$ (that will be later formalised in the notion of *interface graphs*) where the vertices are the transitions of T and there is an edge between two vertices iff the corresponding transitions are in the choice relationship.

Consider now two disjoint non-empty non-sequential sets of transitions, T_1 and T_2 , which are to be composed sequentially. In the example shown in Figure 3(top-left),

$$\begin{aligned} T_1 &= \{i_1, i_2, i_3\} \\ G(T_1) &= (i_1 \uplus i_2) \text{---} i_3 \\ T_2 &= \{o_1, o_2, o_3, o_4, o_5, o_6, o_7\} \\ G(T_2) &= (o_1 \uplus o_2) \text{---} (o_3 \uplus o_4) \text{---} (o_5 \uplus o_6 \uplus o_7) . \end{aligned}$$

The task is now to add places “between” transitions in T_1 and T_2 and connected to the transitions in T_1 by transition \rightarrow place arcs, and to transitions in T_2 by place \rightarrow transition arcs, so that:

- the behaviours of T_1 and T_2 are composed sequentially, i.e., none of the transitions in T_2 can fire until a maximal concurrent set of transitions in T_1 fires;
- the choices between transitions in T_2 are enforced (no need to enforce the choices between transitions in T_1 – this is done elsewhere).

Consider now a place p connected to some set of transitions $C \subseteq T_1 \cup T_2$ (the directions of arcs can be easily inferred). The key observation is that C must be a clique in $G(T_1) \text{---} G(T_2)$, as otherwise the resulting behaviour will be wrong. Indeed, for the sake of contradiction, suppose $t, t' \in C$ but there is no edge in $G(T_1) \text{---} G(T_2)$ connecting t and t' . Then these transitions are either both in T_1 or both in T_2 , as otherwise there would be an edge between them. If these transitions are both in T_1 , they are concurrent and both produce a token on p , resulting in an unsafe Petri net; e.g., in our example i_1 and i_2 are concurrent and including them both into C would result in p being unsafe. If $t, t' \in T_2$ then p creates a choice between them, even though they were meant to be concurrent, thus changing the intended behaviour; e.g., in our example o_1 and o_2 are concurrent, and including them both into C would result in p creating a choice between them.

Furthermore, though we consider acyclic control flows, in practice they are often just fragments of a higher level control flow that may contain cycles, which means the constructed acyclic Petri net will be a fragment of a larger Petri net that may contain cycles. This means the constructed Petri net should be “reusable”, i.e., executable in a cycle without breaking the safeness of Petri net or introducing deadlocks. We now argue that in such a case C must be a max-clique in $G(T_1) \text{---} G(T_2)$.

First, we show that reusability implies $C \not\subseteq T_1$ and $C \not\subseteq T_2$. Indeed, if $C \subseteq T_1$ then p has no outgoing arcs – besides being useless as it will never affect the enabledness of any transitions, p also accumulates tokens and an attempt to reuse this Petri net will result in unsafeness. If $C \subseteq T_2$ then p has no incoming arcs and so cannot obtain a new token – even if it contains a token initially, the Petri net fragment cannot be reused as this can introduce a deadlock.

Now, for the sake of contradiction, suppose C is not maximal, i.e., it can be expanded to a larger clique by adding t . If $t \in T_1$ then firing t means none of the transitions in $C \cap T_1 \neq \emptyset$ can fire, as each of them is in choice relationship with t , in which case p does not obtain a token and transitions in $C \cap T_2 \neq \emptyset$ will not be able to fire, which changes the intended behavior. If $t \in T_2$ then firing t means none of the transitions in $C \cap T_2 \neq \emptyset$ can fire, as each of them is in choice relationship with t , in which case the token in p cannot be consumed and the Petri net is not reusable as this would introduce unsafeness. Hence, C is a max-clique in $G(T_1) \text{---} G(T_2)$; note that this implies $C \not\subseteq T_1$ and $C \not\subseteq T_2$.

We have now established that every constructed place must correspond to some max-clique in $G(T_1) \text{---} G(T_2)$. The question now is how to select a set of such places sufficient to express the desired behavior, which can be reformulated as a question of selecting a sufficient number of max-cliques in $G(T_1) \text{---} G(T_2)$. One can observe that this problem is similar to the well known problem of computing an edge clique cover of $G(T_1) \text{---} G(T_2)$, except that covering the edges in the induced subgraph $G(T_1)$ is optional. (Note that any edge clique cover can be easily extended to one containing only max-cliques, without increasing the number of cliques.) Indeed, suppose (t, t') is an edge of $G(T_1) \text{---} G(T_2)$:

- If $t, t' \in T_1$ then t and t' are in the choice relationship but this is enforced elsewhere, so it is not necessary (and not possible) to enforce it by adding a place p with the arcs $t \rightarrow p$ and $t' \rightarrow p$; on the other hand, there is no harm having such a place, e.g., both t and t' can be in the same max-clique that also covers other edges of $G(T_1) \text{---} G(T_2)$. Hence, it is optional to cover the edge (t, t') .
- If $t, t' \in T_2$ then t and t' are in the choice relationship and this must be enforced by having a place p with the arcs $p \rightarrow t$ and $p \rightarrow t'$. Hence, it is necessary to cover the edge (t, t') .
- If $t \in T_1$ and $t' \in T_2$ then t must fire before t' and this must be enforced by having a place p with the arcs $t \rightarrow p$ and $p \rightarrow t'$. Hence, it is necessary to cover the edge (t, t') .

This version of the edge clique cover problem will be called *partial edge clique cover problem* (PECCP). It is a natural extension of the standard edge clique cover problem (ECCP), in fact some state-of-art ECCP algorithms, e.g., [6], start from an empty cover and keep adding cliques one-by-one until all edges are covered; hence, at the intermediate stages of the computation, the problem is equivalent to PECCP, with the edges covered by previously added cliques becoming optional to cover. This means that many existing ECCP algorithms can be adapted to solve PECCP.

Note that PECCP is trivially in NP. Moreover, since ECCP is NP-complete for general graphs and is a special case of PECCP, it follows that PECCP is NP-complete for general graphs. However, for control flows constructed from atomic actions using sequential, choice, and parallel compositions, the graphs on which PECCP is solved are guaranteed to be cographs, which is a very restricted subclass of graphs, with many NP-complete problems (e.g., computing a clique of maximal cardinality) becoming polynomial when restricted to this class. However, to our knowledge, the question whether ECCP or PECCP is NP-complete on cographs is still open. Note also that for modelling control flows the optimality is not required, so fast heuristic algorithms computing small but not necessarily smallest covers would be sufficient – in fact, the trivial ECC covering each edge by a separate clique already avoids the exponential explosion due to the \times -construction.

Coming back to our example, our previous method described in [8] creates 12 places. It handles incoming and outgoing bursts separately; for the incoming bursts the number of created places corresponds to the maximal input burst cardinality (2 in our example, see Figure 3(top-centre)), and for the outgoing bursts the places correspond to ECC of $G(T_2)$,

i.e., $(o_1 \uplus o_2) \text{---} (o_3 \uplus o_4) \text{---} (o_5 \uplus o_6 \uplus o_7)$ in our example. For BAS $G(T_2)$ is guaranteed to be a multipartite graph, $K_{2,2,3}$ in this case – it has an ECC of size 6, i.e., 6 places are created, see Figure 3(top-right). Then a variant of \times -construction is used to enforce the sequencing of incoming and outgoing bursts, which results in $2 \cdot 6 = 12$ places, as shown in Figure 3(middle).

Though the construction in [8] yields a polynomial BA to Petri net translation, the translation presented in this paper significantly improves it. For this example, $G(T_1) \text{---} G(T_2) = (i_1 \uplus i_2) \text{---} i_3 \text{---} (o_1 \uplus o_2) \text{---} (o_3 \uplus o_4) \text{---} (o_5 \uplus o_6 \uplus o_7)$, with the edges of the induced subgraph $(i_1 \uplus i_2) \text{---} i_3$ being optional to cover. Solving PECCP yields a cover with 6 cliques, and the corresponding Petri net fragment with 6 places is shown in Figure 3(bottom).

4 Slimming down Box Algebra

Box Algebra [2] provided a generic process-algebraic syntax together with a compositional translation to a class of Petri nets called (*Petri*) *boxes*. The algebra has several concrete incarnations, including CCS [11] and TCSP [7]. Here we are only interested in a small fragment of Box Algebra corresponding to the “bare bones” process algebra, in order to focus on the salient aspect of control flow in nets constructed from three fundamental composition operators. In particular, we omit all communication and synchronisation aspects.

Process expressions, or *box expressions*, are derived from the syntax

$$E ::= a \mid E;E \mid E \square E \mid E \parallel E \quad (1)$$

where a is an atomic action. Since we deal only with control flows and actions do not have any special semantics (e.g., communication), we can assume that no action occurs more than once in any box expression we consider. Intuitively, a denotes a process which can execute atomic action a and terminate, $E;F$ denotes sequential composition of two processes, $E \square F$ denotes choice composition, and $E \parallel F$ denotes parallel composition.

The semantics of box expressions is given through a translation into Petri nets, called *boxes*. For the simple syntax (1), each place can be uniquely identified by its input and output transitions, and $\pi_{U,W}$ will denote a place with input transitions U and output transitions W , i.e., there is an arrow from transition t to $\pi_{U,W}$ iff $t \in U$, and there is an arrow from $\pi_{U,W}$ to transition t iff $t \in W$.

A *box* is a pair $N = (P, T)$, where $P(= P_N)$ is a finite set of *places* (local states) and $T(= T_N)$ is a disjoint finite set of *transitions* (actions) such that, for every transition $t \in T$, there is a place $\pi_{U,W} \in P$ with $t \in W$.

The flow relation of N (represented by arrows) is implicit and can be recovered from the sets indexing places: the *input* and *output* places of a transition $t \in T$ are respectively given by $pre_N(t) = \{\pi_{U,W} \in P \mid t \in W\}$ and $post_N(t) = \{\pi_{U,W} \in P \mid t \in U\}$.

Associating boxes with box expressions is done compositionally, through the $\text{box}(\cdot)$ mapping, by combining their entry and exit places to reflect the intended control flow of execution, where the *entry* places N^e of a box N have the form $\pi_{\emptyset,W} \in P$, and the *exit* places N^\times have the form $\pi_{U,\emptyset} \in P$. The *internal* places N^i are all the remaining places of N . Intuitively:

- $\text{box}(E \parallel F)$ is obtained simply by placing $\text{box}(E)$ and $\text{box}(F)$ side by side.
- $\text{box}(E \square F)$ is obtained by placing $\text{box}(E)$ and $\text{box}(F)$ side by side and then gluing each entry place of $\text{box}(E)$ with each entry place of $\text{box}(F)$, effectively creating the product $\text{box}(E)^e \times \text{box}(F)^e$, and similarly for the exit places.
- $\text{box}(E;F)$ is obtained by placing $\text{box}(E)$ above $\text{box}(F)$ and then gluing each exit place of $\text{box}(E)$ with each entry place of $\text{box}(F)$, effectively creating the product $\text{box}(E)^\times \times \text{box}(F)^e$.

8:10 Slimming down Petri Boxes: Compact Petri Net Models of Control Flows

The above procedure will be referred to as the \times -construction. Formally, the box corresponding to a given box expression is defined recursively as follows:

$$\begin{aligned} \mathbf{box}(a) &= (\{\pi_{\emptyset, \{a\}}, \pi_{\{a\}, \emptyset}\}, \{a\}) \\ \mathbf{box}(E \parallel F) &= (P_{\mathbf{box}(E)} \cup P_{\mathbf{box}(F)}, \mathcal{T}) \\ \mathbf{box}(E \square F) &= (\mathcal{E} \cup \mathcal{P} \cup \mathcal{X}, \mathcal{T}) \\ \mathbf{box}(E; F) &= (\mathbf{box}(E)^e \cup \mathcal{P} \cup \mathcal{I} \cup \mathbf{box}(F)^x, \mathcal{T}) \end{aligned}$$

where $\mathcal{T} = T_{\mathbf{box}(E)} \cup T_{\mathbf{box}(F)}$, $\mathcal{P} = \mathbf{box}(E)^i \cup \mathbf{box}(F)^i$, and:

$$\begin{aligned} \mathcal{E} &= \{\pi_{\emptyset, U \cup W} \mid \pi_{\emptyset, U} \in \mathbf{box}(E)^e \wedge \pi_{\emptyset, W} \in \mathbf{box}(F)^e\} \\ \mathcal{X} &= \{\pi_{U \cup W, \emptyset} \mid \pi_{U, \emptyset} \in \mathbf{box}(E)^x \wedge \pi_{W, \emptyset} \in \mathbf{box}(F)^x\} \\ \mathcal{I} &= \{\pi_{U, W} \mid \pi_{U, \emptyset} \in \mathbf{box}(E)^x \wedge \pi_{\emptyset, W} \in \mathbf{box}(F)^e\}. \end{aligned} \quad (2)$$

Markings (global states) of a box $N = (P, T)$ we consider are sets of places. The default initial marking is N^e . A transition $t \in T$ is *enabled* at marking M if $pre_N(t) \leq M$. It then can be *fired* leading to the marking $M' = M - pre_N(t) + post_N(t)$, and we denote this by $M[t]_N M'$. An overall behaviour of N is given by its *reachability graph* defined as a labelled directed graph $RG(N) = (\mathcal{R}, \mathcal{A}, N^e)$, where \mathcal{R} are the *reachable markings* of N (i.e., the least set containing N^e and such that if $M \in \mathcal{R}$ and $M[t]_N M'$ then $M' \in \mathcal{R}$), and \mathcal{A} contains all labelled arcs (M, t, M') such that $M \in \mathcal{R}$ and $M[t]_N M'$.

To formulate a central semantical result of this paper – a full (local) characterisation of successful slimming down of compositionally defined boxes – we need two more notions. The *local conflict* and *local causality* of N are two relations on transitions given respectively by:

$$\begin{aligned} \mathit{Confl}(N) &= \{(t, u) \in T \times T \mid pre_N(t) \cap pre_N(u) \neq \emptyset\} \\ \mathit{Caused}(N) &= \{(t, u) \in T \times T \mid post_N(t) \cap pre_N(u) \neq \emptyset\}. \end{aligned} \quad (3)$$

Intuitively, these two relations capture the dual role of Petri net places in enforcing both choice and causality between transitions.

► **Proposition 1.** *Let E be a box expression derived from the syntax (1), and $N = (P, T_{\mathbf{box}(E)})$ be a box such that $P \subseteq P_{\mathbf{box}(E)}$.*

Then $RG(N)$ and $RG(\mathbf{box}(E))$ are isomorphic reachability graphs if and only if $\mathit{Confl}(N) = \mathit{Confl}(\mathbf{box}(E))$ and $\mathit{Caused}(N) = \mathit{Caused}(\mathbf{box}(E))$.

Proposition 1 means that we can safely delete places from $\mathbf{box}(E)$ iff the local conflict and local causality relations on the transitions are retained. Note that such a property does not hold in general – in either direction – not even for boxes which are safe and acyclic. Indeed, let $N = \mathbf{box}(a; b; c)$, and let N' be N with an added place $\pi_{\{a\}, \{c\}}$. Then $RG(N)$ is isomorphic to $RG(N')$, but

$$\mathit{Caused}(N') = \mathit{Caused}(N) \uplus \{(a, c)\}.$$

As a complementary example, let $N = \mathbf{box}((a \square b); c)$, and let N' be N with an added place $\pi_{\{b\}, \{c\}}$. Then

$$\mathit{Confl}(N) = \mathit{Confl}(N') \quad \text{and} \quad \mathit{Caused}(N) = \mathit{Caused}(N'),$$

but $RG(N)$ is not isomorphic to $RG(N')$ as in $RG(N')$ one cannot “execute” a followed by c .

Our goal now becomes that of finding criteria for identifying possibly largest sets of such “safe deletions” characterised by Proposition 1, and thus transferring the slimming down problem from semantic to algorithmic setting. What is more, we aim at developing a “static” approach to slimming, i.e., one that does not require looking into the behaviour of boxes.

4.1 Interface graphs

In this section, we introduce and investigate a novel concept in the area of Petri boxes which aims at capturing different ways in which local conflict and local causality can arise. We start by introducing *efficient (quadratic) representation* of the entry and exit places of $\text{box}(E)$ without going through the *expensive (exponential) process* of applying the \times -construction.

For each box expression E derived from the syntax (1), let G_E^e and G_E^x be undirected *interface graphs* defined recursively as follows:

$$\begin{array}{ll}
 G_a^e & = a & G_a^x & = a \\
 G_{E;F}^e & = G_E^e & G_{E;F}^x & = G_F^x \\
 G_{E \square F}^e & = G_E^e \text{---} G_F^e & G_{E \square F}^x & = G_E^x \text{---} G_F^x \\
 G_{E \parallel F}^e & = G_E^e \uplus G_F^e & G_{E \parallel F}^x & = G_E^x \uplus G_F^x
 \end{array} \tag{4}$$

Note that the vertices of G_E^e and G_E^x are transitions of $\text{box}(E)$, and that interface graphs are cographs. For example,

$$\begin{array}{l}
 G_{(a \parallel b) \square c; (d \parallel e)}^e = G_{(a \parallel b) \square c}^e \\
 \quad = G_{a \parallel b}^e \text{---} c \\
 \quad = (a \uplus b) \text{---} c \\
 G_{(a \parallel b) \square c; (d \parallel e)}^x = G_{d \parallel e}^x \\
 \quad = d \uplus e .
 \end{array}$$

► **Proposition 2.** *For box expression E derived from the syntax (1),*

$$\begin{array}{l}
 \text{box}(E)^e = \{\pi_{\emptyset, Cl} \mid Cl \in \text{maxCL}(G_E^e)\} \\
 \text{box}(E)^x = \{\pi_{Cl, \emptyset} \mid Cl \in \text{maxCL}(G_E^x)\} .
 \end{array}$$

Hence the entry places of $\text{box}(E)$ can be identified with the set of all max-cliques of G_E^e , and similarly for the exit places. As a result, by Proposition 1, removing some entry places from $\text{box}(E)$ without changing the overall behaviour is the same as choosing an edge covering of G_E^e by max-cliques (and the best result is therefore obtained by taking a minimal edge covering of G_E^e by max-cliques).

A similar observation holds for the internal places:

► **Proposition 3.** *If \mathcal{I} is a set of internal places as in Eq. (2), derived during the \times -construction, then:*

$$\mathcal{I} = \{\pi_{Cl \cap T_{\text{box}(E)}, Cl \cap T_{\text{box}(F)}} \mid Cl \in \text{maxCL}(G_E^x \text{---} G_F^e)\} .$$

4.2 New construction

*“Some people want it to happen.
Some wish it would happen.
Others make it happen”*

Michael Jordan on healthy dieting

We now introduce an alternative to the \times -construction, for a box expression H derived from the syntax (1). Formally, a *slimming* of H is any box N derived from $\text{box}(H)$, in the following way:

- The places in $\text{box}(H)^e$ are replaced by $\{\pi_{\emptyset, Cl} \mid Cl \in \mathcal{CL}\}$, where \mathcal{CL} is any minimal edge covering of G_H^e by max-cliques.

- Set \mathcal{I} of internal places as in Eq. (2) corresponding to every sub-expression of the form $E; F$ within H is replaced by

$$\{\pi_{Cl \cap T_{\text{box}(E)}, Cl \cap T_{\text{box}(F)}} \mid Cl \in \mathcal{CL}\},$$

where \mathcal{CL} is any minimal edge covering of $G_E^x \text{---} G_F^e$ by max-cliques, with the edges within its subgraph G_E^x being optional to cover.

- The places in $\text{box}(H)^x$ are deleted.

Note also that there is no need to generate at all the sets $\text{box}(H)^e$, $\text{box}(H)^x$ and \mathcal{I} , and only operate on graphs and their covers to derive the places of N . The proposed (non-deterministic) construction is therefore truly static and local.

► **Proposition 4.** *Let E be a box expression derived from the syntax (1) and N be a slimming of E . Then the following hold:*

1. $RG(N)$ and $RG(\text{box}(E))$ are isomorphic reachability graphs.
2. If $N' = (P, T_{\text{box}(E)})$ is a box such that $P \subset P_{\text{box}(E)}$ and $|P| < |P_N|$, then $RG(N')$ and $RG(\text{box}(E))$ are not isomorphic reachability graphs (and not even language equivalent).

The above is a key result validating the proposed way of deleting places from the composite boxes, and demonstrating its local optimality in the sense that deleting any remaining place will change the behaviour (note that there are other reasonable notions of optimality, see below). As was explained before, computing an optimal slimming is equivalent to solving PECCP on a cograph – to our knowledge it is still an open question whether this problem is NP-complete. Having said that, in practice, one does not need minimal covers and heuristic approximations yield good solutions (even trivial covers yield polynomial boxes contrasting with the exponential \times -construction). Note also that Proposition 4 can be lifted to other control flow operators of the Box Algebra, e.g., iteration.

4.3 No global optimality

The construction proposed in this paper achieves *local* optimality in terms of the number of places, as well as a guaranteed polynomial (in the length of the box expression) size of the overall Petri net, as opposed to the original \times -construction that is exponential. Furthermore, in cases like

$$(a_1 \parallel b_1) \square \dots \square (a_n \parallel b_n),$$

the number of created places can be as low as logarithmic – a double-exponential reduction w.r.t. the \times -construction.

This naturally raises the question whether the proposed construction is globally optimal, i.e., whether for a given box expression, a safe Petri net with the minimum possible number of places is always constructed. Unfortunately the answer turns out to be negative. In fact, the number of places is not even asymptotically optimal. We demonstrate this using the following very simple example.

Consider the box expression $a_1; \dots; a_n$, for which the proposed construction generates a Petri net with n places (denoted p_i , for $i = 1, \dots, n$), such that there are arcs from a_i to p_{i+1} , for $i = 1, \dots, n-1$, and from p_i to a_i , for $i = 1, \dots, n$. Moreover, p_1 is initially marked.

Observe that at most one of these places contains a token, and so the reachable markings of this Petri nets can be compactly encoded using only a logarithmic number of places, e.g., using the following construction. For simplicity, we assume that $n = \binom{k}{k/2}$ for some even k , i.e., n is the number of subsets of size $k/2$ of $\{1, \dots, k\}$. We denote these subsets P_1, \dots, P_n (the order can be chosen arbitrarily). Note that $k \sim \log_2 n$ as one can check (using, e.g., wolframalpha.com) that

$$\lim_{k \rightarrow +\infty} \frac{\log_2 n}{k} = \lim_{k \rightarrow +\infty} \frac{\log_2 \binom{k}{k/2}}{k} = 1.$$

Consider now a Petri net with the transitions a_i , $i = 1, \dots, n$, and k places numbered 1 to k . One can now interpret P_1, \dots, P_n as subsets of the set of places of this Petri net. These places and transitions are connected so that there are arcs from a_i to each place in P_{i+1} , for $i = 1, \dots, n-1$, and from each place in P_i to a_i , for $i = 1, \dots, n$. Moreover, the places in P_1 are initially marked. One can easily see that this Petri net has the expected behaviour, but only $k \sim \log_2 n$ rather than n places.

5 Conclusions

In this paper, we observed that the \times -construction traditionally used for composing, e.g., Petri boxes, is sub-optimal and causes an exponential explosion in the size of Petri nets that can be avoided. We showed the equivalence between this modelling problem and the problem of finding an ECC of a cograph, where the covering of some edges is considered optional. This allowed us to develop a polynomial Petri net translation of arbitrary control flows built from atomic actions using the sequential, choice, and parallel compositions.

These results affect the “core” modelling techniques based on Petri nets and eliminate a source of exponential explosion when modelling control flows, and in translations from various process algebras and other formalisms to Petri nets.

References

- 1 Eike Best, Raymond R. Devillers, and Jon G. Hall. The box calculus: a new causal algebra with multi-label communication. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1992, The DEMON Project*, volume 609 of *Lecture Notes in Computer Science*, pages 21–69. Springer, 1992. doi:10.1007/3-540-55610-9_167.
- 2 Eike Best, Raymond R. Devillers, and Maciej Koutny. *Petri net algebra*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2001.
- 3 A. Chan, Danil Sokolov, Victor Khomenko, David Lloyd, and Alex Yakovlev. Burst automaton: Framework for speed-independent synthesis using burst-mode specifications. *submitted*, 2021.
- 4 Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alex Yakovlev. *Logic Synthesis for Asynchronous Controllers and Interfaces*. Springer, 2002.
- 5 Ursula Goltz and Alan Mycroft. On the relationship of CCS and Petri nets. In Jan Paredaens, editor, *Automata, Languages and Programming, 11th Colloquium, Antwerp, Belgium, July 16-20, 1984, Proceedings*, volume 172 of *Lecture Notes in Computer Science*, pages 196–208. Springer, 1984.
- 6 Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Data reduction and exact algorithms for clique cover. *ACM J. Experimental Algorithmics*, 13, 2009.
- 7 C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
- 8 Victor Khomenko, Maciej Koutny, and Alex Yakovlev. Avoiding exponential explosion in Petri net models of control flows. In *Proc. Petri Nets’22*, Lecture Notes in Computer Science. Springer, 2022. accepted paper.
- 9 Victor Khomenko, Roland Meyer, and Reiner Hüchting. A polynomial translation of pi-calculus FCPs to safe Petri nets. *Log. Methods Comput. Sci.*, 9(3), 2013.
- 10 H. Lerchs. *On cliques and kernels*. Tech. Report, Dept. of Comp. Sci., Univ. of Toronto, 1971.
- 11 Robin Milner. *A Calculus of Communicating Systems*. Springer, 1980.

- 12 Ernst-Rüdiger Olderog. Operational Petri net semantics for CCSP. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1987, covers the 7th European Workshop on Applications and Theory of Petri Nets, Oxford, UK, June 1986*, volume 266 of *Lecture Notes in Computer Science*, pages 196–223. Springer, 1986.
- 13 Antti Valmari. The state explosion problem. In Wolfgang Reisig and Grzegorz Rozenberg, editors, *Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996*, volume 1491 of *Lecture Notes in Computer Science*, pages 429–528. Springer, 1996.
- 14 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, *Stepwise Refinement of Distributed Systems, Models, Formalisms, Correctness, REX Workshop, Mook, The Netherlands, May 29 - June 2, 1989, Proceedings*, volume 430 of *Lecture Notes in Computer Science*, pages 267–300. Springer, 1989. doi:10.1007/3-540-52559-9_68.

A Additional notions and notations

Let $N = (P, T)$ be a box.

- If a transition $t \in T$ is enabled at marking M , we denote $t \in \text{enabled}_N(M)$.
- A marking M' is *reachable from marking M* if there are markings M_0, \dots, M_k ($k \geq 0$) and transitions t_1, \dots, t_k such that $M_0 = M$, $M_k = M'$, and

$$M_0[t_1]_N M_1 \dots M_{k-1}[t_k]_N M_k.$$

Moreover, if $M_0 = N^e$ then $\sigma = t_1 \dots t_k$ is a *firing sequence* of N (we denote this by $\sigma \in \text{fseq}(N)$) and M_k is a *reachable marking* of N (we denote this by $M_k \in \text{reach}(N)$).

- Two transitions $t \neq u \in T$ are *in conflict* if there is a place $p = \pi_{U,W} \in P$ such that $t, u \in W$ (we denote this by $(t, u) \in \text{cfl}_N(p)$), and t *directly causes* u if there is a place $p = \pi_{U,W} \in P$ such that $t \in U$ and $u \in W$ (we denote this by $(t, u) \in \text{csd}_N(p)$). Moreover, for a set of places $P' \subseteq P$:

$$\text{cfl}_N(P') = \bigcup \{ \text{cfl}_N(p) \mid p \in P' \} \quad \text{and} \quad \text{csd}_N(P') = \bigcup \{ \text{csd}_N(p) \mid p \in P' \}.$$

B Proof of Proposition 1

► **Lemma 5.** *Let E be a box expression derived from the syntax (1), $N = \text{box}(E)$, $t \in T_N$, and $(N^e \Rightarrow) M_0[t_1]_N M_1 \dots M_{k-1}[t_k]_N M_k$.¹*

1. N is safe box (i.e., each reachable marking is a set of places).²
2. N^\times is a marking reachable from M_k , and $\text{enabled}_N(N^\times) = \emptyset$.
3. $\text{post}_N(t_i) \cap \text{post}_N(t_j) = \emptyset$ and $t_i \neq t_j$, for all $1 \leq i < j \leq k$.
4. If $M_k \cap \text{pre}_N(t) \neq \emptyset$ then:
 - there is $1 \leq i \leq k$ such that $t \in \text{enabled}_N(M_i)$, or
 - there is a marking M reachable from M_k such that $t \in \text{enabled}_N(M)$.

Proof. Different parts follow by induction on the structure of a box expression (see also [2]). ◀

¹ Hence $t_1 \dots t_k$ is a firing sequence of N .

² Hence we can use set notation when dealing with the semantics of composite boxes.

We then proceed with the proof proper, using the same notations as in the formulation of Proposition 1. We first observe that $N^e = \text{box}(E)^e \cap P$ as well as $\text{cfl}_{\text{box}(E)}(p) = \text{cfl}_N(p)$ and $\text{csd}_{\text{box}(E)}(p) = \text{csd}_N(p)$, for every $p \in P$. Hence $\text{cfl}_{\text{box}(E)}(P_{\text{box}(E)}) = \text{cfl}_N(P)$ and $\text{csd}_{\text{box}(E)}(P_{\text{box}(E)}) = \text{csd}_N(P)$. Moreover, if

$$\text{box}(E)^e[t_1]_{\text{box}(E)}M_1 \dots M_{k-1}[t_k]_{\text{box}(E)}M_k$$

then $N^e[t_1]_N(M_1 \cap P) \dots (M_{k-1} \cap P)[t_k]_N(M_k \cap P)$ since N is a subnet of $\text{box}(E)$ with the same set of transitions (*).

We then observe that $\text{fseq}(N) = \text{fseq}(\text{box}(E))$. Indeed, the (\supseteq) inclusion follows from (*). If the (\subseteq) inclusion does not hold, then there is $t_1 \dots t_k t \in \text{fseq}(N) \setminus \text{fseq}(\text{box}(E))$ such that $t_1 \dots t_k \in \text{fseq}(\text{box}(E))$. Thus (also by (*)) there are markings M_0, \dots, M_k such that

$$\begin{aligned} & (\text{box}(E)^e =) M_0[t_1]_{\text{box}(E)}M_1 \dots M_{k-1}[t_k]_{\text{box}(E)}M_k \\ & (N^e =) M_0 \cap P[t_1]_N(M_1 \cap P) \dots (M_{k-1} \cap P)[t_k]_N(M_k \cap P). \end{aligned}$$

We have, $t \in \text{enabled}_N(M_k \cap P) \setminus \text{enabled}_{\text{box}(E)}(M_k)$. Hence there is $p = \pi_{U,W} \in \text{pre}_{\text{box}(E)}(t) \setminus \text{pre}_N(t)$ such that $M_k(p) = 0$. On the other hand, since $\text{pre}_N(t) \neq \emptyset$ and $t \in \text{enabled}_N(M_k \cap P)$, $\text{pre}_{\text{box}(E)}(t) \cap M_k \neq \emptyset$. Hence, by Lemma 5(4), one of the following two cases holds:

Case 1: There is $1 \leq i \leq k$ such that $t \in \text{enabled}_{\text{box}(E)}(M_i)$ and so $p \in M_i$. Then there is $i < j \leq k$ such that $p \in \text{pre}_{\text{box}(E)}(t_j)$. By $\text{cfl}_N(P) = \text{cfl}_{\text{box}(E)}(P_{\text{box}(E)})$, there is $p' \in P$ such that $M_k(p') = 1$ and $p' \in \text{pre}_{\text{box}(E)}(t) \cap \text{pre}_{\text{box}(E)}(u)$. However, by Lemma 5(1), this means that p' must have been filled with a token twice along the firing sequence $t_1 \dots t_k$, contradicting Lemma 5(3).

Case 2: There is a marking M reachable from M_k such that $t \in \text{enabled}_{\text{box}(E)}(M)$. This means $p \in M$, and there is $u \in U$ which is executed when reaching M from M_k . Then $(u, t) \in \text{csd}_{\text{box}(E)}(p)$, and so, by $\text{csd}_{\text{box}(E)}(P) = \text{csd}_{\text{box}(E)}(P_{\text{box}(E)})$, there is $p' \in P$ such that $(u, t) \in \text{csd}_{\text{box}(E)}(p')$. Thus $p' \in M_k$. As a result, p' must have received a token twice along a firing sequence of $\text{box}(E)$ leading to M , contradicting Lemma 5(3).

Hence $\text{fseq}(N) = \text{fseq}(\text{box}(E))$ (**). Moreover, by (*), if σ and σ' are firing sequences leading in $\text{box}(E)$ to the same marking, then they also lead to the same marking in N . Suppose then that σ and σ' are firing sequences leading in N to the same marking. Then, by Lemma 5(2), they can be extended by the same σ'' to yield firing sequences $\sigma\sigma''$ and $\sigma'\sigma''$ leading to the marking N^\times . Thus, by (**), $\sigma\sigma''$ and $\sigma'\sigma''$ lead to some markings M and M' in $\text{box}(E)$. If, for example, $\sigma\sigma'' \neq \text{box}^\times$ then, by Lemma 5(2), it can be extended by a nonempty σ''' to lead to $\text{box}(E)^\times$. However, contradicting (*), σ''' cannot be fired from N^\times . Hence, $M = M'$, and so σ and σ' lead to the same marking in $\text{box}(E)$. This and the deterministic nature of $\text{RG}(\text{box})$ and $\text{RG}(N)$ means that the two reachability graphs are isomorphic.

C Proofs of Propositions 2 and 3

Proposition 2 follows by a straightforward induction on the structure of E . Propositions 3 follows directly from the definitions and Proposition 2.

D Proof of Proposition 4

We first shows that interface graphs are what we need to have in order to characterise local conflict and causalities.

8:16 Slimming down Petri Boxes: Compact Petri Net Models of Control Flows

► **Lemma 6.** *Let E and F be box expressions derived from the syntax (1) with disjoint sets of actions. Then:*

$$\begin{aligned}
 \text{Confl}(\text{box}(E; F)) &= \text{Confl}(\text{box}(E)) \cup \text{Confl}(\text{box}(F)) \\
 \text{Caused}(\text{box}(E; F)) &= \text{Caused}(\text{box}(E)) \cup \text{Confl}(\text{box}(F)) \cup \\
 &\quad \{\{t, u\} \in T_{\text{box}(E)} \times T_{\text{box}(F)} \mid \{t, u\} \in \text{edge}(G_E^x \text{---} G_F^e)\} \\
 \text{Confl}(\text{box}(E \square F)) &= \text{Confl}(\text{box}(E)) \cup \text{Confl}(\text{box}(F)) \cup \\
 &\quad \{\{t, u\} \in T_{\text{box}(E)} \times T_{\text{box}(F)} \mid \{t, u\} \in \text{edge}(G_E^e \text{---} G_F^e)\} \\
 \text{Caused}(\text{box}(E \square F)) &= \text{Caused}(\text{box}(E)) \cup \text{Caused}(\text{box}(F)) \\
 \text{Confl}(\text{box}(E \parallel F)) &= \text{Confl}(\text{box}(E)) \cup \text{Confl}(\text{box}(F)) \\
 \text{Caused}(\text{box}(E \parallel F)) &= \text{Caused}(\text{box}(E)) \cup \text{Caused}(\text{box}(F))
 \end{aligned}$$

Proof. The result follows directly from the definitions. ◀

We then proceed with the proof proper, using the same notations as in the formulation of Proposition 4. The first part follows directly from Proposition 1(\Leftarrow), Propositions 2 and 3, and Lemma 6. The second part follows from the minimality of covers used in the construction of N and Proposition 1(\Rightarrow).

On the Sequential Probability Ratio Test in Hidden Markov Models

Oscar Darwin 

Department of Computer Science, Oxford University, UK

Stefan Kiefer 

Department of Computer Science, Oxford University, UK

Abstract

We consider the Sequential Probability Ratio Test applied to Hidden Markov Models. Given two Hidden Markov Models and a sequence of observations generated by one of them, the Sequential Probability Ratio Test attempts to decide which model produced the sequence. We show relationships between the execution time of such an algorithm and Lyapunov exponents of random matrix systems. Further, we give complexity results about the execution time taken by the Sequential Probability Ratio Test.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains; Mathematics of computing → Stochastic processes; Theory of computation → Logic and verification

Keywords and phrases Markov chains, hidden Markov models, probabilistic systems, verification

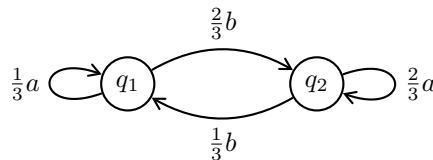
Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.9

Related Version *Full Version:* <https://arxiv.org/abs/2207.14088>

Acknowledgements The authors thank anonymous referees for valuable suggestions.

1 Introduction

A (discrete-time, finite-state) *Hidden Markov Model (HMM)* (often called *labelled Markov chain*) has a finite set Q of states and for each state a probability distribution over its possible successor states. Every state is associated with a probability transition over a successor state and an emitted letter (*observation*). For example, consider the following HMM:



In state q_1 , the probability of emitting a and the next state being also q_1 is $\frac{1}{3}$, and the probability of emitting b and the next state being q_2 is $\frac{2}{3}$. An HMM is typically viewed as a producer of a finite or infinite word of emitted observations. For example, starting in q_1 , the probability of producing a word with prefix aba is $\frac{1}{3} \cdot \frac{2}{3} \cdot \frac{2}{3}$, whereas starting in q_2 , the probability of aba is $\frac{2}{3} \cdot \frac{1}{3} \cdot \frac{1}{3}$. The random sequence of states is considered not observable (which explains the term *hidden* in HMM).

HMMs are widely employed in fields such as speech recognition (see [28] for a tutorial), gesture recognition [6], signal processing [10], and climate modeling [1]. HMMs are heavily used in computational biology [14], more specifically in DNA modeling [8] and biological sequence analysis [13], including protein structure prediction [22] and gene finding [3]. In computer-aided verification, HMMs are the most fundamental model for probabilistic systems; model-checking tools such as Prism [23] or Storm [12] are based on analyzing HMMs efficiently.



© Oscar Darwin and Stefan Kiefer;

licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 9; pp. 9:1–9:16

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

One of the most fundamental questions about HMMs is whether two initial distributions are *(trace) equivalent*, i.e., generate the same distribution on infinite observation sequences. In the example above, we argued that (the Dirac distributions on) the states q_1, q_2 are not equivalent. The equivalence problem is very well studied and can be solved in polynomial time using algorithms that are based on linear algebra [29, 25, 31, 9]. The equivalence problem has applications in verification, e.g., of randomised anonymity protocols [20].

Equivalence is a strong notion, and a natural question about nonequivalent distributions in a given HMM is *how* different they are. For initial distributions π_1, π_2 on the states of the HMM, let us write $\mathbb{P}_{\pi_1}, \mathbb{P}_{\pi_2}$ for the induced probability measure on infinite observation sequences; i.e., $\mathbb{P}_{\pi_i}(E)$, for a measurable event $E \subseteq \Sigma^\omega$, is the probability that the random infinite word $w \in \Sigma^\omega$ produced starting from π_i is in E . Then, the *total variation distance* between $\mathbb{P}_{\pi_1}, \mathbb{P}_{\pi_2}$ is defined as

$$d(\pi_1, \pi_2) := \sup \{ |\mathbb{P}_{\pi_1}(E) - \mathbb{P}_{\pi_2}(E)| \mid \text{measurable } E \subseteq \Sigma^\omega \}.$$

This supremum is a maximum; i.e., there always exists a “maximizing event” $E \subseteq \Sigma^\omega$ with $d(\pi_1, \pi_2) = \mathbb{P}_{\pi_1}(E) - \mathbb{P}_{\pi_2}(E)$. In these terms, initial distributions π_1, π_2 are equivalent if and only if $d(\pi_1, \pi_2) = 0$. The total variation distance was studied in more detail in [7]. There it was shown that the problem whether $d(\pi_1, \pi_2) = 1$ holds can also be decided in polynomial time. Call distributions π_1, π_2 *distinguishable* if $d(\pi_1, \pi_2) = 1$. Distinguishability was used for runtime monitoring [21] and diagnosability [4, 2] of stochastic systems.

Distributions π_1, π_2 that are distinguishable (i.e., $d(\pi_1, \pi_2) = 1$) can nevertheless be “hard” to distinguish. In our example above, (the Dirac distributions on) q_1, q_2 are distinguishable. If we replace the transition probabilities $\frac{1}{3}, \frac{2}{3}$ in the HMM by $\frac{1}{2} - \varepsilon, \frac{1}{2} + \varepsilon$, respectively, states q_1, q_2 remain distinguishable for every $\varepsilon > 0$, although, intuitively, the smaller $\varepsilon > 0$ the more observations are needed to define an event E such that $\mathbb{P}_{\pi_1}(E) - \mathbb{P}_{\pi_2}(E)$ is close to 1.

To make this more precise, for initial distributions π_1, π_2 , a word $w \in \Sigma^\omega$ and $n \in \mathbb{N}$ consider the *likelihood ratio*

$$L_n(w) := \frac{\mathbb{P}_{\pi_1}(w_n \Sigma^\omega)}{\mathbb{P}_{\pi_2}(w_n \Sigma^\omega)},$$

where w_n denotes the length- n prefix of w . In the example above, we argued that $\mathbb{P}_{q_1}(aba \Sigma^\omega) = \frac{1}{3} \cdot \frac{2}{3} \cdot \frac{2}{3}$ and $\mathbb{P}_{q_2}(aba \Sigma^\omega) = \frac{2}{3} \cdot \frac{1}{3} \cdot \frac{1}{3}$. Thus, for any word w starting with aba we have $L_n(w) = 2$. We consider the likelihood ratio L_n as a random variable for every $n \in \mathbb{N}$. It turns out more natural to focus on the *log-likelihood ratio* $\ln L_n$. One can show that the limit $\lim_{n \rightarrow \infty} \ln L_n \in [-\infty, \infty]$ exists \mathbb{P}_{π_1} -almost surely and \mathbb{P}_{π_2} -almost surely (see, e.g., [7, Proposition 6]). In fact, if π_1, π_2 are distinguishable, then $\lim_{n \rightarrow \infty} \ln L_n = \infty$ holds \mathbb{P}_{π_1} -almost surely and $\lim_{n \rightarrow \infty} \ln L_n = -\infty$ holds \mathbb{P}_{π_2} -almost surely. This suggests the “average slope”, $\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n$, of increase or decrease of $\ln L_n$ as a measure of *how* distinguishable two distinguishable distributions π_1, π_2 are.

The log-likelihood ratio plays a central role in the *sequential probability ratio test (SPRT)* [32], which is optimal [33] among sequential hypothesis tests (such tests attempt to decide between two hypotheses without fixing the sample size in advance). In terms of an HMM and two initial distributions π_1, π_2 , the SPRT attempts to decide, given longer and longer prefixes of an observation sequence $w \in \Sigma^\omega$, which of π_1, π_2 is more likely to emit w . The SPRT works as follows: fix a lower and an upper threshold (which determine type-I and type-II errors); given increasing prefixes of w keep track of $\ln L_n(w)$, and when the upper threshold is crossed output π_1 and stop, and when the lower threshold is crossed output π_2 and stop. Again, it is natural to assume that the average slope of increase or decrease of $\ln L_n$ determines how long the SPRT needs to cross one of the thresholds.

If the average slope $\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n$ exists and equals a number ℓ with positive probability, we call ℓ a *likelihood exponent*. The term is motivated by a close relationship to *Lyapunov exponents*, which characterise the growth rate of certain random matrix products. As the most fundamental contribution of this paper, we show that the average slope exists almost surely and that any HMM with m states has at most $m^2 + 1$ likelihood exponents.

The rest of the paper is organised as follows. In Section 3 we exhibit a tight connection between the SPRT and likelihood exponents; i.e., the time taken by the SPRT depends on the likelihood exponents of the HMM. This connection motivates our results on likelihood exponents in the rest of the paper. In Section 4 we prove complexity results concerning the probability that the average slope equals a particular likelihood exponent. In Section 5 we show that the average slope exists almost surely and prove our bound on the number of likelihood exponents. Further, we show that the likelihood exponents can be efficiently expressed in terms of Lyapunov exponents. In Section 6 we show that for *deterministic* HMMs one can compute likelihood exponents in polynomial time. We conclude in Section 7.

2 Preliminaries

We write \mathbb{N} for the set of non-negative integers. For $d \in \mathbb{N}$ we write $[d] = \{1, \dots, d\}$. For a finite set Q , vectors $\mu \in \mathbb{R}^Q$ are viewed as row vectors, and their transpose (a column vector) is denoted by μ^\top . The norm $\|\mu\|$ is assumed to be the l_1 norm: $\|\mu\| = \sum_{q \in Q} |\mu_q|$. We write $\vec{0}, \vec{1}$ for the vectors all whose entries are 0, 1, respectively. For $q \in Q$, we denote by $e_q \in \{0, 1\}^Q$ the vector with $(e_q)_q = 1$ and $(e_q)_{q'} = 0$ for $q' \neq q$. A matrix $M \in [0, 1]^{Q \times Q}$ is *stochastic* if $\vec{1}^\top = M\vec{1}^\top$. We often identify vectors $\mu \in [0, 1]^Q$ such that $\|\mu\| = 1$ with the corresponding probability distribution on Q . For $\mu \in [0, \infty)^Q$ we write $\text{supp}(\mu) := \{q \in Q \mid \mu_q > 0\}$.

For a finite alphabet Σ and $n \in \mathbb{N}$ we denote by $\Sigma^n, \Sigma^*, \Sigma^\omega$ the sets of length- n words, finite words, infinite words, respectively. For $w \in \Sigma^\omega$ we write w_n for the length- n prefix of w .

A *Hidden Markov Model* (HMM) is a triple $\mathcal{H} = (Q, \Sigma, \Psi)$ where Q is a finite set of states, Σ is a set of observations (or “letters”), and the function $\Psi : \Sigma \rightarrow [0, 1]^{Q \times Q}$ specifies the transitions such that $\sum_{a \in \Sigma} \Psi(a)$ is stochastic. A *Markov chain* is a pair (Q, T) where Q is a finite set of states and $T \in [0, 1]^{Q \times Q}$ is a stochastic matrix. A Markov chain (Q, T) is naturally associated with its directed *graph* $(Q, \{(q, r) \mid T_{q,r} > 0\})$, and so we may use graph concepts, such as strongly connected components (SCCs), in the context of a Markov chain. Trivial SCCs are considered SCCs. The *embedded* Markov chain of an HMM (Q, Σ, Ψ) is the Markov chain $(Q, \sum_{a \in \Sigma} \Psi(a))$. We say that an HMM is *strongly connected* if the graph of its embedded Markov chain is.

► **Example 1.** The HMM from the introduction is the triple $\mathcal{H} = (\{q_1, q_2\}, \{a, b\}, \Psi)$ with $\Psi(a) = \begin{pmatrix} \frac{1}{3} & 0 \\ 0 & \frac{2}{3} \end{pmatrix}$ and $\Psi(b) = \begin{pmatrix} 0 & \frac{2}{3} \\ \frac{1}{3} & 0 \end{pmatrix}$. The embedded Markov chain is $(\{q_1, q_2\}, \begin{pmatrix} \frac{1}{3} & \frac{2}{3} \\ \frac{1}{3} & \frac{2}{3} \end{pmatrix})$.

Fix an HMM $\mathcal{H} = (Q, \Sigma, \Psi)$ for the rest of the section. We extend Ψ to the mapping $\Psi : \Sigma^* \rightarrow [0, 1]^{Q \times Q}$ with $\Psi(a_1 \cdots a_n) = \Psi(a_1) \cdots \Psi(a_n)$ and $\Psi(\varepsilon) = I$, where ε is the empty word and I the $Q \times Q$ identity matrix. We call a finite sequence $v = q_0 a_1 q_1 \cdots a_n q_n \in Q(\Sigma Q)^*$ a *path* and $v(\Sigma Q)^\omega$ a *cylinder set* and an infinite sequence $q_0 a_1 q_1 a_2 q_2 \cdots \in Q(\Sigma Q)^\omega$ a *run*. To \mathcal{H} and an *initial probability distribution* $\pi \in [0, 1]^Q$ we associate the probability space $(Q(\Sigma Q)^\omega, \mathcal{G}^*, \mathbb{P}_\pi)$ where \mathcal{G}^* is the σ -algebra generated by the cylinder sets and \mathbb{P}_π is the unique probability measure with $\mathbb{P}_\pi(q_0 a_1 q_1 \cdots a_n q_n(\Sigma Q)^\omega) = \pi_{q_0} \prod_{i=1}^n \Psi(a_i)_{q_{i-1}, q_i}$. As the states are often irrelevant, for $E \subseteq \Sigma^\omega$ and $E^\uparrow := \{q_0 a_1 q_1 a_2 q_2 \cdots \mid a_1 a_2 \cdots \in E\} \in \mathcal{G}^*$ we view also E as an event and may write $\mathbb{P}_\pi(E)$ to mean $\mathbb{P}_\pi(E^\uparrow)$. In particular, for $w \in \Sigma^*$ we

have $\mathbb{P}_\pi(w\Sigma^\omega) = \|\pi\Psi(w)\|$. For $E \subseteq \Sigma^\omega$ we write $\mathbb{1}_E$ for the indicator random variable with $\mathbb{1}_E(w) = 1$ if $w \in E$ and $\mathbb{1}_E(w) = 0$ if $w \notin E$. By \mathbb{E}_π we denote the expectation with respect to \mathbb{P}_π . If π is the Dirac distribution on state q , then we write \mathbb{E}_q .

A Markov chain (Q, T) and an initial distribution $\nu \in [0, 1]^Q$ are associated with a probability measure \mathbb{P}_ν on measurable subsets of Q^ω ; the construction of the probability space is similar to HMMs, without the observation alphabet Σ .

Let (Q, Σ, Ψ) be an HMM and let π_1, π_2 be two initial distributions. The *total variation distance* is $d(\pi_1, \pi_2) := \sup_{E \uparrow \in \mathcal{G}^*} |\mathbb{P}_{\pi_1}(E) - \mathbb{P}_{\pi_2}(E)|$. This supremum is actually a maximum due to Hahn’s decomposition theorem; i.e., there is an event $E \subseteq \Sigma^\omega$ such that $d(\pi_1, \pi_2) = \mathbb{P}_{\pi_1}(E) - \mathbb{P}_{\pi_2}(E)$. We call π_1 and π_2 *distinguishable* if $d(\pi_1, \pi_2) = 1$. Distinguishability is decidable in polynomial time [7].

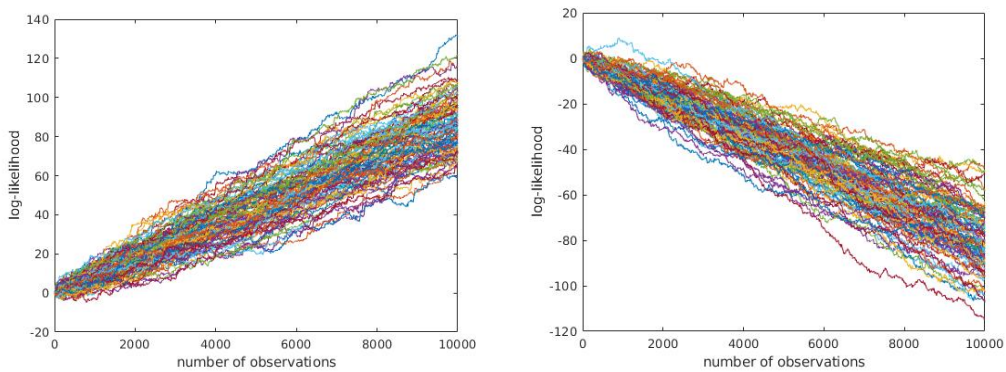
Let π_1 and π_2 be initial distributions. For $n \in \mathbb{N}$, the *likelihood ratio* L_n is a random variable on Σ^ω given by $L_n(w) = \frac{\|\pi_1\Psi(w_n)\|}{\|\pi_2\Psi(w_n)\|}$. Based on results from [7] we have the following lemma.

► **Lemma 2.** *Let π_1, π_2 be initial distributions.*

1. $\lim_{n \rightarrow \infty} L_n$ exists \mathbb{P}_{π_2} -almost surely and lies in $[0, \infty)$.
2. $\lim_{n \rightarrow \infty} L_n = 0$ \mathbb{P}_{π_2} -almost surely if and only if π_1 and π_2 are distinguishable.

► **Example 3.** We illustrate convergence of the likelihood ratio using an example from [24] where the authors use HMMs to model sleep cycles. They took measurements of 51 healthy and 51 diseased individuals and using electrodes attached to the scalp, they read electrical signal data as part of an electroencephalography (EEG) during sleep. They split the signal into 30 second intervals and mapped each interval onto the simplex $\Delta^3 = \{(x_1, x_2, x_3, x_4) \in [0, 1]^4 \mid \sum_{i=1}^4 x_i = 1\}$. For each individual this results in a time series of points in Δ^3 . They modelled this data using two HMMs, each with 5 states, for healthy and diseased individuals using a numerical maximum likelihood estimate. Each state is associated with a probability density function describing the distribution of observations in Δ^3 . We describe in [11] how we obtained from this an HMM $\mathcal{H} = (Q, \Sigma, \Psi)$ with (finite) observation alphabet $\Sigma = \{a_1, \dots, a_5\}$ and two initial distributions π_1, π_2 corresponding to healthy and diseased individuals, respectively. Using the algorithm from [7] one can show that π_1 and π_2 are distinguishable.

We sampled runs of \mathcal{H} started from π_1 and π_2 and plotted the corresponding sequences of $\ln L_n$. We refer to each of these two plots as a *log-likelihood plot*; see Figure 1.



■ **Figure 1** The two images show two log-likelihood plots of sample runs produced by π_1 and π_2 , respectively.

By Lemma 2.2 it follows that $\ln L_n$ converges \mathbb{P}_{π_1} -a.s. (almost-surely) to ∞ and \mathbb{P}_{π_2} -a.s. to $-\infty$. This is affirmed by Figure 1. Both log-likelihood plots also appear to follow a particular slope. This suggests that we can distinguish between words produced by π_1 and π_2 by tracking the value of $\ln L_n$ to see whether it crosses a lower or upper threshold. This is the intuition behind the *Sequential Probability Ratio Test* (SPRT).

3 Sequential Probability Ratio Test

Fix an HMM $H = (Q, \Sigma, \Psi)$ for the rest of the paper. Given initial distributions π_1, π_2 and error bounds $\alpha, \beta \in (0, 1)$, the SPRT runs as follows. It continues to read observations and computes the value of $\ln L_n$ until $\ln L_n$ leaves the interval $[A, B]$, where $A := \ln \frac{\alpha}{1-\beta}$ and $B := \ln \frac{1-\alpha}{\beta}$. If $\ln L_n \leq A$ the test outputs “ π_2 ” and if $\ln L_n \geq B$ the test outputs “ π_1 ”. We may view the SPRT as a random variable $\text{SPRT}_{\alpha, \beta} : \Sigma^\omega \rightarrow \{\pi_1, \pi_2, ?\}$, where ? denotes that the SPRT does not terminate, i.e., $\ln L_n \in [A, B]$ for all n . We have the following correctness property.

► **Proposition 4.** *Suppose π_1 and π_2 are distinguishable. Let $\alpha, \beta \in (0, 1)$. By choosing $A = \ln \frac{\alpha}{1-\beta}$ and $B = \ln \frac{1-\alpha}{\beta}$, we have $\mathbb{P}_{\pi_1}(\text{SPRT}_{\alpha, \beta} = \pi_2) \leq \alpha$ and $\mathbb{P}_{\pi_2}(\text{SPRT}_{\alpha, \beta} = \pi_1) \leq \beta$.*

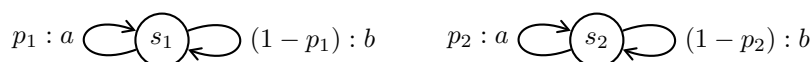
In the following we consider the SPRT with respect to the measure \mathbb{P}_{π_2} . This is without loss of generality as there is a dual version of the SPRT, say $\overline{\text{SPRT}}$ with $\overline{L}_n = 1/L_n$ instead of L_n , such that $\overline{\text{SPRT}}_{\beta, \alpha} = \text{SPRT}_{\alpha, \beta}$. Define the stopping time

$$N_{\alpha, \beta} := \min\{n \in \mathbb{N} \mid \ln L_n \notin [A, B]\} \in \mathbb{N} \cup \{\infty\}.$$

We have that $N_{\alpha, \beta}$ is monotone decreasing in the sense that for $\alpha \leq \alpha'$ and $\beta \leq \beta'$ we have $N_{\alpha, \beta} \geq N_{\alpha', \beta'}$. When π_1 and π_2 are distinguishable, $N_{\alpha, \beta}$ is \mathbb{P}_{π_2} -a.s. finite by Lemma 2.2.

3.1 Expectation of $N_{\alpha, \beta}$

Consider the two-state HMM where $p_1 \neq p_2$.



(The Dirac distributions of) s_1 and s_2 are distinguishable. Further, the increments $\ln L_{n+1} - \ln L_n$ are independent and identically distributed (i.i.d.) and $0 > \mathbb{E}_{s_2}[\ln L_{n+1} - \ln L_n] = p_2 \ln \frac{p_1}{p_2} + (1 - p_2) \ln \frac{1-p_1}{1-p_2} =: \ell$. Intuitively as ℓ gets more negative, the HMMs become more different.¹ Indeed, Wald [32] shows that the expected stopping time $\mathbb{E}_{s_2}[N_{\alpha, \beta}]$ and ℓ are inversely proportional:

$$\mathbb{E}_{s_2}[N_{\alpha, \beta}] = \frac{\beta \ln \frac{1-\alpha}{\beta} + (1 - \beta) \ln \frac{\alpha}{1-\beta}}{\ell}. \quad (1)$$

This Wald formula cannot hold in general for (multi-state) HMMs. The increments $\ln L_{n+1} - \ln L_n$ need not be independent and $\mathbb{E}_{s_2}[\ln L_{n+1} - \ln L_n]$ can be different for different n . Further, $|\ln L_{n+1} - \ln L_n|$ can be unbounded; cf. [21, Example 6].

¹ In fact, ℓ is the *KL-divergence* of the distributions f_1, f_2 where $f_i(a) = p_i$ and $f_i(b) = 1 - p_i$ for $i = 1, 2$.

Nevertheless, in Figure 1 we observed that $\ln L_n$ appears to decrease linearly (on the π_2 plot). Indeed, we show in Theorem 8 below that the limit $\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n$ exists \mathbb{P}_{π_2} -almost surely. Intuitively it corresponds to the average slope of the log-likelihood plot for π_2 . In the two-state case, there is a simple proof of this using the law of large numbers:

$$\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n = \lim_{n \rightarrow \infty} \frac{1}{n} \sum_{i=0}^{n-1} [\ln L_{i+1} - \ln L_i] = \mathbb{E}_{\pi_2}[\ln L_1 - \ln L_0] = \ell \quad \mathbb{P}_{\pi_2}\text{-a.s.}$$

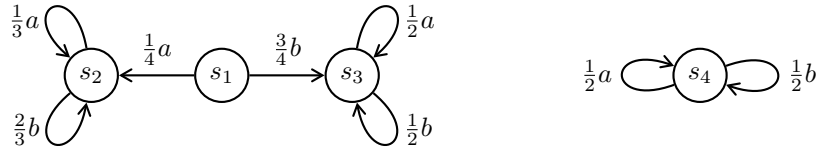
The number ℓ is called a likelihood exponent, as defined generally in the following definition.

► **Definition 5.** For initial distributions π_1, π_2 , a number $\ell \in [-\infty, 0]$ is a likelihood exponent if $\mathbb{P}_{\pi_2}(\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n = \ell) > 0$.

By Lemma 2.1 we have $\mathbb{P}_{\pi_2}(\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n > 0) = 0$, as $\mathbb{P}_{\pi_2}(\lim_{n \rightarrow \infty} L_n < \infty) = 1$. Hence, we may restrict likelihood exponents to $[-\infty, 0]$. We write $\Lambda_{\pi_1, \pi_2} \subseteq [-\infty, 0]$ for the set of likelihood exponents for π_1, π_2 and define $\Lambda := \bigcup_{\pi_1, \pi_2} \Lambda_{\pi_1, \pi_2}$; i.e., Λ depends only on the HMM \mathcal{H} . For $\ell \in \Lambda$ we define the event $E_\ell = \{\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n = \ell\}$.

► **Example 6.** In the case of Example 3 we have $\Lambda_{\pi_1, \pi_2} = \{\ell\}$ where the slope of the right hand side of Figure 1 suggests that $\ell \approx -\frac{80}{10000} = -0.008$.

► **Example 7.** Even for fixed π_1, π_2 there may be multiple likelihood exponents. Consider the following HMM with initial Dirac distributions $\pi_1 = e_{s_1}$ and $\pi_2 = e_{s_4}$.



We observe two different likelihood exponents depending on the first letter produced. If the first letter is a then $\ln L_{n+1} - \ln L_n$ are i.i.d. for $n \geq 1$ and $\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n = \frac{1}{2} \ln \frac{1/3}{1/2} + \frac{1}{2} \ln \frac{2/3}{1/2} = \frac{1}{2} \ln \frac{8}{9} =: \ell$ like the two-state example above. If the first letter is b then $L_n = \frac{3}{2}$ for all $n \geq 1$ and $\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n = 0$. Thus, $\Lambda_{\pi_1, \pi_2} = \{\ell, 0\}$ and $\mathbb{P}_{\pi_2}(E_\ell) = \mathbb{P}_{\pi_2}(E_0) = \frac{1}{2}$.

The following theorem is perhaps the most fundamental contribution of this paper.

► **Theorem 8.** For any initial distributions π_1, π_2 the limit $\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n$ exists \mathbb{P}_{π_2} -almost surely. Furthermore, we have $|\Lambda| \leq |Q|^2 + 1$.

It follows from a stronger theorem, Theorem 23, which we prove in Section 5.

Returning to the SPRT, we investigate how $\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n$ influences the performance of the SPRT for small α and β . Intuitively we expect a steeper slope in the likelihood plot (cf. Figure 1) to lead to faster termination. In the two-state case, Wald's formula (1) becomes

$$\mathbb{E}_{s_2}[N_{\alpha, \beta}] = \frac{\beta \ln \frac{1-\alpha}{\beta} + (1-\beta) \ln \frac{\alpha}{1-\beta}}{\ell} \sim \frac{\ln \alpha}{\ell} \quad (\text{as } \alpha, \beta \rightarrow 0), \quad (2)$$

where we use the notation \sim defined as follows. For functions $f, g: (0, \infty) \times (0, \infty) \rightarrow (0, \infty)$ we write " $f(x, y) \sim g(x, y)$ (as $x, y \rightarrow 0$)" to denote that for all $\varepsilon > 0$ there is $\delta > 0$ such that for all $x, y \in (0, \delta)$ we have $f(x, y)/g(x, y) = [1 - \varepsilon, 1 + \varepsilon]$.

In Theorem 9 below we generalise Equation (2) to arbitrary HMMs. Indeed a very similar asymptotic identity holds. In the case that $\Lambda = \{\ell\}$ and $\ell \in (-\infty, 0)$ we have $\mathbb{E}_{s_2}[N_{\alpha, \beta}] \sim \frac{\ln \alpha}{\ell}$ as $\alpha, \beta \rightarrow 0$. If $|\Lambda| > 1$ then we condition our expectation on $\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n$.

► **Theorem 9** (Generalised Wald Formula). *Let ℓ be a likelihood exponent and let π_1 and π_2 be initial distributions.*

1. *If $\ell \in (-\infty, 0)$ then $\mathbb{E}_{\pi_2}[N_{\alpha,\beta} \mid E_\ell] \sim \frac{\ln \alpha}{\ell}$ (as $\alpha, \beta \rightarrow 0$).*
2. *If $\ell = 0$ then there exist $\alpha, \beta > 0$ such that $\mathbb{E}_{\pi_2}[N_{\alpha,\beta} \mid E_\ell] = \infty$.*
3. *If $\ell = -\infty$ then $\sup_{\alpha,\beta} \mathbb{E}_{\pi_2}[N_{\alpha,\beta} \mid E_\ell] < \infty$.*

The theorem above pertains to the expectation of $N_{\alpha,\beta}$. In the next subsection we give additional information about the distribution of $N_{\alpha,\beta}$, further strengthening the connection between $N_{\alpha,\beta}$ and likelihood exponents.

3.2 Distribution of $N_{\alpha,\beta}$

3.2.1 Likelihood Exponent 0

► **Example 10.** We continue with Example 7 to illustrate the second case in Theorem 9. By picking $\alpha = \frac{1}{4}, \beta = \frac{1}{4}$ the thresholds for the SPRT are $A = \ln \frac{1}{3}$ and $B = \ln 3$. If the first letter is b , then $\ln L_n = \ln \frac{3}{2}$ for all $n > 1$, thus never crosses the SPRT bounds and $\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n = 0$. Hence with probability $\frac{1}{2}$ the SPRT fails to terminate and $N_{\alpha,\beta} = \infty$. It follows that $\mathbb{P}_{\pi_2}(E_0) = \frac{1}{2}$ and $\mathbb{E}_{\pi_2}[N_{\alpha,\beta} \mid E_0] = \infty$ and, thus, $\mathbb{E}_{\pi_2}[N_{\alpha,\beta}] = \infty$. The second part of Theorem 9 says that the expectation of $N_{\alpha,\beta}$ conditioned under E_0 is infinite. The following proposition strengthens this statement. Conditioning under E_0 , the probability that $N_{\alpha,\beta}$ is infinite converges to 1 as $\alpha, \beta \rightarrow 0$. Recall that $N_{\alpha,\beta}$ is monotone decreasing. It follows that $\{N_{\alpha',\beta'} = \infty\} \subseteq \{N_{\alpha,\beta} = \infty\}$ if $\alpha \leq \alpha'$ and $\beta \leq \beta'$.

► **Proposition 11.** *The following two equalities hold up to \mathbb{P}_{π_2} -null sets:*

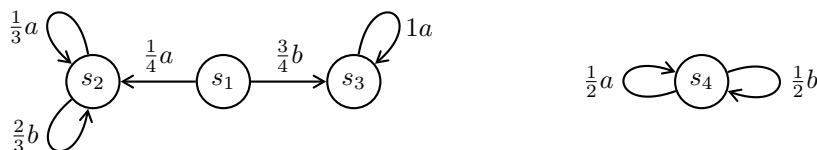
$$E_0 = \left\{ \lim_{n \rightarrow \infty} L_n > 0 \right\} = \bigcup_{\alpha,\beta > 0} \{N_{\alpha,\beta} = \infty\}.$$

Thus, $\lim_{\alpha,\beta \rightarrow 0} \mathbb{P}_{\pi_2}(N_{\alpha,\beta} = \infty) = \mathbb{P}_{\pi_2}(E_0)$.

► **Corollary 12** (using Lemma 2.2). *Initial distributions π_1 and π_2 are distinguishable if and only if $\mathbb{P}_{\pi_2}(E_0) = 0$ if and only if $\mathbb{P}_{\pi_2}(N_{\alpha,\beta} < \infty) = 1$ holds for all $\alpha, \beta > 0$.*

3.2.2 Likelihood Exponent $-\infty$

► **Example 13.** Consider now a modification of Example 7 where state s_3 has the b loop removed.



The likelihood exponents are $-\infty$ and $\ell := \frac{1}{2} \ln \frac{8}{9}$ so that $\Lambda = \{-\infty, \ell\}$. Also, $\mathbb{P}_{s_4}(E_{-\infty}) = \mathbb{P}_{s_4}(E_\ell) = \frac{1}{2}$. Up to \mathbb{P}_{s_4} -null sets the events $E_{-\infty}$, $b\Sigma^\omega$ and $ba^*b\Sigma^\omega$ are equal. The event $ba^*b\Sigma^\omega$ represents the right chain producing an observation which the left chain cannot produce, causing the SPRT to terminate for any α, β . Therefore conditioned on $E_{-\infty}$, the random variable $N_{\alpha,\beta} - 1$ is bounded by a geometric random variable with parameter $\frac{1}{2}$. Hence $\sup_{\alpha,\beta} \mathbb{E}_{\pi_2} [N_{\alpha,\beta} \mid E_{-\infty}] \leq 1 + 2$.

We define the stopping time $N_\perp = \min\{n \in \mathbb{N} \mid L_n = 0\}$. Note that $\sup_{\alpha,\beta} N_{\alpha,\beta} \leq N_\perp$ since $\{L_n = 0\} \subseteq \{L_n \leq \frac{\alpha}{1-\beta}\}$ for all α, β . By the following proposition, the reverse inequality also holds.

► **Proposition 14.** *The events $E_{-\infty}$ and $\{L_n = 0 \text{ for some } n\}$ are equal. Thus, $\sup_{\alpha,\beta} N_{\alpha,\beta} = N_{\perp}$ and $\lim_{\alpha,\beta \rightarrow 0} \mathbb{P}_{\pi_2}(N_{\alpha,\beta} < \infty) = \mathbb{P}_{\pi_2}(E_{-\infty})$.*

Applying this to Example 13, we obtain $\sup_{\alpha,\beta} \mathbb{E}_{\pi_2}[N_{\alpha,\beta} \mid E_{-\infty}] = 3$.

3.2.3 Likelihood Exponent in $(-\infty, 0)$

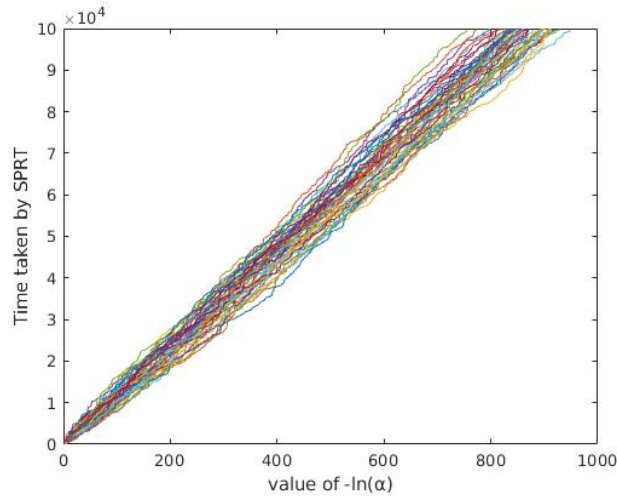
Conditioned on E_{ℓ} where $\ell \in (-\infty, 0)$, Theorem 9 states that $N_{\alpha,\beta}$ scales with $\frac{\ln \alpha}{\ell}$ in expectation. The following result shows that this relationship also holds \mathbb{P}_{π_2} -almost surely.

► **Proposition 15.** *Let $\ell \in \Lambda$ and assume $\ell \in (-\infty, 0)$. We have*

$$\mathbb{P}_{\pi_2}\left(N_{\alpha,\beta} \sim \frac{\ln \alpha}{\ell} \text{ (as } \alpha, \beta \rightarrow 0) \mid E_{\ell}\right) = 1.$$

In fact, we prove the first part of Theorem 9 using Proposition 15. If there were a bound $M \in \mathbb{N}$ such that \mathbb{P}_{π_2} -a.s. $\frac{N_{\alpha,\beta}}{-\ln \alpha} \leq M$, the first part of Theorem 9 would follow from Proposition 15 by the dominated convergence theorem. However this is not the case in general. Instead we show in [11] that the set of random variables $\{\frac{N_{\alpha,\beta}}{-\ln \alpha} \mid 0 < \alpha, \beta \leq \frac{1}{2}\}$ is uniformly integrable with respect to the measure \mathbb{P}_{π_2} and then use Vitali's convergence theorem.

► **Example 16.** Recall Example 3, where $\Lambda = \{\ell\}$. Figure 2 demonstrates the asymptotic



■ **Figure 2** The time taken by the SPRT for $0 \leq -\ln \alpha = -\ln \beta \leq 1000$.

relationship in Proposition 15. Each of the 50 lines correspond to a sample run and we record the value of $N_{\alpha,\beta}$ for $0 \leq -\ln \alpha = -\ln \beta \leq 1000$. From the figure we estimate $-\frac{1}{\ell}$ as $\frac{10^5}{800} = 125$. This coincides with the estimate given in Example 6.

We conclude from this section that the performance of the SPRT, in terms of its termination time $N_{\alpha,\beta}$, is tightly connected to likelihood exponents. This motivates our study of likelihood exponents in the rest of the paper.

4 Probability of E_{ℓ}

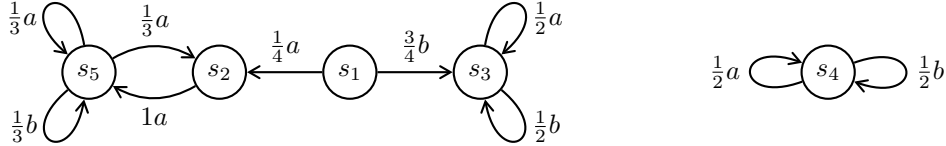
In this section we aim at computing $\mathbb{P}_{\pi_2}(E_{\ell})$ for a likelihood exponent ℓ . We show the following theorem.

► **Theorem 17.** *Given an HMM and initial distributions π_1, π_2 ,*

1. *one can compute $\mathbb{P}_{\pi_2}(E_{-\infty})$ and $\mathbb{P}_{\pi_2}(E_0)$ in PSPACE;*
2. *one can decide whether $\mathbb{P}_{\pi_2}(E_0) = 0$ (i.e., $0 \notin \Lambda_{\pi_1, \pi_2}$) in polynomial time;*
3. *deciding whether $\mathbb{P}_{\pi_2}(E_0) = 1$, whether $\mathbb{P}_{\pi_2}(E_{-\infty}) = 0$, and whether $\mathbb{P}_{\pi_2}(E_{-\infty}) = 1$ are all PSPACE-complete problems.*

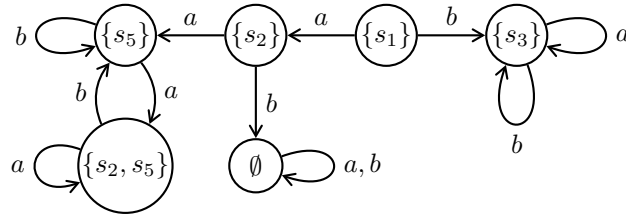
The following example illustrates the construction underlying the PSPACE upper bound.

► **Example 18.** Consider another adaption of Example 7.



If the first letter produced by s_4 is b , then $L_n = \frac{3}{2}$ for all $n \in \mathbb{N}$. If the first two letters are ab , then $L_1 = \frac{1}{2}$ and $L_n = 0$ for $n \geq 2$. If the first two letters are aa , then $s_5 \in \text{supp}(e_{s_1}\Psi(aaw))$ for all $w \in \Sigma^*$, and therefore, up to a \mathbb{P}_{s_4} -null set, $L_n > 0$ holds for all $n \in \mathbb{N}$, which implies (using Proposition 14) that there is $\ell \in (-\infty, 0)$ such that $\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n = \ell$. Thus, $\Lambda_{s_1, s_4} = \{-\infty, \ell, 0\}$.

The likelihood ratio L_n is 0 if and only if $\text{supp}(\pi_1\Psi(w_n)) = \emptyset$. In order to track the support of $\pi_1\Psi(w_n)$, we consider the left part of the HMM as an NFA with s_1 as the initial state and its determinisation as shown in the DFA below.



Almost surely, s_4 produces a word that drives this DFA into a bottom SCC, which then determines $\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n$: concretely, the bottom SCC $\{\{s_5\}, \{s_2, s_5\}\}$ is associated with ℓ , the bottom SCC $\{\emptyset\}$ with $-\infty$, and the bottom SCC $\{\{s_3\}\}$ with 0.

In general, the observations need not be produced uniformly at random but by an HMM. Therefore, in the following construction, we also keep track of the “current” state of the HMM which produces the observations. For $S \subseteq Q$ and $a \in \Sigma$, define $\delta(S, a) := \{q' \in Q \mid \exists q \in S : \Psi(a)_{q, q'} > 0\}$. Define the Markov chain $\mathcal{B} := (2^Q \times Q, T)$ where

$$T_{(S, q), (S', q')} := \sum_{\delta(S, a) = S'} \Psi(a)_{q, q'}.$$

Given initial distributions π_1, π_2 on Q as before, define an initial distribution ι on $2^Q \times Q$ by $\iota(\text{supp}(\pi_1), q) := (\pi_2)_q$. Intuitively, the left part S of a state (S, q) tracks the support of $\pi_1\Psi(w_n)$, and the right part q tracks the current state of the HMM that had been initialised at a random state from π_2 . The following lemma states the key properties of this construction.

► **Lemma 19.** *Consider the Markov chain $\mathcal{B} = (2^Q \times Q, T)$ defined above.*

1. *Every bottom SCC of \mathcal{B} is associated with a single likelihood exponent; i.e., for every bottom SCC $C \subseteq 2^Q \times Q$ there is $\ell(C) \in [-\infty, 0]$ such that for any initial distribution $\pi_1 \in [0, 1]^Q$ and any state $q_2 \in Q$ with $(\text{supp}(\pi_1), q_2) \in C$ we have $\Lambda_{\pi_1, e_{q_2}} = \{\ell(C)\}$.*

2. Let $(S, q) \in C$ for a bottom SCC C . If $S = \emptyset$ then $\ell(C) = -\infty$; otherwise, if e_q and the uniform distribution on S are not distinguishable then $\ell(C) = 0$; otherwise $\ell(C) \in (-\infty, 0)$.
3. We have $\mathbb{P}_{\pi_2}(E_\ell) = \mathbb{P}_i(\{\text{visit bottom SCC } C \text{ with } \ell(C) = \ell\})$.

All parts of the lemma rely on the observation that $\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n$ depend only on the support of π_1 and on the support of π_2 . The first part of the lemma follows from Lévy's 0-1 law. We use this lemma for the proof of Theorem 17.1.

Proof sketch for Theorem 17.1. The Markov chain \mathcal{B} from Lemma 19 is exponentially big but can be constructed by a PSPACE transducer, i.e., a Turing machine whose work tape (but not necessarily its output tape) is PSPACE-bounded. This PSPACE transducer can also identify the bottom SCCs. For each bottom SCC C , the PSPACE transducer also decides whether $\ell(C) = -\infty$ or $\ell(C) \in (-\infty, 0)$ or $\ell(C) = 0$, using Lemma 19.2 and the polynomial-time algorithm for distinguishability from [7]. Finally, to compute $\mathbb{P}_{\pi_2}(E_{-\infty})$ and $\mathbb{P}_{\pi_2}(E_0)$, by Lemma 19.3, it suffices to set up and solve a linear system of equations for computing hitting probabilities in a Markov chain. This system can also be computed by a PSPACE transducer. Since linear systems of equations can be solved in the complexity class NC, which is included in polylogarithmic space, one can use standard techniques for composing space-bounded transducers to compute $\mathbb{P}_{\pi_2}(E_{-\infty})$ and $\mathbb{P}_{\pi_2}(E_0)$ in PSPACE. ◀

Proof of Theorem 17.2. Immediate from Corollary 12 and the polynomial-time decidability of distinguishability [7]. ◀

Towards a proof of Theorem 17.3, we use the *mortality* problem, which asks, given a finite set of states Q , a finite alphabet Σ , and a function $\Phi : \Sigma \rightarrow \{0, 1\}^{Q \times Q}$, whether there exists a word $w \in \Sigma^*$ such that $\Phi(w)$ is the zero matrix. The mortality problem can be viewed as a special case of the NFA non-universality problem (given an NFA, does it reject some word?). Like NFA universality, the mortality problem is PSPACE-complete [19].

Concerning $\mathbb{P}_{\pi_2}(E_{-\infty})$ (cf. Theorem 17.3), we actually show a stronger result, namely that any nontrivial approximation of $\mathbb{P}_{\pi_2}(E_{-\infty})$ is PSPACE-hard. The proof is also based on the mortality problem.

► **Proposition 20.** *There is a polynomial-time computable function that maps any instance of the mortality problem to an HMM and initial distributions π_1, π_2 so that if the instance is positive then $\mathbb{P}_{\pi_2}(E_{-\infty}) = 1$ and if the instance is negative then $\mathbb{P}_{\pi_2}(E_{-\infty}) = 0$. Thus, any nontrivial approximation of $\mathbb{P}_{\pi_2}(E_{-\infty})$ is PSPACE-hard.*

Proof. Let (Q, Σ, Φ) be an instance of the mortality problem. If there is $q \in Q$ that indexes a zero row in $\sum_{a \in \Sigma} \Phi(a)$, remove the row and column indexed by q in all $\Phi(a)$. Thus, we can assume without loss of generality that $\sum_{a \in \Sigma} \Phi(a)$ has no zero row. Construct an HMM (Q, Σ, Ψ) so that $\Phi(a)$ and $\Psi(a)$ have the same zero pattern for all $a \in \Sigma$. Define π_1 as a uniform distribution on Q . Define π_2 as a Dirac distribution on a fresh state that emits letters from Σ uniformly at random. Thus, if (Q, Σ, Φ) is a positive instance of the mortality problem then $\mathbb{P}_{\pi_2}(E_{-\infty}) = 1$, and if (Q, Σ, Φ) is a negative instance then $\mathbb{P}_{\pi_2}(E_{-\infty}) = 0$. ◀

The proof that deciding whether $\mathbb{P}_{\pi_2}(E_0) = 1$ is PSPACE-hard is similarly based on mortality.

5 Representing Likelihood Exponents

In the following we show that one can efficiently represent likelihood exponents in terms of *Lyapunov exponents*. The definition of Lyapunov exponents is based on the following definition.

► **Definition 21.** A matrix system is a triple $\mathcal{M} = (Q, \Sigma, \Psi)$ where Q is a finite set of states, Σ is a finite set of observations, and $\Psi : \Sigma \rightarrow \mathbb{R}_{\geq 0}^{Q \times Q}$ specifies the transitions. (Note that an HMM is a matrix system.) A Lyapunov system is a pair $\mathcal{S} = (\mathcal{M}, \rho)$ where $\mathcal{M} = (Q, \Sigma, \Psi)$ is a matrix system and $\rho \in (0, 1]^\Sigma$ is a probability distribution with full support, such that the directed graph (Q, E) with $E = \{(q, r) \mid \sum_{a \in \Sigma} \Psi_{q,r}(a) > 0\}$ is strongly connected.

We can identify the probability distribution ρ from this definition with the single-state HMM $(\{s\}, \Sigma, \Psi_\rho)$ where $\Psi_\rho(a)_{s,s} = \rho(a)$ for all $a \in \Sigma$. In this way, ρ produces a random infinite word from Σ^ω . The following lemma is known from [26].

► **Lemma 22** ([26]). Let $((Q, \Sigma, \Psi), \rho)$ be a Lyapunov system. Then there is $\lambda \in \mathbb{R}$ such that, for all $q \in Q$, \mathbb{P}_ρ -a.s., either $e_q \Psi(w_n) = \vec{0}$ for some $n \in \mathbb{N}$ or the limit $\lim_{n \rightarrow \infty} \frac{1}{n} \ln \|e_q \Psi(w_n)\|$ exists and equals λ .

For a Lyapunov system \mathcal{S} we call $\lambda(\mathcal{S}) = \lambda$ from the lemma the *Lyapunov exponent* defined by \mathcal{S} . We prove the following theorem, which implies Theorem 8.

► **Theorem 23.** Given an HMM (Q, Σ, Ψ) we can compute in polynomial time $2K \leq 2|Q|^2$ Lyapunov systems $\mathcal{S}_1^1, \mathcal{S}_1^2, \mathcal{S}_2^1, \mathcal{S}_2^2, \dots, \mathcal{S}_K^1, \mathcal{S}_K^2$ such that for any initial distributions π_1, π_2 the limit $\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n$ exists \mathbb{P}_{π_2} -a.s. and lies in

$$\Lambda \subseteq \{-\infty\} \cup \{\lambda(\mathcal{S}_1^1) - \lambda(\mathcal{S}_1^2), \dots, \lambda(\mathcal{S}_K^1) - \lambda(\mathcal{S}_K^2)\}.$$

In particular, the HMM (Q, Σ, Ψ) has at most $|Q|^2 + 1$ likelihood exponents.

In the rest of the section we provide more details on the construction underlying Theorem 23. As an intermediate concept (between the given HMM and the Lyapunov systems from Theorem 23) we define *generalized Lyapunov systems*.

First, for two matrix systems $\mathcal{M}_1 = (Q_1, \Sigma, \Psi_1)$ and $\mathcal{M}_2 = (Q_2, \Sigma, \Psi_2)$ with finite Q_1, Q_2, Σ and transitions $\Psi_1, \Psi_2 : \Sigma \rightarrow \mathbb{R}_{\geq 0}^{Q \times Q}$ we define the directed graph $G_{\mathcal{M}_1, \mathcal{M}_2} = (Q_1 \times Q_2, E)$ such that there is an edge from (q_1, q_2) to (r_1, r_2) if there is $a \in \Sigma$ with $\Psi_1(a)_{q_1, r_1} > 0$ and $\Psi_2(a)_{q_2, r_2} > 0$.

A *generalized Lyapunov system* is a triple $\mathcal{S} = (\mathcal{M}, \mathcal{H}, C)$ where $\mathcal{M} = (Q_1, \Sigma, \Psi_1)$ is a matrix system and $\mathcal{H} = (Q_2, \Sigma, \Psi_2)$ is a strongly connected HMM and $C \subseteq Q_1 \times Q_2$ is a bottom SCC of $G_{\mathcal{M}, \mathcal{H}}$. Given a generalized Lyapunov system, one can efficiently compute an “equivalent” Lyapunov system:

- **Lemma 24.** Let $\mathcal{S} = ((Q_1, \Sigma, \Psi_1), (Q_2, \Sigma, \Psi_2), C)$ be a generalized Lyapunov system.
1. There is $\lambda \in \mathbb{R}$, henceforth called $\lambda(\mathcal{S})$, such that, for all $\pi_1 \in [0, \infty)^{Q_1}$ and all probability distributions $\pi_2 \in [0, 1]^{Q_2}$ with $\text{supp}(\pi_1) \times \text{supp}(\pi_2) \subseteq C$, we have \mathbb{P}_{π_2} -a.s. that either $\pi_1 \Psi_1(w_n) = \vec{0}$ for some $n \in \mathbb{N}$ or the limit $\lim_{n \rightarrow \infty} \frac{1}{n} \ln \|\pi_1 \Psi_1(w_n)\|$ exists and equals $\lambda(\mathcal{S})$.
 2. One can compute in polynomial time a Lyapunov system \mathcal{S}' such that $\lambda(\mathcal{S}) = \lambda(\mathcal{S}')$.

Let $\mathcal{H} = (Q, \Sigma, \Psi)$ be an HMM. Let $R \subseteq Q \times Q$ be a (not necessarily bottom) SCC of the graph $G_{\mathcal{H}, \mathcal{H}}$ such that $Q_R := \{q_2 \in Q \mid \exists q_1 \in Q : (q_1, q_2) \in R\}$ is a bottom SCC of the graph of $\sum_{a \in \Sigma} \Psi(a)$. We call such R a *right-bottom* SCC. Clearly there are at most

9:12 On the Sequential Probability Ratio Test in Hidden Markov Models

$|Q|^2$ right-bottom SCCs. Towards Theorem 23 we want to define, for each right-bottom SCC R , two generalized Lyapunov systems $\mathcal{S}_R^1, \mathcal{S}_R^2$. Intuitively, \mathcal{S}_R^1 and \mathcal{S}_R^2 correspond to the numerator and the denominator of the likelihood ratio, respectively.

For a function of the form $\Phi : \Sigma \rightarrow \mathbb{R}^{Q \times Q}$ and $P \subseteq Q$ we write $\Phi|_P : \Sigma \rightarrow \mathbb{R}^{P \times P}$ for the function with $\Phi|_P(a)(q, r) = \Phi(a)(q, r)$ for all $a \in \Sigma$ and $q, r \in P$; i.e., $\Phi|_P(a)$ denotes the principal submatrix obtained from $\Phi(a)$ by restricting it to the rows and columns indexed by P .

Define $\Psi'(a, r)_{q,r} := \Psi(a)_{q,r}$ for all $a \in \Sigma$ and $q, r \in Q$. Then $(Q, \Sigma \times Q, \Psi')$ is an HMM, which is similar to \mathcal{H} , but which emits, in addition to an observation from Σ , also the next state. Since Q_R is a bottom SCC of the graph of $\sum_{a \in \Sigma} \Psi(a)$, the HMM $\mathcal{H}_2 := (Q_R, \Sigma \times Q_R, \Psi'|_{Q_R})$ is strongly connected. This HMM \mathcal{H}_2 will be used both in \mathcal{S}_R^1 and in \mathcal{S}_R^2 .

Next, define $\bar{\Psi} : (\Sigma \times Q) \rightarrow [0, 1]^{(Q \times Q) \times (Q \times Q)}$ by

$$\bar{\Psi}(a, r_2)_{(q_1, q_2), (r_1, r_2)} := \Psi(a)_{q_1, r_1} \quad \text{for all } a \in \Sigma \text{ and } q_1, q_2, r_1, r_2 \in Q.$$

Now define $\mathcal{S}_R^1 := (\mathcal{M}^1, \mathcal{H}_2, C^1)$, where $\mathcal{M}^1 := (R, \Sigma \times Q_R, \bar{\Psi}|_R)$ and $C^1 := \{((q_1, q_2), q_2) \mid (q_1, q_2) \in R\}$. Finally, denoting by $R' \subseteq Q_R \times Q_R$ the SCC of the graph $G_{\mathcal{H}, \mathcal{H}}$ that contains the ‘‘diagonal’’ vertices $(q, q) \in Q_R \times Q_R$, define $\mathcal{S}_R^2 := (\mathcal{M}^2, \mathcal{H}_2, C^2)$, where $\mathcal{M}^2 := (R', \Sigma \times Q_R, \bar{\Psi}|_{R'})$ and $C^2 := \{((q_1, q_2), q_2) \mid (q_1, q_2) \in R'\}$.

For sets $U, V \subseteq Q \times Q$ let $U \xrightarrow{G_{\mathcal{H}, \mathcal{H}}} V$ denote that there are $u \in U$ and $v \in V$ such that v is reachable from u in $G_{\mathcal{H}, \mathcal{H}}$. We are ready to state the following key technical lemma:

► **Lemma 25.** *Given an HMM (Q, Σ, Ψ) , let $\mathcal{R} \subseteq 2^{Q \times Q}$ be the set of its right-bottom SCCs, and, for $R \in \mathcal{R}$, let $\mathcal{S}_R^1, \mathcal{S}_R^2$ be the generalized Lyapunov systems defined above. Then, for any initial distributions π_1, π_2 , the limit $\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n$ exists \mathbb{P}_{π_2} -a.s. and lies in*

$$\{-\infty\} \cup \{\lambda(\mathcal{S}_R^1) - \lambda(\mathcal{S}_R^2) \mid R \in \mathcal{R}, \text{supp}(\pi_1) \times \text{supp}(\pi_2) \xrightarrow{G_{\mathcal{H}, \mathcal{H}}} R\}.$$

Thus, $\Lambda_{\pi_1, \pi_2} \subseteq \{-\infty\} \cup \{\lambda(\mathcal{S}_R^1) - \lambda(\mathcal{S}_R^2) \mid R \in \mathcal{R}, \text{supp}(\pi_1) \times \text{supp}(\pi_2) \xrightarrow{G_{\mathcal{H}, \mathcal{H}}} R\}$.

Proof sketch. Let π_1, π_2 be initial distributions. Very loosely speaking, we show in the appendix that on \mathbb{P}_{π_2} -almost every run w there is a right-bottom SCC R which ‘‘traps’’ ‘‘most’’ of the mass of $\pi_1 \Psi(w_n)$ and $\pi_2 \Psi(w_n)$. This can be made meaningful and formal using (the cross-product systems) $\mathcal{S}_R^1, \mathcal{S}_R^2$. We then show that on \mathbb{P}_{π_2} -almost every such run w , for both $i = 1, 2$, the limit $\lim_{n \rightarrow \infty} \frac{1}{n} \ln \|\pi_i \Psi(w_n)\|$ exists and equals $\lambda(\mathcal{S}_R^i)$ (or $\pi_i \Psi(w_n) = \vec{0}$ for some n). It follows that

$$\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n = \lim_{n \rightarrow \infty} \frac{1}{n} \ln \frac{\|\pi_1 \Psi(w_n)\|}{\|\pi_2 \Psi(w_n)\|} = \lambda(\mathcal{S}_R^1) - \lambda(\mathcal{S}_R^2). \quad \blacktriangleleft$$

With Lemma 25 at hand, the proof of Theorem 23 is easy:

Proof of Theorem 23. As argued before, the set \mathcal{R} of right-bottom SCCs of the given HMM has at most $|Q|^2$ elements. These right-bottom SCCs R and the associated generalized Lyapunov systems $\mathcal{S}_R^1, \mathcal{S}_R^2$ can be computed in polynomial time. By Lemma 25 we have $\Lambda = \bigcup_{\pi_1, \pi_2} \Lambda_{\pi_1, \pi_2} \subseteq \{-\infty\} \cup \{\lambda(\mathcal{S}_R^1) - \lambda(\mathcal{S}_R^2) \mid R \in \mathcal{R}\}$. By Lemma 24.2, for each $R \in \mathcal{R}$ one can compute in polynomial time an equivalent Lyapunov system. ◀

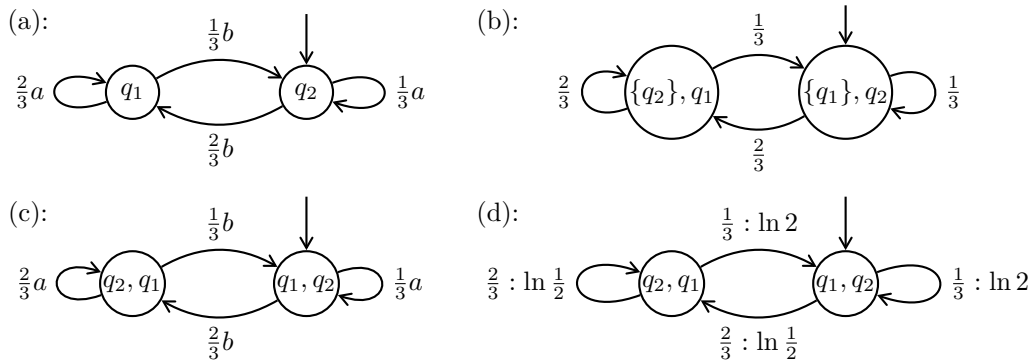
Theorem 23 allows us to represent the likelihood exponents of an HMM in terms of Lyapunov exponents. In general, approximating or even computing Lyapunov exponents is hard, but there are practical approximation algorithms using convex optimisation [27, 30].

6 Deterministic HMMs

In Sections 4 and 5 we have seen that the problems of representing/computing likelihood exponents and of computing their probabilities tend to be computationally difficult. In this section we study *deterministic* HMMs and show that this subclass leads to tractable problems. An HMM (Q, Σ, Ψ) is *deterministic* if, for all $a \in \Sigma$, all rows of $\Psi(a)$ contain at most one non-zero entry. Thus, for all $q \in Q$ and $w \in \Sigma^*$, we have $|\text{supp}(e_q \Psi(w))| \leq 1$.

A useful observation is that the Markov chain $\mathcal{B} = (2^Q \times Q, T)$, which was defined before Lemma 19 and can be exponential in general, has only quadratic size in the deterministic case if we restrict it to the part that is reachable from initial Dirac distributions.

► **Example 26.** Consider the deterministic HMM (Q, Σ, Ψ) in Figure 3(a). Let $\pi_1 = e_{q_1}$



■ **Figure 3** Cross-product constructions for a deterministic HMM.

and $\pi_2 = e_{q_2}$ (the latter is indicated by an arrow pointing to q_2). Then the relevant (i.e., reachable from $(\{q_1\}, q_2)$) part of \mathcal{B} is shown in Figure 3(b). Let us add back the observations that gave rise to the transitions in \mathcal{B} , and for simplicity drop the set brackets in the left component of states. We obtain the HMM in Figure 3(c). With this HMM we may keep track of the exact likelihood ratio. For example, suppose that the word aba is emitted, so that $L_3 = \frac{\|e_{q_1} \Psi(aba)\|}{\|e_{q_2} \Psi(aba)\|} = \frac{1}{2}$ and $\text{supp}(e_{q_1} \Psi(aba)) = \{q_2\}$ and $\text{supp}(e_{q_2} \Psi(aba)) = \{q_1\}$. Suppose the next letter is b (which is the case with probability $\frac{1}{3}$). Then L_4 arises from L_3 by multiplying with $\frac{\Psi_{q_2, q_1}(b)}{\Psi_{q_1, q_2}(b)} = 2$, and the supports are switched again. In terms of log-likelihoods, we have $\ln L_4 = \ln L_3 + \ln 2$. This motivates the Markov chain shown in Figure 3(d), where the transitions outgoing from a state (r_1, r_2) are labelled by the log-likelihood ratio of their corresponding probabilities in the HMM. The Markov chain has stationary distribution $(\frac{2}{3}, \frac{1}{3})$. By the strong ergodic theorem for Markov chains, we obtain (the irrational number)

$$\lim_{n \rightarrow \infty} \frac{1}{n} \ln L_n = \frac{2}{3} \left(\frac{2}{3} \ln \frac{1}{2} + \frac{1}{3} \ln 2 \right) + \frac{1}{3} \left(\frac{1}{3} \ln 2 + \frac{2}{3} \ln \frac{1}{2} \right) = \frac{1}{3} \ln 2 + \frac{2}{3} \ln \frac{1}{2} = -\frac{1}{3} \ln 2.$$

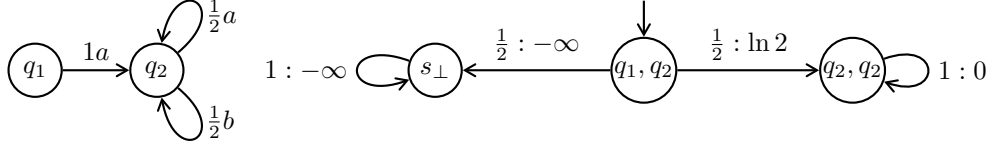
9:14 On the Sequential Probability Ratio Test in Hidden Markov Models

In general there may again be several likelihood exponents, including $-\infty$ and 0 . For the rest of the section, let $\mathcal{H} = (Q, \Sigma, \Psi)$ be a deterministic HMM. Motivated by Example 26, define an HMM $\mathcal{A} = ((Q \times Q) \cup s_\perp, \hat{\Sigma}, \hat{\Psi})$, where s_\perp is a fresh state, and

$$\begin{aligned}\hat{\Sigma} &:= \left\{ \ln \frac{\Psi(a)_{q_1, r_1}}{\Psi(a)_{q_2, r_2}} \in [-\infty, \infty) \mid a \in \Sigma, q_1, r_1, q_2, r_2 \in Q, \Psi(a)_{q_2, r_2} \neq 0 \right\} \cup \{-\infty\} \\ \hat{\Psi}(\hat{a})_{(q_1, q_2), (r_1, r_2)} &:= \sum \left\{ \Psi(a)_{q_2, r_2} \mid a \in \Sigma : \hat{a} = \ln \frac{\Psi(a)_{q_1, r_1}}{\Psi(a)_{q_2, r_2}} \right\} \quad \text{for } \hat{a} \neq -\infty \\ \hat{\Psi}(-\infty)_{(q_1, q_2), s_\perp} &:= \sum \left\{ \Psi(a)_{q_2, r_2} \mid a \in \Sigma, r_2 \in Q : \sum_{r_1 \in Q} \Psi(a)_{q_1, r_1} = 0 \right\} \\ \hat{\Psi}(-\infty)_{s_\perp, s_\perp} &:= 1.\end{aligned}$$

Note that the embedded Markov chain of \mathcal{A} is similar to the Markov chain \mathcal{B} from Lemma 19: states $(\{q_1\}, q_2)$ in \mathcal{B} are called (q_1, q_2) in \mathcal{A} , the states (\emptyset, q) in \mathcal{B} are subsumed by the state s_\perp of \mathcal{A} , and the states (S, q) in \mathcal{B} with $|S| > 1$ are not represented in \mathcal{A} . The observations in $\hat{\Sigma} \subseteq [-\infty, \infty)$ track the log-likelihood ratio.

► **Example 27.** Consider the HMM \mathcal{H} on the left, with initial distributions $\pi_1 = e_{q_1}$ and $\pi_2 = e_{q_2}$. The part of \mathcal{A} reachable from (q_1, q_2) is shown on the right:



Here we have $\Lambda_{\pi_1, \pi_2} = \{-\infty, 0\}$ with $\mathbb{P}_{\pi_2}(E_{-\infty}) = \mathbb{P}_{\pi_2}(E_0) = \frac{1}{2}$.

Denote by $\bar{\mathcal{A}}$ the embedded Markov chain of \mathcal{A} . Let $C \subseteq Q \times Q$ be a non- $\{s_\perp\}$ bottom SCC of $\bar{\mathcal{A}}$. Let $\mu \in [0, 1]^C$ denote the stationary distribution of the restriction of $\bar{\mathcal{A}}$ on C . Define the vector $\nu \in \mathbb{R}^C$ of average observations by $\nu_{(r_1, r_2)} := \sum_{\hat{a} \in \hat{\Sigma}} \|e_{(r_1, r_2)} \hat{\Psi}(\hat{a})\| \cdot \hat{a}$. By the strong ergodic theorem for Markov chains, the *average observation* in C equals $\mu \nu^\top =: \ell(C)$. Extend this definition by $\ell(\{s_\perp\}) := -\infty$. Then we have the following lemma.

► **Lemma 28.** *Let $\pi_1 = e_{q_1}$ and $\pi_2 = e_{q_2}$ be initial distributions. For the Markov chain $\bar{\mathcal{A}}$ define $\iota := e_{(q_1, q_2)}$. We have $\mathbb{P}_{\pi_2}(E_\ell) = \mathbb{P}_\iota(\{\text{visit bottom SCC } C \text{ with } \ell(C) = \ell\})$.*

The proof is essentially the same as in Lemma 19.3. This gives us the following result.

► **Theorem 29.** *Given a deterministic HMM (Q, Σ, Ψ) with initial Dirac distributions π_1, π_2 , one can compute in polynomial time*

1. Λ_{π_1, π_2} as a set of expressions of the form $\sum_i x_i \ln y_i$ where $x_i, y_i \in \mathbb{Q}$, and
2. $\Pr_{\pi_2}(E_\ell)$ for each such $\ell \in \Lambda_{\pi_1, \pi_2}$.

Proof sketch. The theorem follows mostly from Lemma 28, with the slight complication that for part 2 we have to check numbers of the form $\sum_i x_i \ln y_i$ (where $x_i, y_i \in \mathbb{Q}$) for equality. But this can be done in polynomial time as shown in [15]. ◀

7 Conclusions

We have shown that the performance of the SPRT is tightly connected with likelihood exponents. These numbers are related to Lyapunov exponents and can be viewed as a distance measure between HMMs. We have shown that the number of likelihood exponents is quadratic in the number of states. The associated computational problems tend to be

complex (PSPACE-hard), but become tractable for deterministic HMMs. In our work we did not make any ergodicity assumptions on the HMMs, unlike in earlier works from mathematics and engineering such as [18, 5, 16, 17]. Efficient approximation of likelihood exponents, in theory or praxis, remains an open problem.

References

- 1 P. Ailliot, C. Thompson, and P. Thomson. Space-time modelling of precipitation by using a hidden Markov model and censored Gaussian distributions. *Journal of the Royal Statistical Society*, 58(3):405–426, 2009.
- 2 S. Akshay, H. Bazille, E. Fabre, and B. Genest. Classification among hidden Markov models. In *Proceedings of the Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 150 of *LIPICs*, pages 29:1–29:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.FSTTCS.2019.29.
- 3 M. Alexandersson, S. Cawley, and L. Pachter. SLAM: Cross-species gene finding and alignment with a generalized pair hidden Markov model. *Genome Research*, 13:469–502, 2003.
- 4 N. Bertrand, S. Haddad, and E. Lefaucheu. Accurate approximate diagnosability of stochastic systems. In *Proceedings of Language and Automata Theory and Applications (LATA)*, volume 9618 of *Lecture Notes in Computer Science*, pages 549–561. Springer, 2016. doi:10.1007/978-3-319-30000-9_42.
- 5 B. Chen and P. Willett. Detection of hidden Markov model transient signals. *IEEE Transactions on Aerospace and Electronic Systems*, 36(4):1253–1268, 2000. doi:10.1109/7.892673.
- 6 F.-S. Chen, C.-M. Fu, and C.-L. Huang. Hand gesture recognition using a real-time tracking method and hidden Markov models. *Image and Vision Computing*, 21(8):745–758, 2003.
- 7 T. Chen and S. Kiefer. On the total variation distance of labelled Markov chains. In *Proceedings of the 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)*, pages 33:1–33:10, Vienna, Austria, 2014.
- 8 G.A. Churchill. Stochastic models for heterogeneous DNA sequences. *Bulletin of Mathematical Biology*, 51(1):79–94, 1989.
- 9 C. Cortes, M. Mohri, and A. Rastogi. L_p distance and equivalence of probabilistic automata. *International Journal of Foundations of Computer Science*, 18(04):761–779, 2007.
- 10 M.S. Crouse, R.D. Nowak, and R.G. Baraniuk. Wavelet-based statistical signal processing using hidden Markov models. *IEEE Transactions on Signal Processing*, 46(4):886–902, April 1998.
- 11 O. Darwin and S. Kiefer. On the sequential probability ratio test in hidden Markov models, 2022. arXiv:2207.14088.
- 12 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.
- 13 R. Durbin. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.
- 14 S.R. Eddy. What is a hidden Markov model? *Nature Biotechnology*, 22(10):1315–1316, October 2004.
- 15 K. Etesami, A. Stewart, and M. Yannakakis. A note on the complexity of comparing succinctly represented integers, with an application to maximum probability parsing. *ACM Trans. Comput. Theory*, 6(2):9:1–9:23, 2014. doi:10.1145/2601327.
- 16 C.-D. Fuh. SPRT and CUSUM in hidden Markov models. *The Annals of Statistics*, 31(3):942–977, 2003. doi:10.1214/aos/1056562468.
- 17 E. Grossi and M. Lops. Sequential detection of Markov targets with trajectory estimation. *IEEE Transactions on Information Theory*, 54(9):4144–4154, 2008. doi:10.1109/TIT.2008.928261.

- 18 B.-H. Juang and L. R. Rabiner. A probabilistic distance measure for hidden Markov models. *AT&T Technical Journal*, 64(2):391–408, 1985. doi:10.1002/j.1538-7305.1985.tb00439.x.
- 19 J.-Y. Kao, N. Rampersad, and J. Shallit. On NFAs where all states are final, initial, or both. *Theoretical Computer Science*, 410(47):5010–5021, 2009. doi:10.1016/j.tcs.2009.07.049.
- 20 S. Kiefer, A.S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. Language equivalence for probabilistic automata. In *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV)*, volume 6806 of *LNCS*, pages 526–540. Springer, 2011.
- 21 S. Kiefer and A.P. Sistla. Distinguishing hidden Markov chains. In *Proceedings of the 31st Annual Symposium on Logic in Computer Science (LICS)*, pages 66–75, New York, USA, 2016. ACM.
- 22 A. Krogh, B. Larsson, G. von Heijne, and E.L.L. Sonnhammer. Predicting transmembrane protein topology with a hidden Markov model: Application to complete genomes. *Journal of Molecular Biology*, 305(3):567–580, 2001.
- 23 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.
- 24 R. Langrock, B. Swihart, B. Caffo, N. Punjabi, and C. Crainiceanu. Combining hidden Markov models for comparing the dynamics of multiple sleep electroencephalograms. *Statistics in medicine*, 32, August 2013. doi:10.1002/sim.5747.
- 25 A. Paz. *Introduction to Probabilistic Automata (Computer Science and Applied Mathematics)*. Academic Press, Inc., Orlando, FL, USA, 1971.
- 26 V.Yu. Protasov. Asymptotics of products of nonnegative random matrices. *Functional Analysis and Its Applications*, 47:138–147, 2013.
- 27 V.Yu. Protasov and R.M. Jungers. Lower and upper bounds for the largest Lyapunov exponent of matrices. *Linear Algebra and its Applications*, 438(11):4448–4468, 2013. doi:10.1016/j.laa.2013.01.027.
- 28 L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
- 29 M.P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2):245–270, 1961.
- 30 D. Sutter, O. Fawzi, and R. Renner. Bounds on Lyapunov exponents via entropy accumulation. *IEEE Transactions on Information Theory*, 67(1):10–24, 2021. doi:10.1109/TIT.2020.3026959.
- 31 W.-G. Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM J. Comput.*, 21(2):216–227, April 1992.
- 32 A. Wald. Sequential Tests of Statistical Hypotheses. *The Annals of Mathematical Statistics*, 16(2):117–186, 1945. doi:10.1214/aoms/1177731118.
- 33 A. Wald and J. Wolfowitz. Optimum character of the sequential probability ratio test. *The Annals of Mathematical Statistics*, 19(3):326–339, 1948. URL: <http://www.jstor.org/stable/2235638>.


Parameter Synthesis for Parametric Probabilistic Dynamical Systems and Prefix-Independent Specifications

Christel Baier 

Technische Universität Dresden, Germany

Simon Jantsch 

Technische Universität Dresden, Germany

Engel Lefauchaux 

University of Lorraine, CNRS, Inria,
LORIA, Nancy, France

David Purser 

University of Warsaw, Poland

James Worrell 


Department of Computer Science,
University of Oxford, UK

Florian Funke 

Technische Universität Dresden, Germany

Toghrul Karimov 

Max Planck Institute for Software Systems,
Saarland Informatics Campus,
Saarbrücken, Germany

Joël Ouaknine¹ 

Max Planck Institute for Software Systems,
Saarland Informatics Campus,
Saarbrücken, Germany

Markus A. Whiteland 

University of Liège, Belgium

Abstract

We consider the model-checking problem for parametric probabilistic dynamical systems, formalised as Markov chains with parametric transition functions, analysed under the distribution-transformer semantics (in which a Markov chain induces a sequence of distributions over states).

We examine the problem of synthesising the set of parameter valuations of a parametric Markov chain such that the orbits of induced state distributions satisfy a prefix-independent ω -regular property.

Our main result establishes that in all non-degenerate instances, the feasible set of parameters is (up to a null set) semialgebraic, and can moreover be computed (in polynomial time assuming that the ambient dimension, corresponding to the number of states of the Markov chain, is fixed).

2012 ACM Subject Classification Theory of computation \rightarrow Logic and verification

Keywords and phrases Model checking, parametric Markov chains, distribution transformer semantics

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.10

Funding This work was funded by DFG grant 389792660 as part of TRR 248 – CPEC (see perspicuous-computing.science).

1 Introduction

The algorithmic analysis of Markov chains, in particular by means of model checking, is a central topic in probabilistic verification [7]. It is in fact fairly common to consider *parametric* Markov chains (PMCs), in which probabilities are given not as explicit numbers but rather as functions of certain parameters. One is then interested in the set of parameters giving rise to a Markov chain that meets a certain specification.

Markov chains are typically analysed under one of two standard semantics: the *path semantics* considers the set of all possible control-state trajectories, weighted by relevant probabilities, whereas the *distribution-transformer semantics* views the Markov chain as a

¹ Also affiliated with Keble College, Oxford as [emmy.network](https://www.emmy.network) Fellow.



single sequence of distributions over control states; this sequence is the orbit of the initial distribution under the repeated application of the underlying stochastic linear transformation. In this paper we focus exclusively on the second modelling paradigm, viewing Markov chains as special instances of linear dynamical systems (LDS). We consider parametric Markov chains, in which probabilities are given by rational functions over a set of parameters. Such parameters might account for uncertainties in the environment or in the exact values of the probabilities at hand, etc. Given a particular specification, we are interested in computing the set of all parameter valuations such that the resulting concrete Markov chain meets the specification. More precisely: *Given a parametric Markov chain \mathcal{M} over set of parameters X , i.e., a (parametric) matrix M and initial distribution π (see Definition 3), as well as a specification φ , compute the set of parameter instantiations $p \in \mathbb{R}^X$ for which the orbit $(\pi \cdot M[p]^n)_{n \geq 0}$ of \mathcal{M} satisfies φ .*

Our properties are specified with respect to the *characteristic word* of a Markov chain, which describes the orbit of the Markov chain relative to a set of targets. Given a Markov chain (Q, M, π) and a partition of the space $[0, 1]^Q = T_1 \cup \dots \cup T_k$, the characteristic word is the infinite word $w \in \{1, \dots, k\}^\omega$ such that $w_i = j$ if and only if $\pi \cdot M^i \in T_j$. For a parametric Markov chain, each admissible valuation of the parameters $p \in \mathbb{R}^X$ induces a concrete characteristic word $w[p]$. In this case we call $w : \mathbb{R}^X \rightarrow \{1, \dots, k\}^\omega$ the *parametric characteristic word* of the parametric Markov chain.

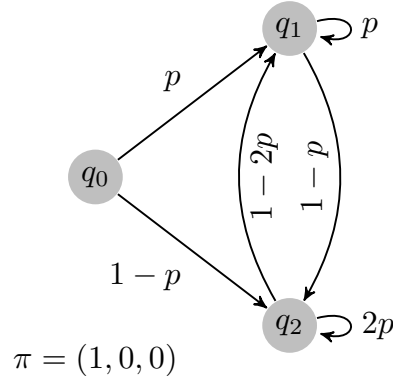
The model-checking problem asks, given a specification φ over $\{1, \dots, k\}$ (typically specified in LTL, MSO, or simply as an automaton), whether the characteristic word w satisfies φ , denoted by $w \models \varphi$. In the parametric setting, we are interested in the set of parameters $D_\varphi = \{p \in \mathbb{R}^X \mid w[p] \models \varphi\}$.

We consider ω -regular *prefix-independent* specifications, i.e., (intuitively speaking) properties that are invariant under finitely many changes to w (see Section 2.4 for the formal definition). The Ultimate Positivity Problem [22] is an example of a prefix-independent property: it asks whether the orbit is eventually trapped inside a certain target (chosen to be the region where a particular quantity is always positive). Other examples include repeatedly revisiting a target, since such a property only depends on any infinite suffix of w , regardless of the initial prefix. On the other hand, reachability is *not* a prefix-independent property. Note that LTL properties starting with “eventually always” or “always eventually” define prefix-independent properties (see [4]), although we do not limit ourselves to LTL properties.

Consider a parametric Markov chain with parametric characteristic word w and a prefix-independent specification φ . The set $D_\varphi = \{p \in \mathbb{R}^X \mid w[p] \models \varphi\}$ of feasible parameters can be a highly complex object. Nevertheless, one of our main results is that, assuming the specification is *non-degenerate* (a fairly mild technical condition), D_φ differs from a semialgebraic set by a null set (a set of Lebesgue measure zero), and moreover we can compute this semialgebraic set (in polynomial time assuming that the ambient dimension, corresponding to the number of states of the Markov chain, is fixed). More precisely, we show how to synthesise a semialgebraic set D' contained in the full set of feasible parameters such that $D_0 = D_\varphi \setminus D'$ is a null set.

Before going into the details of our construction, we show that the restriction to prefix-independent specifications is indeed necessary. Dropping it may lead to situations in which the set of feasible parameters is not semialgebraic, even up to a null set, as the following example shows.

► **Example 1** (A prefix-dependent property). Consider the parametric Markov chain depicted in Figure 1a, with the single parameter p . Let us denote by (l, r, s) a distribution over Q , with l, r, s denoting the probability in states q_L, q_R, q_S respectively. Consider the following



■ **Figure 2** Example Parametric Markov chain with parameter p .

In Appendix A we give a second example, which shows that reachability properties also do not have semialgebraic feasibility sets (up to a null set).

Let us consider another example, highlighting how we can use the limit distribution of an aperiodic Markov chain to decide prefix-independent properties.

► **Example 2 (Ultimate Positivity).** Consider the parametric Markov chain, depicted in Figure 2, with a single parameter p . The system represented by the diagram is a Markov chain for $p \in [0, 0.5]$ and has constant structure for all for $p \in (0, 0.5)$ (that is, each edge either exists for all p in the interval, or for none).

Consider the property that the probability distribution in states q_1 is eventually above 0.4 and q_2 is eventually above 0.55. We are interested in the set of parameters $D = \{p \in [0, 0.5] \mid \exists N \in \mathbb{N}. \forall n \geq N. \pi \cdot M[p]^n \geq (0, 0.4, 0.55)\}$.

The limit distribution of the Markov chain is $(0, \frac{1-2p}{2-3p}, \frac{1-p}{2-3p})$. Hence D , up to a null set, corresponds to the interval $(0, 0.5) \cap \{p \mid \frac{1-2p}{2-3p} > 0.4\} \cap \{p \mid \frac{1-p}{2-3p} > 0.55\} = (\frac{2}{13}, \frac{1}{4})$. Moreover all parameters in the interval $(\frac{2}{13}, \frac{1}{4})$ satisfy the property.

1.1 Related work

The papers [18, 19] introduce the logic iLTL to specify LTL-definable properties of the orbit of a Markov chain, where atomic propositions correspond to half-spaces. The authors devise a model-checking procedure which assumes that the Markov chain is aperiodic and diagonalisable, and that the unique limit distribution, which exists due to the aperiodicity condition, does not lie on the boundary of any of the half-spaces used to define the property. The paper [19] also presents case studies in the areas of software reliability and medicine. Our work extends these previous works in three directions: we consider parametric Markov chains, we allow the Markov chains to be periodic, and we allow semialgebraic sets as atomic propositions. Due to new difficulties that arise in the parametric setting we do not cover full LTL, rather we handle arbitrary prefix-independent ω -regular properties.

Agrawal et al. [1] consider the model-checking problem of Markov chains under the distribution-transformer semantics, where the target sets are specified as intervals on each component. They also remark that full ω -regular model checking will not be possible in general, and instead they consider whether an approximation of the trajectory satisfies a property.

In [17], a related problem for Markov decision processes (MDPs) is studied. The orbit of an MDP is not fixed but depends on the scheduler, and this additional feature often leads to undecidability. Several restrictions on schedulers and specifications are studied [17] under which decidability can be achieved. [10] considers a restriction on the MDP.

Markov chains under the distribution-transformer semantics are a special case of LDS. Presumably one is interested in expressive specifications, e.g., those that are specifiable in LTL or MSO. Unfortunately, even simple reachability queries for LDS are known to be extremely challenging [11], and the attendant hardness propagates to Markovian dynamical systems as well [2].

Baier et al. [5] showed that parametric *point-to-point reachability*, which asks whether there *exist* parameter choices under which a given state distribution is reachable, is decidable only for a single parameter, and Skolem-hard for two or more parameters. The problem is well-known to be decidable in polynomial time for LDS [16] (and thus for non-parametric Markov chains). We circumvent this limitation, allowing us to consider an arbitrary number of parameters, by synthesising, up to a null set, the set of parameter choices for which an arbitrary prefix-independent property holds (rather than reachability of a single point target).

Model checking prefix-independent properties on diagonalisable LDS is decidable [3] (see also [22] for Ultimate Positivity specifically). However, in general, the decidability status of the Ultimate Positivity Problem is a major open question – in fact, decidability of Ultimate Positivity for LDS of dimension 6 would entail major breakthroughs in number theory as it would solve certain longstanding open problems in Diophantine approximation of transcendental numbers that are widely believed to be hard [21].

Typically, only very few border cases are particularly difficult, and thus in the parametric setting such border cases amount to a null set which we can exclude. This is the case for all but degenerate instances in which *all* of the parameter valuations lead to such hard border cases. It is therefore necessary to impose a technical restriction on the expressible targets in order to exclude these degenerate instances.

The problem of model checking parametric Markov chains with respect to the standard trace semantics has been considered extensively [12, 20, 14, 6, 13]. In this setting one can express properties such as “the set of traces reaching a certain state has probability above λ ”, which can be described using standard logics such as PCTL [15, 7]. This semantics does not allow specifying properties such as “the probability of being in state s_1 is eventually larger than the probability of being in state s_2 ”, which can be expressed by the properties we consider.

2 Preliminaries

2.1 Parametric Markov chains

Given a set of variables X , we denote the field of rational functions over X with base field \mathbb{Q} by $\mathbb{Q}(X)$. We denote the set of all probability distributions over Q by $\text{Dist}(Q)$.

- **Definition 3.** A parametric Markov chain (PMC) is a tuple $\mathcal{M} = (Q, X, M, \pi)$, where
- Q is a finite set of states;
 - X is a finite set of variables, here typically called parameters;
 - $M \in \mathbb{Q}(X)^{Q \times Q}$ is the parametrised transition matrix;
 - $\pi \in \text{Dist}(Q)$ is an initial distribution.

Given a concrete instantiation $p \in \mathbb{R}^X$ of the parameters X , we denote by $M[p] \in \mathbb{R}^{Q \times Q}$ the matrix $M[p]_{s,t} = M_{s,t}(p)$, provided that $M_{s,t}(p)$ is defined for every $s, t \in Q$. That is, $M[p]$ is the concrete transition function obtained by replacing in M every occurrence of a

parameter $v \in X$ by the value assigned to v in p . We call $p \in \mathbb{R}^X$ *admissible* if $M[p]$ is a probabilistic transition function, i.e., $0 \leq M[p]_{s,t} \leq 1$ for all $s, t \in Q$, and $\sum_{t \in Q} M[p]_{s,t} = 1$ for all $s \in Q$. The Markov chain induced by the parameter value p will be denoted by $\mathcal{M}[p] = (Q, M[p], \pi)$. Finally, we remark that parametrised initial distributions can be encoded in our framework by adding a single state to the Markov chain that is visited only once at the beginning. The probabilities associated with the outgoing edges of the new start state are then used to simulate the parametrised initial probabilities.

2.2 The topological structure of a PMC

Throughout the paper we will use structural arguments about the *underlying graph* (or *topological structure*) of a Markov chain (Q, M, π) , which is defined as $(Q, \{(s, t) \mid M_{s,t} > 0\})$. For a parametric Markov chain (Q, X, M, π) we consider the *main structure* $(Q, \{(s, t) \mid \exists p. M[p]_{s,t} \neq 0\})$. That is, we only keep the entries of M that are not identically zero. We will show that the main structure matches the structure of $M[p]$ almost everywhere (that is, everywhere except possibly on a set with null measure). This means that w.l.o.g. we can assume that the given PMC has a constant topological structure. We begin by recalling a well-known fact which is immediate from the observation that a non-zero polynomial is non-zero almost everywhere (see, e.g., [9]).

► **Lemma 4.** *Any non-zero rational function $f \in \mathbb{Q}(X)$ is almost everywhere defined and non-zero.*

► **Lemma 5** (Constant topological structure). *Let $D \subseteq \mathbb{R}^X$ be the set of parameters defined as $D = \{p \mid p \text{ is admissible and } M[p] \text{ has the main structure}\}$. Then $\mathbb{R}^X \setminus D$ has null measure.*

Proof. Observe that

$$\mathbb{R}^X \setminus D = \bigcup_{s,t \in Q} \{p \mid M_{s,t} \text{ is not well-defined at } p\} \cup \bigcup_{s,t \in Q} \{p \mid \text{the structure of } M[p] \text{ differs from the main structure at } (s, t)\}$$

which is a finite union of sets of measure zero. Hence $\mathbb{R}^X \setminus D$ also has a null measure. ◀

Henceforth we define D to be the set described above.

We recall some basic structural notions about Markov chains. These descriptions also apply to the main structure of a parametric Markov chain. We say a collection of states $\mathcal{C} \subseteq Q$ is *strongly connected* if there is a path from any state to another in the restriction of the underlying graph of \mathcal{M} to \mathcal{C} . We only refer to (maximally) *strongly connected components* (*SCCs*), that is, SCCs for which there does not exist $s \in Q$ such that $\mathcal{C} \cup \{s\}$ is also strongly connected. A singleton state with no self loops and no other path to itself is considered its own SCC. Given an SCC \mathcal{C} , its *period* is the greatest common divisor of the lengths of cycles in \mathcal{C} . A SCC is called *aperiodic* if its period is 1, and otherwise it is called *periodic*. We say that a SCC \mathcal{C} is a *bottom SCC*, or *recurrent*, if for all $s \in \mathcal{C}$, $M_{s,s'} = 0$ for all $s' \in Q \setminus \mathcal{C}$. That is, no probability is lost from \mathcal{C} . If \mathcal{C} is not recurrent, it is called *transient*.

A Markov chain is *aperiodic* if all of its SCCs are aperiodic (and otherwise periodic) and *recurrent* if all its SCCs are recurrent. If the Markov chain consists of only one recurrent SCC then the Markov chain is said to be irreducible. As these properties are structural, depending only on the matrix M of $\mathcal{M} = (Q, X, M, \pi)$, we may say M is aperiodic or irreducible.

2.3 Semialgebraic targets

A set $T \subseteq \mathbb{R}^d$ is semialgebraic if it is a finite Boolean combination of sets specified by a polynomial inequality. That is, T can be obtained from sets of the form $\{x \in \mathbb{R}^d \mid f(x) \bowtie 0\}$ for some $\bowtie \in \{\geq, \leq, >, <, =\}$ using finitely many union and intersection operations. In fact, without loss of generality we can assume the sets to be of the form $\{x \in \mathbb{R}^d \mid f(x) \triangleright 0\}$ for $\triangleright \in \{\geq, >\}$. Written in disjunctive normal form, with \wedge corresponding to \cap and \vee corresponding to \cup , we can write T as $\bigcup_{i=1}^k \bigcap_{j=1}^{l_i} \{x \in \mathbb{R}^d \mid f_{ij}(x) \triangleright_{ij} 0\}$. Note that many restricted classes of target sets, such as singleton points and Boolean combinations of linear inequalities (e.g., polyhedra, halfspaces, and cones) are all examples of semialgebraic sets.

We will be considering the semialgebraic targets T_1, \dots, T_k within the universe of $U = \text{Dist}(Q) \subset \mathbb{R}^Q$, which will be endowed with the subspace topology with respect to the usual Euclidean topology on \mathbb{R}^Q . In this topology, a vector (i.e., a probability distribution) x is in the interior T° of a target T if and only if there exists $\epsilon > 0$ such that $B_\epsilon(x) \cap U \subseteq T$, where $B_\epsilon(x)$ is the ϵ -ball around x in \mathbb{R}^Q . We will be particularly interested in points on the boundary of T . The boundary of T , denoted ∂T , is the set of all limit points of T in U that are not in the interior of T . That is, $\partial T = \overline{T} \setminus T^\circ$, where \overline{T} is the closure of T in U .

We denote by $\text{vol}(D)$ the Lebesgue measure of a measurable set $D \subseteq \mathbb{R}^X$. Recalling that a vector v lies on the boundary of T if $v \in \partial T$, we say that a parametrised vector $(v[p])_{p \in D}$ is contained within the boundary of T if $v[p] \in \partial T$ for all $p \in D$. Given a parametrised vector, we will often be interested in the quantity $\text{vol}(\{p \in D \mid v[p] \in \partial T\})$.

2.4 Prefix-independent model checking

Let $\{T_1, \dots, T_k\}$ be a partition of the ambient space $\text{Dist}(Q)$ and $\Sigma = \{1, \dots, k\}$. Recall that properties over the predicates T_1, \dots, T_k are modelled by the subsets of Σ^ω . An ω -regular property P is *prefix-independent* if for every infinite word w and every finite word u acting as a prefix, $w \in P \iff uw \in P$. For such a property P it holds that for every $w, w' \in \Sigma^\omega$ that can be obtained from one another through finitely many insertions and deletions, $w \in P \iff w' \in P$.³

Given a property φ over Σ , we say a Markov chain \mathcal{M} satisfies φ , denoted $\mathcal{M} \models \varphi$, when the characteristic word of the Markov chain with respect to the targets T_1, \dots, T_k satisfies φ . In this paper we assume the property to be given as an ω -automaton (e.g., a non-deterministic Büchi automaton) over Σ . Then, one can check whether a given ultimately periodic word is accepted by such an automaton. This is done by checking non-emptiness on the automaton built by the product construction on the given automaton and an automaton for the ultimately periodic word. Properties given in other specification languages such as LTL or MSO can be handled by first creating an equivalent non-deterministic Büchi automaton, provided that the input property is prefix-independent.

2.5 Problems: synthesising parameters

First, we consider the set of parameters such that the sequence of distributions of the resulting Markov chain is ultimately trapped inside one of the target sets (“the positive set”). This is the parametric analogue of the well-known Ultimate Positivity Problem [22] (with the halfspace generalised to arbitrary semialgebraic set). Formally, we consider the following problem:

³ To see this, consider the common suffix v such that $w = uv$ and $w' = u'v$ and then observe that $w \in P \iff v \in P \iff u'v \in P$.

► **Problem 6** (Ultimate Positivity on PMCs). *Given a PMC $\mathcal{M} = (Q, X, M, \pi)$ and a semialgebraic set $T \subseteq \text{Dist}(Q)$, and letting $D \subseteq \mathbb{R}^X$ be the set of admissible parameter instantiations that give rise to the main structure, synthesise the set of feasible parameters $\{p \in D \mid \exists N \in \mathbb{N}. \forall n \geq N. \pi \cdot M[p]^n \in T\}$.*

Since the set of parameters could give rise to concrete instances which are hard, we do not synthesise the full set of feasible parameters exactly, but rather compute a semialgebraic subset that differs from the full set by a null set. In particular all of the parameter valuations in the set we compute give rise to an ultimately positive instance. If the computed semialgebraic set is non-empty, one can be sure that there does exist a parameter choice satisfying the property, and that such a parameter valuation can be computed. However, if the set is empty, then one cannot be sure that there does not exist a choice; but in this case one would know that even if there is such a parameter choice, there are “not too many choices”.

In Theorem 9 of Section 3 we compute this set for aperiodic recurrent finite-state parametric Markov chains, before generalising the result to periodic Markov chains in Theorem 13 in Section 4.

Being ultimately trapped inside a semialgebraic set is a prefix-independent property. Next, we generalise the problem to any prefix-independent property.

► **Problem 7** (Prefix-independent model checking on PMCs). *Given*

- a PMC $\mathcal{M} = (Q, X, M, \pi)$,
- semialgebraic sets T_1, \dots, T_k which form a partition of $\text{Dist}(Q)$, and
- a prefix-independent property φ over T_1, \dots, T_k ,

and letting $D \subseteq \mathbb{R}^X$ be the set of admissible parameter instantiations that give rise to the main structure, synthesise the set of the feasible parameters, i.e., those satisfying φ : $\{p \in D \mid \mathcal{M}[p] \models \varphi\}$.

In Theorem 15 of Section 5 we compute a semialgebraic subset of the feasible parameters differing up to a null set for the prefix-independent model checking problem.

3 Synthesising satisfying parameters for Ultimate Positivity in aperiodic and irreducible PMCs

Let $\mathcal{M} = (Q, X, M, \pi)$ be an aperiodic and irreducible Markov chain. It is well-known that any such Markov chain has a unique stationary distribution and that this distribution is also the unique limit distribution. In our case this means that for every choice of parameters $p \in D$ there is a unique probability distribution $\mu[p] \in \text{Dist}(Q)$ such that $\mu[p] \cdot M[p] = \mu[p]$. The fact that \mathcal{M} is irreducible implies that $\mu[p]$ will be strictly positive in each entry. The following lemma assures that this distribution is also a rational function in X and can be effectively computed.

► **Lemma 8.** *Given an aperiodic and irreducible PMC $\mathcal{M} = (Q, X, M, \pi)$, let $D \subseteq \mathbb{R}^X$ be the set of admissible parameters leading to the main structure of \mathcal{M} . There exists a unique parametric limit distribution $\mu \in \mathbb{Q}(X)^Q$ such that $\lim_{n \rightarrow \infty} \pi \cdot M[p]^n = \mu[p]$ for all $p \in D$. Furthermore, μ can be effectively computed.*

Proof. The stationary distribution $\mu : D \rightarrow \text{Dist}(Q)$ is the unique solution of the linear equation system $\mu[p] \cdot M[p] = \mu[p]$ in probability distributions. Hence μ can be computed by performing Gaussian elimination on the system $\mu[p] \cdot M[p] = \mu[p]$ followed by a normalisation step. This shows that every entry $\mu[p]_s$ of $\mu[p]$ is a rational function in p . ◀

We now establish the main theorem of our paper, showing how to compute a semialgebraic set of parameters which, up to a null set, equals the set of admissible parameters satisfying Ultimate Positivity. Our approach relies on the assumption that the volume of limit distributions lying on the boundary of a target T is null, that is, $\text{vol}(\{p \in D \mid \mu[p] \in \partial T\}) = 0$. We say that an instance of Problem 6 is *degenerate* if $\text{vol}(\{p \in D \mid \mu[p] \in \partial T\}) > 0$. If one considers only half-spaces as target sets, our requirement of non-degeneracy corresponds exactly to the third condition of [19, Theorem 1], which tackles the corresponding model-checking question for non-parametric Markov chains.

To see why such an assumption is strictly needed we show that, without this assumption, Problem 6 is as hard as ultimate positivity. That is, one would need to answer potentially intractable instances of the Ultimate Positivity Problem. Consider the following scenario. There is a single parameter p and all of its instantiations lead to the same non-parametric Markov chain \mathcal{M} . For any Markov chain \mathcal{M} , such a parametric Markov chain $\mathcal{M}[p]$ can easily be constructed. Recall that the Ultimate Positivity Problem for stochastic matrices asks whether there exists N such that for all $n \geq N$, it holds that $(\pi \cdot M^n)_s \geq 1/2$, for a given stochastic matrix M , an initial vector π and a state s [2]. Ultimate Positivity Problem for stochastic matrices can be easily expressed as an Ultimate Positivity Problem for PMCs (Problem 6). Then, the answer to the non-parametric Ultimate Positivity instance is yes if and only if the measure of parameters satisfying the formula is 1 (and in case the answer is no, the computed semialgebraic set will be empty, having measure zero). The decidability status of the Ultimate Positivity Problem is a major open question. However, it is solvable if the limit distribution of M in state s is not zero. Our non-degeneracy assumption essentially excludes the currently intractable cases of this problem.

► **Theorem 9.** *Consider a non-degenerate instance of Problem 6, in which the following are given:*

- *an aperiodic and irreducible PMC $\mathcal{M} = (Q, X, M, \pi)$, for which $D \subseteq \mathbb{R}^X$ is the semialgebraic set of admissible parameter values that give rise to the main structure,*
- *the parametric limit distribution $\mu \in \mathbb{Q}(X)^Q$ such that $\lim_{n \rightarrow \infty} \pi \cdot M[p]^n = \mu[p]$ for all $p \in D$, and*
- *a semialgebraic set $T = \bigcup_{i=1}^k \bigcap_{j=1}^{l_i} \{x \in \text{Dist}(Q) \mid f_{ij}(x) \triangleright_{ij} 0\}$, for which $\text{vol}(\{p \in D \mid \mu[p] \in \partial T\}) = 0$.*

Then a semialgebraic set D'_T , contained in $D_T = \{p \in D \mid \exists N \in \mathbb{N}. \forall n \geq N. \pi \cdot M[p]^n \in T\}$ but differing from D_T only by a null set, can be effectively computed.

Proof. Since for all $p \in D$, we have $\lim_{n \rightarrow \infty} \pi M[p]^n = \mu[p]$ then for all $p \in D$ such that $\mu[p] \in T^\circ$ it holds that there exists N such that for all $n \geq N$, $\pi M[p]^n \in T$. Clearly, if $p \in D$ is such that $\mu[p] \notin \bar{T}$, then the sequence of distributions of $\mathcal{M}[p]$ is eventually outside of T . It remains to consider the case where $\mu[p] \in \partial T$. Since by our assumption $\text{vol}(\{p \in D \mid \mu[p] \in \partial T\}) = 0$ it holds that D_T differs from $D'_T = \{p \in D \mid \mu[p] \in T^\circ\}$ by only a null set. Therefore it suffices to show how to compute a representation for D'_T .

The set D'_T is a semialgebraic set, for which an implicit representation in the first order theory of the reals can be found in polynomial time. To see this, observe that $D'_T = \{p \in \mathbb{R}^X \mid p \in D \wedge \exists y. y = \mu[p] \wedge y \in T^\circ\}$, with also D semialgebraic. The set T° is itself a semialgebraic set, which can easily be seen by specification in the theory of the reals as $\{x \in U \mid \exists \epsilon > 0. \forall z \in U, |z - x| \leq \epsilon \implies z \in T\}$ (recall from Section 2.3 that $U = \text{Dist}(Q)$ is the universe of probability distributions over Q). Finally, $z \in T$ can be expressed in the theory of the reals by asserting that $\bigvee_{i=1}^k \bigwedge_j^{l_i} f_{ij}(x) \triangleright_{ij} 0$ where $f_{i,j} \triangleright_{ij} 0$ are the polynomial inequalities defining T . This concludes the proof in case one is satisfied

with the set D'_T represented in the first order theory of the reals. In case an explicit form is required, quantifier elimination can be used to compute D'_T as boolean combination of polynomial inequalities, i.e., of the form $\{x \in \mathbb{R}^X \mid \bigwedge_i \bigvee_j g_{ij}(x) \triangleright_{ij} 0\}$ [23, Theorem 1.2]. ◀

► **Remark 10.** Since the Lebesgue measure is complete (i.e., every subset of a null set is measurable), it follows from the second part of Theorem 9 that the set D_T is Lebesgue measurable and hence Problem 6 is well-defined.

► **Remark 11.** Observe that it is decidable whether the instance is degenerate. This amounts to asking whether $\text{vol}(\{p \in D \mid \mu[p] \in \partial T\}) = 0$, which is the case if and only if the interior of the set is empty. The set $D_{\partial T} = \{p \in D \mid \mu[p] \in \partial T\}$ is semialgebraic, thus the interior $D_{\partial T}^\circ$ is also semialgebraic, for which one can test emptiness.

Secondly, we note that degenerate instances are somewhat unlikely. Recall the limit distribution is a rational function. Should the limit distribution coincide with the boundary of T for a positive volume of points then the function must essentially correspond with one of the polynomials defining a boundary of T . If the difference is zero with positive measure then by Lemma 4 the difference is the zero function, and we conclude that they must be the same function. This would seem to indicate that the target had been constructed adversarially with a priori knowledge of the limit distribution.

3.1 Complexity

Together Lemma 8 (which shows how to compute μ using Gaussian elimination) and Theorem 9 produce, up to a null set, the set of parameters of \mathcal{M} satisfying ultimate positivity for a target T in the case that \mathcal{M} is an irreducible and aperiodic PMC. We now consider the complexity of this reduction.

In general, the number of terms of a rational functions one gets from applying Gaussian elimination over the field of rational functions may become exponential. However, for a fixed number of parameters the parametrised stationary distribution μ (from Lemma 8) can be computed in polynomial time using *fraction-free* Gaussian elimination [6] (thus the representation of μ needs at most polynomial space).

It is then straightforward to see that the implicit representation, given as a sentence in the first order theory of the reals, can be found in polynomial time. We consider the complexity of computing the explicit representation in the following lemma and observe that this is polynomial time for fixed Markov chains \mathcal{M} .

► **Lemma 12.** *The explicit representation of D'_T can be found in time $p(x)^{O(|X||Q|^2)}$, where p is a polynomial in the size of the inputs $\mathcal{M} = (Q, X, M, \pi)$, μ , and T (represented by x).*

Proof. Consider an implicit description of a semialgebraic set given by a sentence in the theory of the reals of the form $\{y \in \mathbb{R}^\ell \mid Q_1 x_1 \in \mathbb{R}^{n_1} \dots Q_\omega x_\omega \in \mathbb{R}^{n_\omega} P(B_1(y, x), \dots, B_m(y, x))\}$, where $Q_i \neq Q_{i+1}$ are quantifiers in $\{\exists, \forall\}$, P is a Boolean formula in m variables and the B_i 's are polynomial inequalities of degree at most d in variables from x_1, \dots, x_ω and integer coefficients of bit-size at most L . Define $K_1 = \ell \prod_{k=1}^\omega n_k$ and $K_2 = \ell + \sum_{k=1}^\omega n_k$. By Theorem 1.2 of [23] the explicit description can be found from the implicit description using $L^{O(1)}(md)^{2^{O(\omega)}K_1}$ many arithmetic operations and $(md)^{K_2}$ evaluations of the Boolean formula P .

The formula described in Theorem 9 implicitly uses inequalities on rational functions, with rational coefficients. Observe that this can be converted to polynomial inequalities with integer coefficients, e.g., $f(x)/g(x) > 0 \iff g(x) \neq 0 \wedge f(x)g(x) > 0$. Then, rational coefficients can be removed by multiplying through by the lcm of denominators.

In the proof of Theorem 9, the implicit representation of D'_T is constructed in polynomial time by suitably describing the set in the first order theory of the reals, in time polynomial in the sizes of \mathcal{M} , μ and T . Let $q(x)$ be such a polynomial. We observe that the resulting representation has bounded quantifier alternation. In particular, composing the descriptions of D_T and T° into a single formula, the description has free parameters $p \in D$ (thus $\ell = |X|$) and quantification of the form $\exists y \in \text{Dist}(Q), \epsilon > 0, \forall z \in \text{Dist}(Q)$ followed by a Boolean combination of polynomial inequalities. Hence, there are two blocks of quantifiers ($\omega = 2$), of size $n_1 = |Q| + 1$ and $n_2 = |Q|$. The degree d of the polynomials and the number of such polynomial inequalities m are polynomial in the same parameters to describe T and μ . Let $u(x)$ be such a polynomial.

Since $|P|$ is at most $q(x)$, the Boolean formula can be evaluated in linear time, i.e., $q(x)$. Thus the conversion to explicit representation using the procedure of Renegar thus takes $O(u(x)^{O(|X||Q|^2)}q(x))$ many operations. But $O(u(x)^{O(|X||Q|^2)}q(x)) = O(p(x)^{O(|X||Q|^2)})$ for some larger polynomial $p(x)$ (assuming $|X| \neq 0$ and $|Q| \neq 0$), which concludes the proof. \blacktriangleleft

Recall that when $|X|$ is fixed then μ can be computed in time polynomial in \mathcal{M} , then the size of x is itself polynomial. Further, when the size of $|Q|$ is fixed, then the procedure is polynomial time in the size of \mathcal{M} and T and polynomial in T when \mathcal{M} is fixed (the parametric Markov chain may be considered fixed when the chain is given but the problem needs to be considered for several possible targets).

4 The limit distribution of periodic Markov chains with transient states

We have observed that we can compute the parametric stationary distribution for aperiodic and irreducible PMCs. Next, we show how to drop both of these restrictions by handling periodicity and transient states.

4.1 Managing periodicity

We observe that we can assume that the matrix is aperiodic for all parameters by considering subsequences. Recall that we can assume that the topological structure of $\mathcal{M} = (Q, X, M, \pi)$ is constant. When a Markov chain is periodic with period H we have that M^H is aperiodic. We consider H many parametric Markov chains $\mathcal{M}^{(h)} = (Q, X, M^H, \pi \cdot M[p]^h)$ for each $h \in \{0, \dots, H-1\}$, each leading to the parametric orbit $(\pi \cdot M[p]^h (M[p]^H)^n)_n$. Each Markov chain $\mathcal{M}^{(h)}$ has the same aperiodic update matrix $M[p]^H$ but a different starting point $\pi \cdot M[p]^h$. For reachability questions, we can simply analyse each subsequence independently, although we must suitably interleave the results if considering more general properties.

4.2 Managing transient states

Secondly, we consider transient states, that is, the states outside of a bottom strongly connected component. Let us assume $\mathcal{M} = (Q, X, M, \pi)$ is an aperiodic Markov chain. We know from the standard literature that the limit probability for any transient state is zero. However, we must decide how much of the total weight which started in a transient state ultimately reaches each of the bottom strongly connected components and weight the respective stationary distributions accordingly.

We are interested in the absorption probability of each bottom SCC. We consider a new (parametric) Markov chain (Q', X, N, π) where each BSCC \mathcal{C} is reduced to a single, aperiodic, absorbing state $q_{\mathcal{C}}$. Let $B = \{b_{\mathcal{C}} \mid \mathcal{C} \text{ is a bottom SCC}\}$, $F = \{q \in Q \mid q \text{ is transient}\}$ and

10:12 Parameter Synthesis for Parametric Probabilistic Dynamical Systems

$Q' = F \cup B$, the set of transient states and the new representative bottom states. Let $N \in \mathbb{Q}(X)^{Q' \times Q'}$, be defined such that $N_{q,q'} = M_{q,q'}$ if q, q' are in F , $N_{q,b_C} = \sum_{q' \in \mathcal{C}} M_{q,q'}$, $N_{b_C,b_C} = 1$ and $N_{b_C,q'} = 0$ if $q' \neq b_C$.

We compute absorbing probabilities $a \in \mathbb{Q}(X)^{Q' \times B}$, where a_{q,b_C} is the probability of reaching bottom state b_C starting in state q . Note that this is parametric in variables X . To compute a , we solve the linear equation system, where for each $b_C \in B$ we require that $a_{b_C,b_C} = 1$ (every bottom SCC is absorbing), $a_{q,b_C} = 0$ if q cannot reach b_C , and $a_{q,b_C} = \sum_{q' \in F} N_{q,q'} a_{q',b_C}$ if q can reach b_C .

We can also compute the limit distribution $\mu^{\mathcal{C}}$ for each bottom strongly connected component \mathcal{C} in isolation, this is the stationary distribution as computed in Lemma 8. We can then reweight these stationary distributions according to the probability which reaches each bottom SCC using the absorbing probabilities. For a bottom strongly connected component \mathcal{C} , the limit distribution of state $s \in \mathcal{C}$, is $\ell[p]_s = (\sum_{q \in Q'} \pi_q a[p]_{q,b_C} + \sum_{q \in \mathcal{C}} \pi_q) \mu^{\mathcal{C}}[p]_s$, when the initial distribution is π . For states not in any bottom strongly connected component, $s \in F$, we have $\ell[p]_s = 0$. Note that ℓ is also a rational function, since it is simply the product and sums of functions found by Gaussian elimination.

4.3 Managing periodic Markov chains with transient states

We now induce a limit distribution for each of the H aperiodic Markov chains $(\mathcal{M}^{(h)})_{h=0}^{H-1}$ found in Section 4.1. For each such chain, the matrix is $M[p]^H$, and we assume the stationary distributions $\mu^{\mathcal{C}}$ for each bottom SCC \mathcal{C} (this does not depend on h). However, we must consider the limit distribution for each of the H starting points we consider. That is the initial distribution is $\pi \cdot M[p]^h$ for each $h \in \{0, \dots, H-1\}$. Thus for each subsequence, distinguished by h , we can compute a unique limit distribution, where

$$\ell^{(h)}[p]_s = \left(\sum_{q \in Q'} (\pi \cdot M[p]^h)_q a[p]_{q,b_C} + \sum_{q \in \mathcal{C}} (\pi \cdot M[p]^h)_q \right) \mu^{\mathcal{C}}[p]_s \quad \text{for } s \in \mathcal{C}, \text{ and}$$

$$\ell^{(h)}[p]_s = 0 \quad \text{for } s \text{ transient,}$$

such that $\lim_{n \rightarrow \infty} (M[p]^H)^n (\pi \cdot M[p]^h) = \ell^{(h)}[p]$ for all $p \in D$.

4.4 Ultimate Positivity in the general case

Using the limit distribution established in this section, we now complete the proof of ultimate Positivity for periodic Markov chains with transient states.

► **Theorem 13.** *Consider a non-degenerate instance of Problem 6, in which the following are given:*

- $\mathcal{M} = (Q, X, M, \pi)$ is a PMC with period H , for which $D \subseteq \mathbb{R}^X$ is the semialgebraic set of admissible parameter values that give rise to the main structure,
- H parametric limit distributions $\ell^{(h)} \in \mathbb{Q}(X)$ for $h \in \{0, \dots, H-1\}$ such that $\lim_{n \rightarrow \infty} (M[p]^H)^n (\pi \cdot M[p]^h) = \ell^{(h)}[p]$ for all $p \in D$.
- a semialgebraic set $T = \bigcup_{i=1}^k \bigcap_{j=1}^{l_i} \{x \in \text{Dist}(Q) \mid f_{ij}(x) \triangleright_{ij} 0\}$, such that, for all limit distributions $(\ell^{(h)})_{h=0}^{H-1}$, we have $\text{vol}(\{p \in D \mid \ell^{(h)}[p] \in \delta T\}) = 0$.

Then a semialgebraic set D'_T , contained in $D_T = \{p \in D \mid \exists N \in \mathbb{N}. \forall n \geq N. \pi \cdot M[p]^n \in T\}$ but differing from D_T only by a null set, can be effectively computed.

Proof. Ultimate Positivity requires that $\exists N \in \mathbb{N} \forall n \geq N. \pi \cdot M[p]^n \in T$. Since we have split the orbit into H subsequences, we require that *all* of these subsequences eventually enter and remain inside T . That is there exists $N \in \mathbb{N}$ such that for all $h \in \{0, \dots, H\}$ and all $n \geq N$ we have $\pi \cdot M[p]^h (M[p]^H)^n \in T$. To compute a set D'_T as required, we use Theorem 9 on the limit distribution $\ell^{(h)}$. This gives us a set D'_h contained in $\{p \in D \mid \exists N \in \mathbb{N} \forall n \geq N. \pi \cdot M[p]^h (M[p]^H)^n \in T\}$, but of the same measure as D_h , for each h . Then, the set $D'_T = \bigcap_{h \in \{0, \dots, H\}} D'_h$ satisfies the requirements which concludes the proof. \blacktriangleleft

► **Remark 14.** In the case \mathcal{M} is aperiodic but not irreducible then the complexity result of Section 3.1 also applies. Note that in general the period H may be exponential, thus the periodic case requires consideration of exponentially many subsequences, for which the matrix of the system is M^H , which could be much larger than M .

5 Synthesising satisfying parameters for prefix-independent model checking

Finally we show how to compute a set with volume equivalent to the parameters which induce a Markov chain satisfying a prefix-independent property.

Consider semialgebraic targets T_1, \dots, T_k which partition $\text{Dist}(Q)$. That is $T_i \cap T_j = \emptyset$ for $i \neq j$ and $\text{Dist}(Q) = T_1 \cup \dots \cup T_k$.

We generalise the notion of *degenerate* instances to multiple targets. We say that an instance is *non-degenerate* if the the volume of any of the limit distributions lying on the boundary of any of the targets is zero. That is, $\text{vol}(\{p \in D \mid \ell^{(h)}[p] \in \partial T_i\}) = 0$ for each T_i and $\ell^{(h)}$. This allows us to be sure that every subsequence is eventually inside one of the targets, for all but a null-set of parameters.

► **Theorem 15.** *Consider a non-degenerate instance of Problem 7, in which the following are given:*

- $\mathcal{M} = (Q, X, M, \pi)$ is a PMC, with period H , for which $D \subseteq \mathbb{R}^X$ is the semialgebraic set of admissible parameter values that give rise to the main structure,
- H parametric limit distributions $\ell^{(h)} \in \mathbb{Q}(X)$ for $h \in \{0, \dots, H-1\}$ such that $\lim_{n \rightarrow \infty} (M[p]^H)^n (\pi \cdot M[p]^h) = \ell^{(h)}[p]$ for all $p \in D$,
- T_1, \dots, T_k are semialgebraic targets partitioning $\text{Dist}(Q)$ such that, for all limit distributions $(\ell^{(h)})_{h=0}^{H-1}$, and all targets T_i , we have $\text{vol}(\{p \in D \mid \ell^{(h)}[p] \in \partial T_i\}) = 0$, and
- φ is a prefix-independent ω -regular property over T_1, \dots, T_k .

Then, a semialgebraic set D'_φ , contained in $D_\varphi = \{p \in D \mid \mathcal{M}[p] \models \varphi\}$, but differing from D_φ only by a null set, can be effectively computed.

Proof. We know that any aperiodic Markov chain M will eventually converge to its limit distribution ℓ , that is, for any ϵ for sufficiently large n we have $|\pi M^n - \ell| < \epsilon$. So if the limit distribution is not on the boundary of a target, eventually the Markov chain stays inside the target or outside the target.

Hence, for all but a null set of $p \in D$ and each $h \in \{0, \dots, H-1\}$, we have that the orbit $\pi M[p]^h (M[p]^H)^n$ enters, and stays in, exactly one of T_1, \dots, T_k from some point on. Given p , we can determine this final target by checking in which set T_1, \dots, T_k the point $\ell^{(h)}[p]$ lies. Then, since $\pi M[p]^h (M[p]^H)^n$ is stationary from some point on, we have that every H th character of the characteristic word of $\mathcal{M}[p]$ w.r.t. T_1, \dots, T_k is fixed, and therefore the characteristic word is eventually periodic.

We consider each of the possible k^H periodic words describing the limit behaviour. That is, we consider w^ω for a word $w \in \{1, \dots, k\}^H$, i.e., w repeated infinitely many times. For all but a null set of parameters, the resulting characteristic word must have such a suffix. We can model check each such word, and decide if the word satisfies the property φ by asking whether w is accepted by the automaton representing φ .

We discard the parameter values leading to a periodic suffix which does not satisfy the specification φ . However, for each periodic word w that does satisfy φ , we compute $D'_w \subseteq D$ which, up to a null set, represents the parameters leading to this word. Fix $w \in \{1, \dots, k\}^H$. We compute, up to a null set, the set of parameters for which the periodic word of $\mathcal{M}[p]$ matches w at each position. Using Theorem 9 on limit distribution $\ell^{(h)}$ and target T_{w_h} compute the set $D'_{w,h} \subseteq D$, $D'_{w,h} = \{p \in D \mid \exists N \in \mathbb{N}. \forall n \geq N. \pi \cdot M[p]^h (M[p]^H)^n \in T_{w_h}\}$. To represent the whole word, we take intersection, that is let $D'_w = \bigcap_{h \in \{0, \dots, H-1\}} D'_{w,h}$.

Finally, we compute $D'_\varphi = \bigcup_{w \in \{1, \dots, k\}^H \text{ s.t. } w^\omega \models_\varphi} D'_w$, which is contained in D_φ (the set of parameters for which the PMC satisfies the property φ), and differs from D_φ by at most a null set. ◀

References

- 1 Manindra Agrawal, S. Akshay, Blaise Genest, and P. S. Thiagarajan. Approximate verification of the symbolic dynamics of markov chains. *J. ACM*, 62(1):2:1–2:34, 2015. doi:10.1145/2629417.
- 2 S. Akshay, Timos Antonopoulos, Joël Ouaknine, and James Worrell. Reachability problems for Markov chains. *Inf. Process. Lett.*, 115(2):155–158, 2015. doi:10.1016/j.ipl.2014.08.013.
- 3 Shaull Almagor, Toghrol Karimov, Edon Kelmendi, Joël Ouaknine, and James Worrell. Deciding ω -regular properties on linear recurrence sequences. *Proc. ACM Program. Lang.*, 5(POPL):1–24, 2021. doi:10.1145/3434329.
- 4 Tomáš Babiak, Mojmir Křetínský, Vojtěch Řehák, and Jan Strejček. LTL to Büchi Automata Translation: Fast and More Deterministic. In Cormac Flanagan and Barbara König, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, Lecture Notes in Computer Science, pages 95–109, Berlin, Heidelberg, 2012. Springer. doi:10.1007/978-3-642-28756-5_8.
- 5 Christel Baier, Florian Funke, Simon Jantsch, Toghrol Karimov, Engel Lefauchaux, Florian Luca, Joël Ouaknine, David Purser, Markus A. Whiteland, and James Worrell. The orbit problem for parametric linear dynamical systems. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021*, volume 203 of *LIPICs*, pages 28:1–28:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CONCUR.2021.28.
- 6 Christel Baier, Christian Hensel, Lisa Hutschenreiter, Sebastian Junges, Joost-Pieter Katoen, and Joachim Klein. Parametric Markov chains: PCTL complexity and fraction-free Gaussian elimination. *Information and Computation*, 272:104504, June 2020. doi:10.1016/j.ic.2019.104504.
- 7 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
- 8 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. Semi-algebraic sets. In *Algorithms in Real Algebraic Geometry*, pages 83–99. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. doi:10.1007/3-540-33099-2_4.
- 9 Richard Caron and Tim Traynor. The zero set of a polynomial. Technical report, WSMR report 05-02, 2005.
- 10 Rohit Chadha, Vijay Anand Korthikanti, Mahesh Viswanathan, Gul Agha, and YoungMin Kwon. Model checking mdps with a unique compact invariant set of distributions. In *Eighth International Conference on Quantitative Evaluation of Systems, QEST 2011, Aachen, Germany, 5-8 September, 2011*, pages 121–130. IEEE Computer Society, 2011. doi:10.1109/QEST.2011.22.
- 11 Ventsislav Chonev, Joël Ouaknine, and James Worrell. The polyhedron-hitting problem. In *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 940–956. SIAM, 2014.

- 12 Conrado Daws. Symbolic and Parametric Model Checking of Discrete-Time Markov Chains. In Zhiming Liu and Keijiro Araki, editors, *Theoretical Aspects of Computing – ICTAC 2004*, Lecture Notes in Computer Science, pages 280–294, Berlin, Heidelberg, 2005. Springer. doi:10.1007/978-3-540-31862-0_21.
- 13 Christian Dehnert, Sebastian Junges, Nils Jansen, Florian Corzilius, Matthias Volk, Harold Bruintjes, Joost-Pieter Katoen, and Erika Ábrahám. PROPhESY: A PRObabilistic ParamETER SYnthesis Tool. In Daniel Kroening and Corina S. Păsăreanu, editors, *Computer Aided Verification*, Lecture Notes in Computer Science, pages 214–231, Cham, 2015. Springer International Publishing. doi:10.1007/978-3-319-21690-4_13.
- 14 Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Probabilistic reachability for parametric Markov models. *International Journal on Software Tools for Technology Transfer*, 13(1):3–19, January 2011. doi:10.1007/s10009-010-0146-x.
- 15 Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, September 1994. doi:10.1007/BF01211866.
- 16 Ravindran Kannan and Richard J. Lipton. Polynomial-time algorithm for the orbit problem. *J. ACM*, 33(4):808–821, 1986. doi:10.1145/6490.6496.
- 17 Vijay Anand Korthikanti, Mahesh Viswanathan, Gul Agha, and YoungMin Kwon. Reasoning about MDPs as Transformers of Probability Distributions. In *2010 Seventh International Conference on the Quantitative Evaluation of Systems*, pages 199–208, September 2010. doi:10.1109/QEST.2010.35.
- 18 YoungMin Kwon and Gul Agha. Linear Inequality LTL (iLTL): A Model Checker for Discrete Time Markov Chains. In Jim Davies, Wolfram Schulte, and Mike Barnett, editors, *Formal Methods and Software Engineering*, Lecture Notes in Computer Science, pages 194–208, Berlin, Heidelberg, 2004. Springer. doi:10.1007/978-3-540-30482-1_21.
- 19 YoungMin Kwon and Gul Agha. Verifying the Evolution of Probability Distributions Governed by a DTMC. *IEEE Transactions on Software Engineering*, 37(1):126–141, January 2011. doi:10.1109/TSE.2010.80.
- 20 Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. Parametric probabilistic transition systems for system design and analysis. *Formal Aspects of Computing*, 19(1):93–109, March 2007. doi:10.1007/s00165-006-0015-2.
- 21 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.
- 22 Joël Ouaknine and James Worrell. Ultimate positivity is decidable for simple linear recurrence sequences. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming – 41st International Colloquium, ICALP 2014*, volume 8573 of *Lecture Notes in Computer Science*, pages 330–341. Springer, 2014. doi:10.1007/978-3-662-43951-7_28.
- 23 James Renegar. On the computational complexity and geometry of the first-order theory of the reals, part I: introduction. preliminaries. the geometry of semi-algebraic sets. the decision problem for the existential theory of the reals. *J. Symb. Comput.*, 13(3):255–300, 1992. doi:10.1016/S0747-7171(10)80003-3.

A Reachability properties may fail to have semialgebraic feasible sets

We have shown how to compute, for prefix-independent specifications, the feasible parameters up to a null set, whilst carefully circumventing hard instances. Example 1 demonstrates that our approach will not extend in general to properties that depend on the prefix of the characteristic word. However, the property considered ($B \cup L$) appears more complicated than a reachability property. In this section we give a parametric Markov chain for which the set of parameters satisfying a reachability property cannot be represented as a semialgebraic set, even up to a null set.

10:16 Parameter Synthesis for Parametric Probabilistic Dynamical Systems

Let \mathcal{M} be the 2-parameter Markov chain with states $(q_I, q_1, q_2, q_3, q_S)$ depicted in Figure 3a. For convenience we restrict the parameters to $D = \{(a, b) \mid a > 3, b > 2\}$. The initial distribution of \mathcal{M} is $(1, 0, 0, 0, 0)$, the limit distribution is $(0, 0, 0, 0, 1)$ and after $n > 0$ steps the probability of being in states q_1, q_2, q_3 is $\frac{1}{a} \frac{1}{2^{n-1}}, \frac{1}{b} \frac{1}{4^{n-1}}, \frac{1}{6} \frac{1}{4^{n-1}}$, respectively. Let $T = \{(u, x, y, z, w) \mid \frac{2}{3}z - \frac{2}{x} + \frac{6z}{y} < 0\}$ be the semialgebraic target⁴. Observe that the specification is non-degenerate. We will show that the set $D_T = \{(a, b) \in D \mid \exists n. \pi \cdot M^n[(a, b)] \in T\}$ is not semialgebraic, even up to a null set.

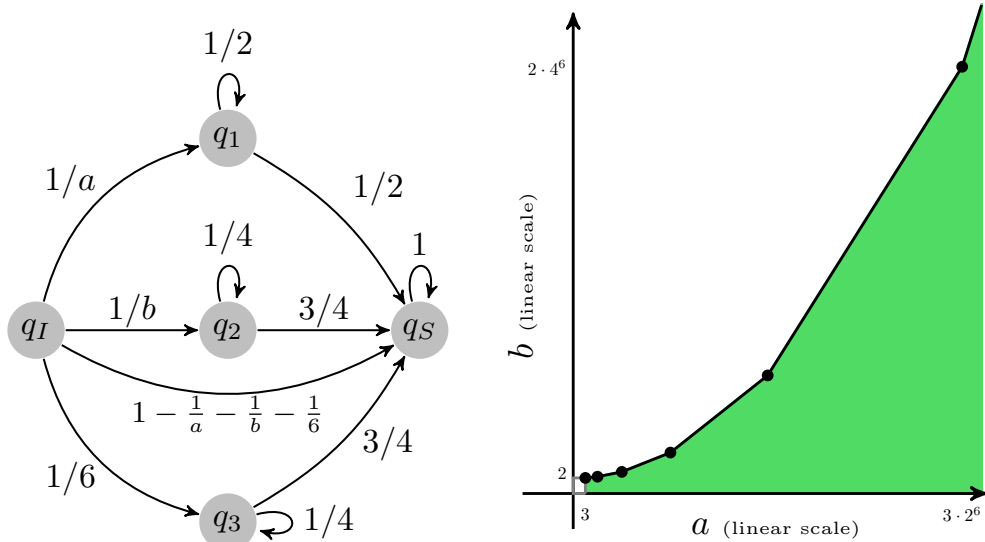
By definition of T it holds that $(a, b) \in D_T$ if and only if the linear recurrence sequence $u_n = 4^n - 2^n a + b$ is negative for some n . Hence

$$D_T = \bigcup_{n \in \mathbb{N}} \{(a, b) \mid 4^n - 2^n a + b < 0\} = \bigcup_{n \in \mathbb{N}} \{(a, b) \mid b < 2^n a - 4^n\}.$$

By analysing the family of inequalities above we can show that the set D_T is a polytope with infinite vertices $\{(3 \cdot 2^n, 2 \cdot 4^n) \mid n \in \mathbb{N}\}$, as depicted in Figure 3b. To prove the desired result, assume for contradiction that there exists semialgebraic $S \subseteq D_T$ such that the measure of $D_T \setminus S$ is null. As D_T is open, it follows that $D_T \subseteq \text{Interior}(\text{Closure}(S))$. On the other hand, inspecting the accumulation points of D_T yields the reverse containment, so that $D_T = \text{Interior}(\text{Closure}(S))$, whence D_T itself is semialgebraic. Finally, observe that we can write the set of vertices $V = \{(3 \cdot 2^n, 2 \cdot 4^n) \mid n \in \mathbb{N}\}$ as the set of all points on the boundary of D_T that cannot be expressed as a convex combination of two distinct points in $D \setminus D_T$. That is,

$$V = \{x \in \text{Closure}(D_T) \setminus D_T \mid \neg \exists y, z \in D \setminus D_T. y \neq z \wedge \exists \lambda \in (0, 1). x = \lambda y + (1 - \lambda)z\}.$$

This in turn makes V a semialgebraic set. However V is an infinite discrete set, and as such has infinitely many distinct connected components, contradicting a well-known property enjoyed by semialgebraic sets.



(a) Markov chain with parameters $p = (a, b)$.

(b) Parameters (green) for which $\pi \cdot M[p]^n$ hits T .



■ **Figure 3** A parametric Markov chain \mathcal{M} and the parameter set satisfying reachability in T .

⁴ One can write $T = \{(u, x, y, z, w) \mid \frac{2}{3}xy - 2yz + 6xz^2 < 0\}$ in order to make the inequality a polynomial.

Anytime Guarantees for Reachability in Uncountable Markov Decision Processes

Kush Grover  

Technische Universität München, Germany

Jan Křetínský  

Technische Universität München, Germany

Tobias Meggendorfer   

Institute of Science and Technology Austria, Wien, Austria

Maximilian Weininger  

Technische Universität München, Germany

Abstract

We consider the problem of approximating the reachability probabilities in Markov decision processes (MDP) with uncountable (continuous) state and action spaces. While there are algorithms that, for special classes of such MDP, provide a sequence of approximations converging to the true value in the limit, our aim is to obtain an algorithm with guarantees on the precision of the approximation.

As this problem is undecidable in general, assumptions on the MDP are necessary. Our main contribution is to identify sufficient assumptions that are as weak as possible, thus approaching the “boundary” of which systems can be correctly and reliably analyzed. To this end, we also argue why each of our assumptions is necessary for algorithms based on processing finitely many observations.

We present two solution variants. The first one provides converging lower bounds under weaker assumptions than typical ones from previous works concerned with guarantees. The second one then utilizes stronger assumptions to additionally provide converging upper bounds. Altogether, we obtain an *anytime* algorithm, i.e. yielding a sequence of approximants with known and iteratively improving precision, converging to the true value in the limit. Besides, due to the generality of our assumptions, our algorithms are very general templates, readily allowing for various heuristics from literature in contrast to, e.g., a specific discretization algorithm. Our theoretical contribution thus paves the way for future practical improvements without sacrificing correctness guarantees.

2012 ACM Subject Classification Mathematics of computing → Markov processes; Mathematics of computing → Continuous mathematics; Computing methodologies → Continuous models

Keywords and phrases Uncountable system, Markov decision process, discrete-time Markov control process, probabilistic verification, anytime guarantee

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.11

Related Version *Full Version:* <https://arxiv.org/abs/2008.04824> [17]

Funding *Kush Grover:* The author has been supported by the DFG research training group GRK 2428 *ConVeY*.

Maximilian Weininger: The author has been partially supported by DFG projects 383882557 *Statistical Unbounded Verification (SUV)* and 427755713 *Group-By Objectives in Probabilistic Verification (GOPro)*.

1 Introduction

The standard formalism for modelling systems with both non-deterministic and probabilistic behaviour are Markov decision processes (MDP) [43]. In the context of many applications such as cyber-physical systems, states and actions are used to model real-valued phenomena like position or throttle. Consequently, the state space and the action space may be uncountably



© Kush Grover, Jan Křetínský, Tobias Meggendorfer, and Maximilian Weininger; licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 11; pp. 11:1–11:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

infinite. For example, the intervals $[a, b] \times [c, d] \subseteq \mathbb{R}^2$ can model a safe area for a robot to move in or a set of available control inputs such as acceleration and steering angle. This gives rise to MDP with uncountable state- and action-spaces (sometimes called controlled discrete-time Markov process [51, 52] or discrete-time Markov control process [11, 28]), with applications ranging from modelling a Mars rover [10, 24], over water reservoir control [36] and warehouse storage management [38], to energy control [51], and many more [41].

Although systems modelled by MDP are often safety-critical, the analysis of uncountable systems is so complex that practical approaches for verification and controller synthesis are usually based on “best effort” learning techniques, for example *reinforcement learning*. While efficient in practice, these methods guarantee, even in the best case, convergence to the true result only in the limit, e.g. [40], or for increasingly precise discretization, e.g. [51, 32]. In line with the tradition of learning and to make the analysis more feasible, the typical objectives considered for MDP are either finite-horizon [37, 3] or discounted properties [18, 53, 25], together with restrictive assumptions. Note that when it comes to approximation, discounted properties effectively are finite-horizon. In contrast, ensuring safety of a reactive system or a certain probability to satisfy its mission goals requires an *unbounded* horizon and reduces to optimizing the reachability probabilities. Moreover, the safety-critical context requires *reliable* bounds on the probability, not an approximation with *unknown* precision.

In this paper, we provide the first provably correct anytime algorithm for (unbounded) reachability in uncountable MDP. As an *anytime* algorithm, it can at every step of the execution return correct lower and upper bounds on the true value. Moreover, these bounds gradually converge to the true value, allowing approximation up to an arbitrary precision. Since the problem is undecidable, the core of our contribution is identifying sufficient conditions on the uncountable MDP to allow for approximation.

Our *primary goal* is to provide *conditions as weak as possible*, thereby pushing towards the boundary of which systems can be analyzed provably correctly. To this end, we do not rely on any particular representation of the system. Nonetheless, for classical scenarios, and, in particular, for finite MDP, our conditions are mostly satisfied trivially.

Our *secondary goal* is to derive the respective algorithms as an *extension of value iteration* (VI) [29, 43], while *avoiding drawbacks of discretization*-based approaches. VI is a de facto standard method for numerical analysis of finite MDP, in particular with reachability objectives, regarded as practically efficient and allowing for heuristics avoiding the exploration of the complete state space, e.g. [9]. Interestingly, even for finite MDP, anytime VI algorithms with precision guarantees are quite recent [9, 19, 4, 44, 22]. Previous to that, the most used model checkers could return arbitrarily wrong results [19]. Providing VI with precision guarantees for general uncountable MDP is thus worthwhile on its own. Finally, while discretization is conceptually simple, we prefer to provide a solution that avoids the need to introduce arbitrary boundaries through gridding the whole state space and, moreover, instead utilizes information from one “cell” of the grid in other places, too.

To summarize, while algorithmic aspects form an important motivation, our primary contribution is theoretical: an explicit and complete set of generic assumptions allowing for guarantees, disregarding practical efficiency at this point. Consequently, while our approach lays foundations for further, more tailored approaches, it is not to be seen as a competitor to the existing practical, best-effort techniques, as these aim for a completely different goal.

Our Contribution. In this work, we provide the following:

Section 3: A set of assumptions that allow for computing converging *lower* bounds on the reachability probability in MDP with uncountable state and action spaces. We discuss in detail why they are weaker than usual, necessary, and applicable to typically considered systems. With these assumptions, we extend the standard (convergent but precision-ignorant) VI to this general setting.

Section 4: An additional set of assumptions that yield the first *anytime* algorithm, i.e. with provable bounds on the precision/error of the result, converging to 0. We combine the preceding algorithm with the technique of *bounded real-time dynamic programming (BRTDP)* [39] and provide also converging *upper* bounds on the reachability probability.

Section 5: A discussion of theoretical extensions and practical applications.

Related work. For detailed *theoretical* treatment of reachability and related problems on uncountable MDP, see e.g. [52, 11]. Reachability on uncountable MDP generalizes numerous problems known to be undecidable. For example, we can encode the halting problem of (probabilistic) Turing machines by encoding the tape content as real value. Similarly, almost-sure termination of probabilistic programs (undecidable [33]) is a special case of reachability on general uncountable MDP (see e.g. [16]). As precise reachability analysis is undecidable even for non-stochastic linear hybrid systems [26], many works turn their attention to more relaxed notions such as δ -reachability, e.g. [48], and/or employ many assumptions.

In order to obtain precision bounds, we assume that the *value* function, mapping states to their reachability probability, is *Lipschitz continuous* (and that we know the Lipschitz constant). This is slightly *weaker* than the classical approach of assuming Lipschitz continuity of the *transition* function (and knowledge of the constant), e.g. [2, 49]. In particular, these assumptions (i) imply our assumption (as we show in [17, App. B.2.1]) and (ii) are used even in the simpler settings of finite-horizon and discounted reward scenarios [5, 2, 49, 51] or even more restricted settings to obtain practical efficiency, e.g. [35]. In contrast to our approach, they are not anytime algorithms and require treatment of the whole state space.

To provide context, we outline how continuity is used (explicitly or implicitly) in related work and mention their respective results. Firstly, [25, 47] assume Lipschitz continuity, *but not* explicit knowledge of the constant. In essence, these approaches solve the problem by successively increasing internal parameters. The parameters then eventually cross a bound implied by the Lipschitz constant, yielding an “eventual correctness”. In particular, they provide “convergence in the limit” or “probably approximately correct” results, but no bounds on the error or the convergence rate; these would depend on knowledge of the constant.

Secondly, [18, 40, 2, 49, 51] (and our work) assume Lipschitz continuity *and* knowledge of the constant. Relying on the constant being provided externally, these works derive guarantees. Previously, the guarantees given are weaker than our convergent anytime bounds: Either convergence in the limit [40] or a bound on a discretization error, relativized to sub-optimal strategies [18] or bounded horizon [2, 49, 51].

Several of the above mentioned works employ *discretization* [18, 2, 49, 51]. This method is quite general, but obtaining any bounds on the error requires continuity assumptions [1]. Further, there are works that use other assumptions: [23, 24] use *reinforcement learning* methods to tackle reachability and more general problems, without any continuity assumption. However, they do not provide any guarantees. See [53] for a detailed exposition of similar approaches. Assuming an abstraction is given, *abstraction and bisimulation* approaches, e.g. [21, 20], provide guarantees, but only on the lower bounds. With significant assumptions on the system’s structure, *symbolic* approaches [37, 54, 45, 14] may even obtain exact solutions.

2 Preliminaries

In this section, we recall basics of probabilistic systems and set up the notation. As usual, \mathbb{N} and \mathbb{R} refer to the (positive) natural numbers and real numbers, respectively. For a set S , $\mathbb{1}_S$ denotes its *characteristic function*, i.e. $\mathbb{1}_S(x) = 1$ if $x \in S$ and 0 otherwise. We write S^* and S^ω to refer to the set of finite and infinite sequences comprising elements of S , respectively.

We assume familiarity with basic notions of measure theory, e.g. *measurable set* or *measurable function*, as well as probability theory, e.g. *probability spaces* and *measures* [8]. For a measure space X with sigma-algebra Σ_X , $\Pi(X)$ denotes the set of all probability measures on X . For a measure $\mu \in \Pi(X)$, we write $\mu(Y) = \int \mathbb{1}_Y d\mu$ to denote the mass of a measurable set $Y \in \Sigma_X$ (also called *event*). For two probability measures μ and ν , the *total variation distance* is defined as $\delta_{TV}(\mu, \nu) := 2 \cdot \sup_{Y \in \Sigma_X} |\mu(Y) - \nu(Y)|$. Some event happens *almost surely* (a.s.) w.r.t. some measure μ if it happens with probability 1. We write $\text{supp}(\mu)$ to denote the *support* of the probability measure μ .

► **Remark 1.** It is surprisingly difficult to give a well-defined notion of support for measures in general. Intuitively, $\text{supp}(\mu)$ describes the “smallest” set which μ assigns a value of 1. However, this is not well-defined for general measures. We discuss these issues and a proper definition in [17, App. E]. Throughout this work, similar subtle issues related to measure theory arise. For the sake of readability, these are mostly delegated to footnotes or the appendix of the full version [17], and readers may safely skip over these points.

We work with *Markov decision processes* (MDP) [43], a widely used model to capture both non-determinism and probability. We consider uncountable state and action spaces.

► **Definition 2.** A (continuous-space, discrete-time) Markov decision process (MDP) is a tuple $\mathcal{M} = (S, \text{Act}, \text{Av}, \Delta)$, where S is a compact set of states (with topology \mathcal{T}_S and Borel σ -algebra $\Sigma_S = \mathfrak{B}(\mathcal{T}_S)$), Act is a compact set of actions (with topology \mathcal{T}_{Act} and Borel σ -algebra $\Sigma_{\text{Act}} = \mathfrak{B}(\mathcal{T}_{\text{Act}})$), $\text{Av}: S \rightarrow \Sigma_{\text{Act}} \setminus \{\emptyset\}$ assigns to every state a non-empty, measurable, and compact set of available actions, and $\Delta: S \times \text{Act} \rightarrow \Pi(S)$ is a transition function that for each state s and (available) action $a \in \text{Av}(s)$ yields a probability measure over successor states (i.e. a Markov Kernel). An MDP is called *finite* if $|S| < \infty$ and $|\text{Act}| < \infty$.

See [43, Sec. 2.3] and [6, Chp. 9] for a more detailed discussion on the technical considerations arising from uncountable state and action spaces. Note that we assume the set of available actions to be non-empty. This means that the system can never get “stuck” in a degenerate state without successors. *Markov chains* are a special case of MDP where $|\text{Av}(s)| = 1$ for all $s \in S$, i.e. a completely probabilistic system without any non-determinism. Our presented methods thus are directly applicable to Markov chains as well.

Given a measure $\mu \in \Pi(X)$ and a measurable function $f: X \rightarrow \mathbb{R}$ mapping elements of a set X to real numbers, we write $\mu\langle f \rangle := \int f(x) d\mu(x)$ to denote the integral of f with respect to μ . For example, $\Delta(s, a)\langle f \rangle$ denotes the expected value $\mathbb{E}_{s' \sim \Delta(s, a)} f(s')$ of $f: S \rightarrow \mathbb{R}$ over the successors of s under action a . Moreover, abusing notation, for some set of state $S' \subseteq S$ and function $\text{Av}': S' \rightarrow \text{Act}$, we write $S' \times \text{Av}' = \{(s, a) \mid s \in S', a \in \text{Av}'(s)\}$ to denote the set of state-action pairs with states from S' under Av' .

An *infinite path* in an MDP is some infinite sequence $\rho = s_1 a_1 s_2 a_2 \dots \in (S \times \text{Av})^\omega$, such that for every $i \in \mathbb{N}$ we have $s_{i+1} \in \text{supp}(\Delta(s_i, a_i))$. A *finite path* (or *history*) $\varrho = s_1 a_1 s_2 a_2 \dots s_n \in (S \times \text{Av})^* \times S$ is a non-empty, finite prefix of an infinite path of length $|\varrho| = n$, ending in state s_n , denoted by $\text{last}(\varrho)$. We use $\rho(i)$ and $\varrho(i)$ to refer to the i -th state in an (in)finite path. We refer to the set of finite (infinite) paths of an MDP \mathcal{M} by $\text{FPaths}_{\mathcal{M}}$ ($\text{Paths}_{\mathcal{M}}$). Analogously, we write $\text{FPaths}_{\mathcal{M}, s}$ ($\text{Paths}_{\mathcal{M}, s}$) for all (in)finite paths starting in s .

In order to obtain a probability measure, we first need to eliminate the non-determinism. This is done by a so-called *strategy* (also called *policy*, *controller*, or *scheduler*). A strategy on an MDP $\mathcal{M} = (S, Act, Av, \Delta)$ is a function $\pi: \text{FPaths}_{\mathcal{M}} \rightarrow \Pi(Act)$, s.t. $\text{supp}(\pi(\rho)) \subseteq Av(\text{last}(\rho))$. The set of all strategies is denoted by $\Pi_{\mathcal{M}}$. Intuitively, a strategy is a “recipe” describing which step to take in the current state, given the evolution of the system so far.

Given an MDP \mathcal{M} , a strategy $\pi \in \Pi_{\mathcal{M}}$, and an initial state s_0 , we obtain a measure on the set of infinite paths $\text{Paths}_{\mathcal{M}}$, which we denote as $\text{Pr}_{\mathcal{M}, s_0}^{\pi}$. See [43, Sec. 2] for further details. Thus, given a measurable set $A \subseteq \text{Paths}_{\mathcal{M}}$, we can define its maximal probability starting from state s_0 under any strategy by $\text{Pr}_{\mathcal{M}, s_0}^{\text{sup}}[A] := \sup_{\pi \in \Pi_{\mathcal{M}}} \text{Pr}_{\mathcal{M}, s_0}^{\pi}[A]$. Depending on the structure of A it may be the case that no optimal strategy exists and we have to resort to the supremum instead of the maximum. This may already arise for finite MDP, see [12].

For an MDP $\mathcal{M} = (S, Act, Av, \Delta)$ and a set of *target states* $T \subseteq S$, (*unbounded*) *reachability* refers to the set $\diamond T = \{\rho \in \text{Paths}_{\mathcal{M}} \mid \exists i \in \mathbb{N}. \rho(i) \in T\}$, i.e. all paths which eventually reach T . The set $\diamond T$ is measurable if T is measurable [51, Sec. 3.1], [52, Sec. 2].

Now, it is straightforward to define the *maximal reachability problem* of a given set of states. Given an MDP \mathcal{M} , target set T , and state s_0 , we are interested in computing the maximal probability of eventually reaching T , starting in state s_0 . Formally, we want to compute the *value* of the state s_0 , defined as $\mathcal{V}(s_0) := \text{Pr}_{\mathcal{M}, s_0}^{\text{sup}}[\diamond T] = \sup_{\pi \in \Pi_{\mathcal{M}}} \text{Pr}_{\mathcal{M}, s_0}^{\pi}[\diamond T]$. This state value function satisfies a straightforward fixed point equation, namely

$$\mathcal{V}(s) = 1 \quad \text{if } s \in T \quad \mathcal{V}(s) = \sup_{a \in Av(s)} \Delta(s, a) \langle \mathcal{V} \rangle \quad \text{otherwise.} \quad (1)$$

Moreover, \mathcal{V} is the *smallest* fixed point of this equation [6, Prop. 9.8, 9.10], [52, Thm. 3]. In our approach, we also deal with values of state-action pairs $(s, a) \in S \times Av$, where $\mathcal{V}(s, a) := \Delta(s, a) \langle \mathcal{V} \rangle$. Intuitively, this represents the value achieved by choosing action a in state s and then moving optimally. Clearly, we have that $\mathcal{V}(s) = \sup_{a \in Av(s)} \mathcal{V}(s, a)$. See [15, Sec. 4] for a discussion of reachability on finite MDP and [52] for the general case.

In this work, we are interested in *approximate solutions* due to the following two reasons. Firstly, obtaining precise solutions for MDP is difficult already under strict assumptions and undecidable in our general setting.⁽¹⁾ We thus resort to approximation, allowing for much lighter assumptions. Secondly, by considering approximation we are able to apply many different optimization techniques, potentially leading to algorithms which are able to handle real-world systems, which are out of reach for precise algorithms even for finite MDP [9].

We are interested in two types of approximations. Firstly, we consider approximating the value function in the limit, without knowledge about how close we are to the true value. This is captured by a semi-decision procedure for queries of the form $\text{Pr}_{\mathcal{M}, s}^{\text{sup}}[\diamond T] > \xi$ for a threshold $\xi \in [0, 1]$. We call this problem **ApproxLower**. Secondly, we consider the variant where we are given a precision requirement $\varepsilon > 0$ and obtain ε -optimal values (l, u) , i.e. values with $\mathcal{V}(s_0) \in [l, u]$ and $0 \leq u - l < \varepsilon$. We refer to this variant as **ApproxBounds**.

3 Converging Lower Bounds

In this section, we present the first set of assumptions, enabling us to compute *converging lower bounds* on the true value, solving the **ApproxLower** problem. In Section 3.1, we discuss each assumption in detail and argue on an intuitive level why it is necessary by means of

⁽¹⁾For example, one can encode the tape of a Turing machine into the binary representation of a real number and reduce the halting problem to a reachability query.

counterexamples. With the assumptions in place, in Section 3.2 we then present our first algorithm, also introducing several ideas we employ again in the following section.

Our assumptions and algorithms are motivated by *value iteration* (VI) [29], which we briefly outline. In a nutshell, VI boils down to repeatedly applying an iteration operator to a value vector v_n . For example, the canonical value iteration for reachability on finite MDP starts with $v_0(s) = 1$ for all $s \in T$ and 0 otherwise and then iterates

$$v_{n+1}(s) = \max_{a \in Act(s)} \sum_{s' \in S} \Delta(s, a, s') \cdot v_n(s') \quad (2)$$

for all $s \notin T$. The vector v_n converges monotonically from below to the true value for all states. We mention two important points. Firstly, the iteration can be applied “asynchronously”. Instead of updating all states in every iteration, we can pick a single state and only update its value. The values v_n still converge to the correct value as long as all states are updated infinitely often. Secondly, instead of storing a value per state, we can store a value for each state-action pair and obtain the state value as the maximum of these values. Both points are a technical detail for finite MDP, however they play an essential role in our uncountable variant. See [17, App. A.1] for more details on VI for finite MDP.

In the uncountable variant of Equation (2), v is a function, $Act(s)$ is potentially uncountable, and the sum is replaced by integration. As in this setting the problem is undecidable, naturally we have to employ some assumptions. Our goal is to sufficiently imitate the essence of Equation (2), obtaining convergence without being overly restrictive. In particular, we want to (i) represent (an approximation of) v_n using finite memory, (ii) safely approximate the maximum and integration, and (iii) select appropriate points to update v_n .

3.1 Assumptions

Before discussing each assumption in detail, we first put them into context. As we argue in the following, most of our assumptions typically hold implicitly. Still, by stating even basic computability assumptions in a form as weak as possible, we avoid “hidden” assumptions, e.g. by assuming that the state space is a subset of \mathbb{R}^d . Two of our assumptions are more restrictive, namely **Assumption C: Value Lipschitz Continuity** (Section 3.1.3) and, introduced later, **Assumption D: Absorption** (Section 4.1.2). However, they are also often used in related works, as we detail in the respective sections. Moreover, in light of previous results, the necessity of restrictive assumptions is to be expected: Computing bounds is hard or even undecidable already for very restricted classes. Aside from the discussion in the introduction, we additionally mention two further cases. In the setting of probabilistic programs (which are a very special case of uncountable MDP), deciding almost sure termination for a fixed initial state (which is a severely restricted subclass of reachability on uncountable MDP without non-determinism) is an actively researched topic with recent advances, see e.g. [30, 31], and shown to be Π_2^0 -complete [33], i.e. highly undecidable. In [27] and the references therein, the authors present (un-)decidability results for hybrid automata, which are a special case of uncountable MDP *without any stochastic dynamics* (flow transitions can be modelled as actions indicating the delay). As such, it is to be expected that the general class of models we consider has to be pruned very strictly in order to hope for any decidability results.

► **Remark 3.** As already mentioned, we want to provide assumptions which are as general as possible. Importantly, we avoid (unnecessarily) assuming any particular representation of the system. Our motivation is to ultimately identify the boundary of what is necessary to derive guarantees. While our assumptions are motivated by VI and built around Equation (2), we

note that being able to represent the state values and evaluate (some aspect of) the transition dynamics intuitively are a necessity for *any* method dealing with such systems. We do not claim that our framework of assumptions is the only way to approach the problem, instead we provide arguments why it is a sensible way to do so.

3.1.1 A: Basic Assumptions (Asm. A1-A4)

We first present a set of basic computability assumptions (**A1-A4**). These are essential, since for uncountable systems even the simplest computations are intractable without any assumptions. More specifically, such systems cannot be given explicitly (due to their infinite size), but instead have to be described symbolically by, e.g., differential equations. Thus, we necessarily require some notion of computability and structural properties for each part of this symbolic description. And indeed, each assumption essentially corresponds to one part of the MDP description (**Metric Space** to $S \times Act$, **Maximum Approximation** to Av , **Transition Approximation** to Δ , and **Target Computability** to T). They are weak and hold on practically all commonly considered systems (see [17, App. B.1]). In particular, finite MDP and discrete components are trivially subsumed by considering the discrete metric.

A1: Metric Space S and Act are metric spaces with (computable) metrics d_S and d_{Act} , respectively, and d_\times is a compatible⁽²⁾ metric on the space of state-action pairs $S \times Av$,

A2: Maximum Approximation For each state s and computable Lipschitz $f : Av(s) \rightarrow [0, 1]$, the value $\max_{a \in Av(s)} f(a)$ can be under-approximated to arbitrary precision.

A3: Transition Approximation For each state-action pair (s, a) and Lipschitz $g : S \rightarrow [0, 1]$ which can be under-approximated to arbitrary precision, the successor expectation $\Delta(s, a)\langle g \rangle$ can be under-approximated to arbitrary precision.

A4: Target Computability The target set T is decidable, i.e. we are given a computable predicate which, given a state s , decides whether $s \in T$.

We denote the approximations for **A2** and **A3** by APPROX_{\leq} , i.e. given a pair (s, a) and functions f, g as in the assumptions, we write (abusing notation) $\text{APPROX}_{\leq}(\max_{a \in Av(s)} f(a), \varepsilon)$ and $\text{APPROX}_{\leq}(\Delta(s, a)\langle g \rangle, \varepsilon)$ for approximation of the respective values up to precision ε , i.e. $0 \leq \max_{a \in Av(s)} f(a) - \text{APPROX}_{\leq}(\max_{a \in Av(s)} f(a), \varepsilon) \leq \varepsilon$ and analogous for $\Delta(s, a)\langle g \rangle$. Note that **A2** and **A3** are satisfied if we can sample densely in $Av(s)$ and approximate $\Delta(s, a)$.

3.1.2 B: Sampling (Asm. B.VI)

As there are uncountably many states, we are unable to explicitly update all of them at once and instead update values asynchronously. Moreover, as there may also be uncountably many actions, we instead store and update the values of state-action pairs. Together, we need to pick state-action pairs to update. We delegate this choice to a selection mechanism `GETPAIR`, an oracle for state-action pairs. We allow for `GETPAIR` to be “stateful”, i.e. the sampled state-action pair may depend on previously returned pairs. This is required in, for example, round-robin or simulation-based approaches. We only require a basic notion of fairness in order to guarantee that we do not miss out on any information. Note the additional identifier **.VI** (*value iteration*) on the assumption name; later on, a similar, but weaker variant (**B.BRTDP**) is introduced.

⁽²⁾For two pairs (s, a) and (s, a') we have that $k \cdot d_{Act}(a, a') \leq d_\times((s, a), (s, a')) \leq K \cdot d_{Act}(a, a')$ for some constants $k, K \geq 0$, analogous for d_S , achieved by, e.g. $d_\times((s, a), (s', a')) := d_S(s, s') + d_{Act}(a, a')$.

B.VI: State-Action Sampling Let $S^\diamond = \{\text{last}(\varrho) \mid \varrho \in \text{FPaths}_{\mathcal{M},s}\}$ the set of all reachable states. Then, for any $\varepsilon > 0$, $s \in S^\diamond$, and $a \in \text{Av}(s)$ we have that GETPAIR eventually yields a pair (s', a') with $d_\times((s, a), (s', a')) < \varepsilon$ and $\delta_{TV}(\Delta(s, a), \Delta(s', a')) < \varepsilon$ a.s.⁽³⁾

Essentially, this means that GETPAIR provides a way to “exhaustively” generate all behaviours of the system up to a precision of ε . This fairness assumption is easily satisfied under usual conditions. For example, if $S \times \text{Av}$ is a bounded subset of \mathbb{R}^d , we can randomly sample points in that space or consider increasingly dense grids. Alternatively, if we can sample from the set of actions and from the distributions of Δ , GETPAIR can be implemented by sampling paths of random length, following random actions. Note that we can view the procedure as a “template”: Instead of requiring a concrete method to acquire pairs to update, we leave this open for generality; we discuss implications of this in Sections 5.1 and 5.3.

The requirement on total variation may seem unnecessary, especially given that we will also assume continuity. However, otherwise we could, for example, miss out on solitary actions which are the “witnesses” for a state’s value: suppose that $\text{Av}(s) = [0, 1]$ and $\Delta(s, 0)$ moves to the goal, while $\Delta(s, a)$ just loops back to s . Only selecting actions close to $a = 0$ w.r.t. the product metric is not sufficient to observe that we can move to the goal. Note that this would not be necessary if we assumed continuity of the transition function – selecting “nearby” actions then also yields “similar” behaviour.

3.1.3 C: Lipschitz Continuity

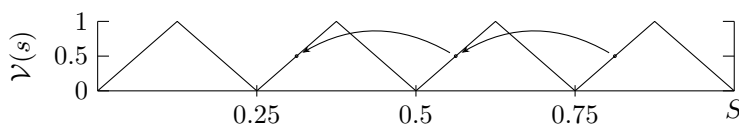
Finally, we present our already advertised continuity assumption. For simplicity, we give it in its strict form and discuss relaxations later in Section 5.2. Intuitively, Lipschitz continuity allows us to extrapolate the behaviour of the system from a single state to its surroundings.

C: Value Lipschitz Continuity The value functions $\mathcal{V}(s)$ and $\mathcal{V}(s, a)$ are Lipschitz continuous with *known* constants C_S and C_\times , i.e. for all $s, s' \in S$ and $a \in \text{Av}(s), a' \in \text{Av}(s')$ we have

$$|\mathcal{V}(s) - \mathcal{V}(s')| \leq C_S \cdot d_S(s, s') \quad |\mathcal{V}(s, a) - \mathcal{V}(s', a')| \leq C_\times \cdot d_\times((s, a), (s', a'))$$

This requirement may seem quite restrictive at first glance. Indeed, it is the only one in this section to not usually hold on “standard” systems. However, in order to obtain any kind of (provably correct) bounds, some notion of continuity is elementary, since otherwise we cannot safely extrapolate from finitely many observations to an uncountable set. The immediately arising questions are (i) why *Lipschitz* continuity is necessary compared to, e.g., regular or uniform continuity, and (ii) why *knowledge of the Lipschitz constant* is required. For the first point, note that we want to be able to extrapolate from values assigned to a single state to its immediate surroundings. While continuity means that the values in the surroundings do not “jump”, it does not give us any way of bounding the rate of change, and this rate may grow arbitrarily (for example, consider the continuous but not Lipschitz function $\sin(\frac{1}{x})$ for $x > 0$). So, also relating to the second point, without knowledge of the Lipschitz constant, regular continuity and Lipschitz continuity are (mostly) equivalent from a computational perspective: The function does not have discontinuities, but we cannot safely estimate the rate of change in general. To illustrate this point further, we give an intuitive example.

⁽³⁾Technically, it is sufficient to satisfy this property on any subset of S^\diamond which only differs from it up to measure 0. More precisely, we only require that this assumption holds for $S^\diamond = \text{supp}(\text{Pr}_{\mathcal{M},s}^{\text{sup}})$, i.e. the set of all reachable paths with non-zero measure. We omit this rather technical notion and the discussion it entails in order to avoid distracting from the central results of this work.



■ **Figure 1** The value function of Example 4, showing that knowledge of the constant is important.

► **Example 4.** We construct an MDP with a periodic, Lipschitz continuous value function, as illustrated in Figure 1 and formally defined below. Intuitively, for a given period width w (e.g. 0.25) and a periodic function f (e.g. a triangle function), a state s between 0 and w moves to a target or sink with probability $f(s)$. All larger states $s \geq w$ transition to $s - w$ with probability 1. The value function thus is periodic and Lipschitz continuous, see Figure 1 for a possible value function and [17, App. B.2.3] for a formal definition.

For a finite number of samples, we can choose f and w such that all samples achieve a value of 1. Nevertheless, we cannot conclude anything about states we have not sampled yet: *Without knowledge of the constant, we cannot extrapolate from samples.*

We note the underlying connection to the Nyquist-Shannon sampling theorem [46, Thm. 1]. Intuitively, the theorem states that, for a function that contains no frequencies higher than W , it is completely determined by giving its ordinates at a series of points spaced $0.5 \cdot W$ apart. If we know the Lipschitz constant, this gives us a way of bounding the “frequency” of the value function, and thus allows us to determine it by sampling a finite number of points. On the other hand, without the Lipschitz constant, we do not know the frequency and cannot judge whether we are “undersampling”.

Since we do not assume any particular representation of the transition system, we cannot derive such constants in general. Instead, these would need to be obtained by, e.g., domain knowledge, or tailored algorithms. As in previous approaches [18, 40, 2, 49, 51], we thus resort to assuming that we are given this constant, offloading this (highly non-trivial) step. Recall that Lipschitz continuity of the transition function implies Lipschitz continuity of the value function (see [17, App. B.2.1]), but can potentially be checked more easily.

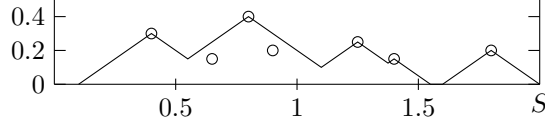
3.2 Assumptions Applied: Value Iteration Algorithm

Before we present our new algorithm, we explain how our assumptions allow us to lift VI to the uncountable domain. Contrary to the finite state setting, we are unable to store precise values for each state explicitly, since there are uncountably many states. Hence, the algorithm exploits the Lipschitz-continuity of the value function as follows. Assume that we know that the value of a state s is bounded from below by a value l , i.e. $\mathcal{V}(s) \geq l$. Then, by Lipschitz-continuity of \mathcal{V} , we know that the value of a state s' is bounded by $l - d_S(s, s') \cdot C_S$. More generally, if we are given a finite set of states Sampled with correct lower bounds $\widehat{\mathcal{L}}: \text{Sampled} \rightarrow [0, 1]$, we can safely extend these values to the whole state space by

$$\mathcal{L}(s) := \max_{s' \in \text{Sampled}} \left(\widehat{\mathcal{L}}(s') - C_S \cdot d_S(s, s') \right).$$

Since $\mathcal{V}(s) \geq \widehat{\mathcal{L}}(s)$ for all $s \in \text{Sampled}$, we have $\mathcal{V}(s) \geq \mathcal{L}(s)$ for all $s \in S$, i.e. $\mathcal{L}(\cdot)$ is a valid lower bound. We thus obtain a lower bound for all of the uncountably many states, described symbolically as a combination of finitely many samples. See Figure 2 for an illustration.

This is sufficient to deal with Markov chains, but for MDPs we additionally need to take care of the (potentially uncountably many) actions. Recall that value iteration updates state values with the maximum over available actions, $v_{n+1}(s) = \max_{a \in Av(s)} \Delta(s, a)(v_n)$.



■ **Figure 2** Example of the function extension on the set $[0, 2]$ with a Lipschitz constant of $C_S = 1$. Dots represent stored values in \widehat{L} , while the solid line represents the extrapolated function L . Note that it is possible to have $\widehat{L}(s) < L(s)$, as seen in the graph.

■ **Algorithm 1** The Value Iteration (VI) Algorithm for MDPs with general state- and action-spaces.

Input: `ApproxLower` query with threshold ξ , satisfying **A1–A4**, **B.VI** and **C**.

Output: `yes`, if $\mathcal{V}(s_0) > \xi$.

```

1: Sampled  $\leftarrow \emptyset, t \leftarrow 1$  ▷ Initialize
2: while APPROX≤(L(s0), PRECISION(t)) ≤ ξ do
3:    $(s, a) \leftarrow \text{GETPAIR}$  ▷ Sample state-action pair
4:   if  $s \in T$  then  $\widehat{L}(s, \cdot) \leftarrow 1$  ▷ Handle target states
5:   else  $\widehat{L}(s, a) \leftarrow \text{APPROX}_{\leq}(\Delta(s, a)\langle L \rangle, \text{PRECISION}(t))$  ▷ Update  $\widehat{L}$ 
6:   Sampled  $\leftarrow \text{Sampled} \cup \{(s, a)\}, t \leftarrow t + 1$ 
7: return yes
    
```

This is straightforward to compute when there are only finitely many actions, but in the uncountable case obtaining $L(s) = \sup_{a \in Av(s)} L(s, a)$ is much more involved. We apply the idea of Lipschitz continuity again, storing values for a set `Sampled` of state-action pairs instead of only states. We bound the value of every state-action pair by

$$L(s, a) := \max_{(s', a') \in \text{Sampled}} \left(\widehat{L}(s', a') - d_x((s, a), (s', a')) \cdot C_x \right) \quad (3)$$

Observe that $L(s, a)$ is computable and Lipschitz-continuous as well, so by **Maximum Approximation** we can approximate the bound of any state, i.e. $L(s) = \max_{a \in Av(s)} L(s, a)$, based on such a finite set of values assigned to state-action pairs. (Recall that $Av(s)$ is compact and $L(s, a)$ continuous, hence the maximum is attained.) Consequently, we can also under-approximate $\Delta(s, a)\langle L \rangle$ by **Transition Approximation**. To avoid clutter, we omit the following two special cases in the definition of $L(s, a)$: Firstly, if `Sampled` = \emptyset , we naturally set $L(s, a) = 0$. Secondly, if all pairs (s', a') are too far away for a sensible estimate, i.e. if Equation (3) was yielding $L(s, a) < 0$, we also set $L(s, a)$ to 0.

We present VI for MDPs with general state- and action-spaces in Algorithm 1. It depends on `PRECISION(t)`, a sequence of precisions converging to zero in the limit, e.g. `PRECISION(t) = $\frac{1}{t}$` . The algorithm executes the main loop until the current approximation of the lower bound of the initial state $L(s_0) = \max_{a \in Av(s_0)} L(s_0, a)$ exceeds the given threshold ξ . Inside the loop, the algorithm updates state-action pairs yielded by `GETPAIR`. For target states, the lower bound is set to 1. Otherwise, we set the bound of the selected pair to an approximation of the expected value of L under the corresponding transition. Here is the crucial difference to VI in the finite setting: Instead of using Equation (2), we have to use Equation (3) and `APPROX≤`, the approximations that exist by assumption, see Section 3.1.1. Since `PRECISION(t)` converges to zero, the approximations eventually get arbitrarily fine. The procedure `PRECISION(t)` may be adapted heuristically in order to speed up computation. For example, it may be beneficial to only approximate up to 0.01 precision at first to quickly get a rough overview. We show that Algorithm 1 is correct, i.e. the stored values (i) are lower bounds and (ii) converge to the true values in [17, App. E.1]. Here, we only provide a sketch, illustrating the main steps.

► **Theorem 5.** *Algorithm 1 is correct under Assumptions A1–A4, B.VI, and C, i.e. it outputs yes iff $\mathcal{V}(s) > \xi$.*

Proof sketch. First, we show that $L_t(s) \leq L_{t+1}(s) \leq \mathcal{V}(s)$ by simple induction on the step. Initially, we have $L_1(s) = 0$, obviously satisfying the condition. The updates in Lines 4 and 5 both keep correctness, i.e. $L_{t+1}(s) \leq \mathcal{V}(s)$, proving the claim.

Since L_t is monotone as argued above, its limit for $t \rightarrow \infty$ is well defined, denoted by L_∞ . By **State-Action Sampling**, the set of accumulation points of s_t contains all reachable states S^\diamond . We then prove that L_∞ satisfies the fixed point equation Equation (1). For this, we use the second part of the assumption on GETPAIR, namely that for every $(s, a) \in S^\diamond \times Av$ we get a converging subsequence (s_{t_k}, a_{t_k}) where additionally $\Delta(s_{t_k}, a_{t_k})$ converges to $\Delta(s, a)$ in total variation. Intuitively, since infinitely many updates occur infinitely close to (s, a) , its limit lower bound $L_\infty(s, a)$ agrees with the limit of the updates values $\lim_{k \rightarrow \infty} \Delta(s_{t_k}, a_{t_k}) \langle L_{t_k} \rangle$. Since L_∞ satisfies the fixed point equation and is less or equal to the value function \mathcal{V} , we get the result, since \mathcal{V} is the smallest fixed point. ◀

4 Converging Upper Bounds

In this section, we present the second set of assumptions, allowing us to additionally compute converging *upper* bounds. With both lower and upper bounds, we can quantify the progress of the algorithm and, in particular, terminate the computation once the bounds are sufficiently close. Therefore, instead of only providing a semi-decision procedure for reachability, this algorithm is able to determine the maximal reachability probability up to a given precision. Thus, we obtain the first algorithm able to handle such general systems with guarantees on its result. We again present our assumptions together with a discussion of their necessity (Section 4.1), and then introduce the subsequent algorithm and prove its correctness (Section 4.2). As expected, obtaining this additional information also requires additional assumptions. On the other hand, quite surprisingly, we can use the additional information of upper bounds to actually *speed up* the computation, as discussed in Section 5.3.

As before, our approach is inspired by algorithms for finite MDP, in this case by *Bounded Real-Time Dynamic Programming* (BRTDP) [39, 9]. BRTDP uses the same update equations as VI, but iterates both lower and upper bounds. A major contribution of [9] was to solve the long standing open problem of how to deal with *end components*. These parts of the state space prevent convergence of the upper bounds by introducing additional fixpoints of Equation (1). We direct the interested reader to [17, App. A.2] for further details on BRTDP and insights on the issue of end components. In the uncountable setting, these issues arise as well alongside several other, related problems, which we discuss in Section 4.1.2.

4.1 Assumptions

The basic assumptions A1–A4 as well as **Lipschitz continuity** (Assumption C) remain unchanged. For **Maximum Approximation** (A2) and **Transition Approximation** (A3), we additionally require that we are able to *over*-approximate the respective results. The respective assumptions are denoted by A5 and A6, respectively, and both over-approximations by APPROX_≥. Further, we only require a weakened variant of **State-Action Sampling**, now called Assumption **B.BRTDP** instead of Assumption **B.VI**. Finally, there is the new Assumption **D** called **Absorption**, addressing the aforementioned issue of end components.

4.1.1 B: Weaker Sampling (Asm. B.BRTDP)

We again assume a GETPAIR oracle, but, perhaps surprisingly, with *weaker* assumptions. Instead of requiring it to return “all” actions, we only require it to yield “optimal” actions, respective to a given state-action value function. We first introduce some notation. Intuitively, we want GETPAIR to yield actions which are optimal with respect to the *upper bounds* computed by the algorithm. However, these upper bounds potentially change after each update. Thus, assume that $f_n: S \times Av \rightarrow [0, 1]$ is an arbitrary sequence of computable, Lipschitz continuous, (point-wise) monotone decreasing functions, assigning a value to each state-action pair, and set $\mathcal{F} = (f_1, f_2, \dots)$. For each state $s \in S$, set

$$Av_{\mathcal{F}}(s) := \{a \in Av(s) \mid \forall \varepsilon > 0. \forall N \in \mathbb{N}. \exists n > N. \max_{a' \in Av(s)} f_n(s, a') - f_n(s, a) < \varepsilon\},$$

i.e. actions that infinitely often achieve values arbitrarily close to the optimum of f_n . Let $S_{\mathcal{F}}^{\diamond} = \{last(\varrho) \mid \varrho \in \text{FPaths}_{\mathcal{M}, s_0} \cap (S \times Av_{\mathcal{F}})^* \times S\}$ be the set of all states reachable using these optimal actions.⁽⁴⁾ Essentially, we require that GETPAIR samples densely in $S_{\mathcal{F}}^{\diamond} \times Av_{\mathcal{F}}$.

B.BRTDP: State-Action Sampling For any $\varepsilon > 0$, \mathcal{F} as above, $s \in S_{\mathcal{F}}^{\diamond}$, and $a \in Av_{\mathcal{F}}(s)$ we have that GETPAIR a.s. eventually yields a pair (s', a') with $d_{\times}((s, a), (s', a')) < \varepsilon$ and $\delta_{TV}(\Delta(s, a), \Delta(s', a')) < \varepsilon$.

While this new variant may seem much more involved, it is *weaker* than its previous variant, since $Av_{\mathcal{F}}(s) \subseteq Av(s)$ for each $s \in S$ and thus also $S_{\mathcal{F}}^{\diamond} \subseteq S^{\diamond}$. As such, it also allows for more practical optimizations, which we briefly discuss in Section 5.3.

4.1.2 D: Absorption

We present our most specific assumption. While it is *not needed* for correctness, we require it for convergence of the upper bounds to the value and thus for termination of the algorithm.

D: Absorption There exists a *known and decidable* set R (called *sink*) such that $\mathcal{V}(s) = 0$ for all $s \in R$. Moreover, for any $s \in S$ and strategy π we have $\Pr_{\mathcal{M}, s}^{\pi}[\diamond(T \cup R)] = 1$.

Intuitively, the assumption requires that for all strategies, the system will eventually reach a target or a goal state; in other words: It is not possible to avoid both target and sink infinitely long. Variants of this assumption are used in numerous settings: On MDP, it is similar to the contraction assumption, e.g. [6, Chp. 4]; in stochastic game theory (a two-player extension of MDP) it is called *stopping*, e.g. [13]; and, using terms from the theory of the stochastic shortest path problem, we require all strategies to be *proper*, see e.g. [7].

This assumption already is important in the finite setting: There, **Absorption** is equivalent to the absence of *end components*, which introduce multiple solutions of Equation (1). Then, a VI algorithm computing upper bounds can be “stuck” at a greater fixpoint than the value and thus does not converge [9, 19]. *Any* procedure using value iteration thus either needs to exclude such cases or detect and treat them. Aside from end components, which are the only issue in the finite setting, uncountable systems may feature other complex behaviour, such as Zeno-like approaching the target closer and closer without reaching it.

Unfortunately, even just detecting these problems already is difficult. For the mentioned, restricted setting of probabilistic programs, almost sure termination is Π_0^2 -complete [33]. Yet, universal termination with goal set $T \cup R$ is exactly what we require for **Absorption**. So, already on a restricted setting (together with a *given* guess for R), we cannot decide whether the assumption holds, let alone treat the underlying problems. Thus, we decide to exclude this issue and delegate treatment to specialized approaches.

⁽⁴⁾As in Section 3, we simplify the definition of $S_{\mathcal{F}}^{\diamond}$ slightly in order to avoid technical details.

■ **Algorithm 2** The BRTDP algorithm for MDPs with general state- and action-spaces.

Input: ApproxBounds query with precision ε , satisfying **A1–A6**, **B.BRTDP**, **C** and **D**.

Output: ε -optimal values (l, u) .

```

1: Sampled  $\leftarrow \emptyset, t \leftarrow 1$  ▷ Initialize
2: while  $\text{APPROX}_{\geq}(U(s_0), \text{PRECISION}(t)) - \text{APPROX}_{\leq}(L(s_0), \text{PRECISION}(t)) \geq \varepsilon$  do
3:    $s, a \leftarrow \text{GETPAIR}$  ▷ Sample stat-action pair
4:   if  $s \in T$  then  $\hat{L}(s, \cdot) \leftarrow 1$  ▷ Handle special cases
5:   else if  $s \in R$  then  $\hat{U}(s, \cdot) \leftarrow 0$ 
6:   else ▷ Update upper and lower bounds
7:      $\hat{U}(s, a) \leftarrow \text{APPROX}_{\geq}(\Delta(s, a)\langle U \rangle, \text{PRECISION}(t))$ 
8:      $\hat{L}(s, a) \leftarrow \text{APPROX}_{\leq}(\Delta(s, a)\langle L \rangle, \text{PRECISION}(t))$ 
9:   Sampled  $\leftarrow \text{Sampled} \cup \{(s, a)\}, t \leftarrow t + 1$ 
10: return  $(L(s_0), U(s_0))$ 

```

In summary, while this assumption is indeed restrictive, it is the key point that allows us to obtain convergent upper bounds and thus an anytime algorithm. As argued above, an assumption of this kind seems to be *necessary* to obtain such an algorithm in this generality.

► **Remark 6.** These problems do not occur when considering *finite horizon* or *discounted* properties, which are frequently used in practice. For details on treating finite horizon objectives, see [17, App. C.1]. Discounted reachability with a factor of $\gamma < 1$ is equivalent to normal reachability where at each step the system moves into a sink state with probability $(1 - \gamma)$. **Absorption** is trivially satisfied and our methods are directly applicable.

4.2 Assumptions Applied: The Convergent Anytime Algorithm

With our assumptions in place, we are ready to present our adaptation of BRTDP to the uncountable setting. Compared to VI, we now also store upper bounds, again using Lipschitz-continuity to extrapolate the stored values. In particular, together with the definitions of Equation (3) we additionally set

$$U(s, a) = \min_{(s', a') \in \text{Sampled}} \left(\hat{U}(s', a') + d_{\times}((s, a), (s', a')) \cdot C_{\times} \right).$$

We also set $U(s, a) = 1$ if either $\text{Sampled} = \emptyset$ or the above equation would yield $U(s, a) > 1$.

We present BRTDP in Algorithm 2. It is structurally similar to BRTDP in the finite setting (see [17, App. A.2]). The major difference is given by the storage tables \hat{U} and \hat{L} used to compute the current bounds U and L , again exploiting Lipschitz continuity. As before, the central idea is to repeatedly update state-action pairs given GETPAIR. If GETPAIR yields a state of the terminal sets T and R , we update the stored values directly. Otherwise, we back-propagate the value of the selected pair by computing the expected value under this transition. Moreover, we again require that $\text{PRECISION}(t)$ converges to zero. Note that the algorithm can easily be supplied with a-priori knowledge by initializing the upper and lower bounds to non-trivial values. Moreover, in contrast to VI, this algorithm is an *anytime* algorithm, i.e. it can at any time provide an approximate solution together with its precision.

Despite the algorithm being structurally similar to the finite variant of [9], the proof of correctness unsurprisingly is more intricate due to the uncountable sets. We again provide both a simplified proof sketch here and the full technical proof in [17, App. E.2].

► **Theorem 7.** *Algorithm 2 is correct under Assumptions A1–A6, B.BRTDP, C and D, and terminates with probability 1.*

Proof sketch. We again obtain monotonicity of the bounds, i.e. $L_t(s, a) \leq L_{t+1}(s, a) \leq V(s, a) \leq U_{t+1}(s, a) \leq U_t(s, a)$ by induction on t , using completely analogous arguments.

By monotonicity, we also obtain well defined limits U_∞ and L_∞ . Further, we define the difference function $\text{Diff}_t(s, a) = U_t(s, a) - L_t(s, a)$ together with its state based counterpart $\text{Diff}_t(s)$ and its limit $\text{Diff}_\infty(s)$. We show that $\text{Diff}_\infty(s_0) = 0$, proving convergence. To this end, similar to the previous proof, we prove that Diff_∞ satisfies a fixed point equation on S_+^\diamond (see **B.BRTDP**), namely $\text{Diff}_\infty(s) = \Delta(s, a(s)) \langle \text{Diff}_\infty \rangle$ where $a(s)$ is a specially chosen “optimal” action for each state satisfying $\text{Diff}_\infty(s, a(s)) = \text{Diff}_\infty(s)$. Now, set $\text{Diff}_* = \max_{s \in S_+^\diamond} \text{Diff}_\infty(s)$ the maximal difference on S_+^\diamond and let S_*^\diamond be the set of witnesses obtaining Diff_* . Then, $\Delta(s, a(s), S_*^\diamond) = 1$: If a part of the transition’s probability mass would move to a region with smaller difference, an appropriate update of a pair close to $(s, a(s))$ would reduce its difference. Hence, the set of states S_*^\diamond is a “stable” subset of the system when following the actions $a(s)$. By **Absorption**, we eventually have to reach either the target T or the sink R starting from any state in S_*^\diamond . Since $\text{Diff}_\infty(s) = 0$ for all (sampled) states in $T \cup R$ and Diff_∞ satisfies the fixed point equation, we get that $\text{Diff}_\infty(s) = 0$ for all states S_*^\diamond and consequently $\text{Diff}_\infty(s_0) = 0$. ◀

5 Discussion

5.1 Relation to Algorithms for Finite Systems and Discretization

Our algorithm directly generalizes the classical value iteration as well as BRTDP for finite MDP by an appropriate choice of GETPAIR. In value iteration, it proceeds in round-robin fashion, enumerating all state-action pairs. Note that the algorithm immediately uses the results of previous updates, corresponding to the *Gauß-Seidel variant* of VI; to exactly obtain synchronous value iteration, we would have to slightly modify the structure for saving the values. In BRTDP, GETPAIR simulates paths through the MDP and we update only those states encountered during the simulation.

Approaches based on discretization through, e.g., grids with increasing precision, essentially reduce the uncountable state space to a finite one. This is also encompassed by GETPAIR, e.g. by selecting the grid points in round robin or randomized fashion. However, our algorithm has the following key advantages when compared to classical discretization. Firstly, it avoids the need to grid the whole state space (typically into cells of regular sizes). Secondly, in discretization, updating the value of one cell does not directly affect the value in other cells; in contrast in our algorithm, knowledge about a state fluently propagates to other areas (by using Equation (3)) without being hindered by (arbitrarily chosen) cell boundaries.

5.2 Extensions

We outline possible extensions and augmentations of our approach to showcase its versatility.

Discontinuities. Our Lipschitz assumption **C** actually is slightly stronger than required. We first give an example of a system exhibiting discontinuities and then describe how our approach can be modified to deal with it. More details are in [17, App. C.2].

► **Example 8.** Consider a robot navigating a terrain with cliffs, where falling down a cliff immediately makes it impossible to reach the target. There, states which are barely on the edge may still reach the goal with significant probability, while a small step to the side results in falling down the cliff and zero probability of reaching the goal.

To solve this example, one could model the cliff as a steep but continuous slope, which would make our approach still possible. Unfortunately, this might not be very practical, since the Lipschitz constant then is quite large.

However, if we know of discontinuities, e.g. the location of cliffs in the terrain the robot navigates, both our algorithms can be extended as follows: Instead of requiring \mathcal{V} to be continuous on the whole domain, we may assume that we are given a (finite, decidable) partitioning of the state set S into several sets S_i . We allow the value function to be discontinuous along the boundaries of S_i (the cliffs), as long as it remains Lipschitz-continuous inside each S_i . We only need to slightly modify the assumption on GETPAIR by requiring that for any state-action pair (s, a) with $s \in S_i$ we eventually get a nearby, similarly behaving state-action pair (s', a') of the same region, i.e. $s' \in S_i$. While computing the bounds of a particular state-action pair, e.g. $U(s, a)$, we first determine which partition S_i the state s belongs to and then only consider the stored values of states inside the region S_i .

Linear Temporal Logic. In [9], the authors extend BRTDP to LTL queries [42]. Several difficulties arise in the uncountable setting. For example, in order to prove *liveness* conditions, we need to solve the *repeated reachability* problem, i.e. whether a particular set of states is reached infinitely often. This is difficult even for restricted classes of uncountable systems, and impossible in the general case. In particular, [9] relies on analysing end components, which we already identified as an unresolved problem. We provide further insight in [17, App. C.3]. Nevertheless, there is a straightforward extension of our approach to the subclass of *reach-avoid* problems [50] (or *constrained reachability* [52]), see [17, App. C.4].

5.3 Implementation and Heuristics

For completeness, we implemented a prototype of our BRTDP algorithm to demonstrate its effectiveness. See [17, App. D] for details and an evaluation on both a one- and two-dimensional navigation model. Our implementation is barely optimized, with no delegation to high-performance libraries. Yet, these non-trivial models are solved in reasonable time. However, since we aim for assumptions that are as general as possible, one cannot expect our generic approach perform on par with highly optimized tools. Our prototype serves as a proof-of-concept and does not aim to be competitive with specialized approaches. We highlight again that the goal of our paper is *not* to be practically efficient in a particular, restricted setting, but rather to provide general assumptions and theoretical algorithms applicable to all kinds of uncountable systems.

Aside from several possible optimizations concerning the concrete implementation, we suggest two more general directions for heuristics:

Adaptive Lipschitz constants. As an example, suppose that a robot is navigating mostly flat land close to its home, but more hilly terrain further away. The flat land has a smaller Lipschitz constant than the hilly terrain, and thus here we can infer tighter bounds. More generally, given a partitioning of the state space and local Lipschitz constants for every subset, we use this local knowledge when computing \hat{L} and \hat{U} instead of using the global Lipschitz constant, which is the maximum of all local ones. See [17, App. C.2] for details.

GetPair-heuristics. In Section 3.1.2, we mentioned two simple implementations of GETPAIR. Firstly, we can discretize both state and action space, yielding each state-action pair in the discretization for a finite number of iterations, choosing a finer discretization constant, and repeating the process until convergence. Assuming that we can sample all state-action pairs in the discretization, this method eventually samples arbitrarily close to *any* state-action pair in $S \times Av$ and thus trivially satisfies the sampling assumption. This intuitively corresponds to executing interval iteration [19] on the (increasingly refined) discretized systems. Note that this approach completely disregards the reachability probability of certain states and invests the same computational effort for all of them. In particular, it invests the same amount of computational effort into regions which are only reached with probability 10^{-100} as in regions around the initial state s_0 .

Thus, a second approach is to sample a path through the system at random, following random actions. This approach updates states roughly proportional to the probability of being reached, which already in the finite setting yields dramatic speed-ups [34].

However, we can also use further information provided by the algorithm, namely the upper bounds. As mentioned in [9], following “promising” actions with a large upper bound proves to be beneficial, since actions with small upper bound likely are suboptimal. To extend this idea to the general domain, we need to apply a bit of care. In particular, it might be difficult to select exactly from the optimal set of actions, since already $\arg \max_{a \in Av(s)} U(s, a)$ might be very difficult to compute. Yet, it is sufficient to choose some constant $\xi > 0$ and over-approximate the set of ξ -optimal actions in a given state, randomly selecting from this set. This over-approximation can easily be performed by, for example, randomly sampling the set of available actions $Av(s)$ until we encounter an action close to the optimum (which can approximate due to our assumptions). By generating paths only using these actions, we combine the previous idea of focussing on “important” states (in terms of reachability) with an additional focus on “promising” states (in terms of upper bounds). This way, the algorithm *learns* from its experiences, using it as a guidance for future explorations.

More generally, we can easily apply more sophisticated learning approaches by interleaving it with one of the above methods. For example, by following the learning approach with probability ν and a “safe” method with probability $1 - \nu$ we still obtain a safe heuristic, since the assumption only requires limit behaviour. As such, we can combine our approach with existing, learning based algorithm by following their suggested heuristic and interleave it with some sampling runs guided by the above ideas. In other words, this means that the learning algorithm can focus on finding a reasonable solution quickly, which is then subsequently *verified* by our approach, potentially *improving* the solution in areas where the learner is performing suboptimally. On top, the (guaranteed) bounds identified by our algorithm can be used as feedback to the learning algorithm, creating a positive feedback loop, where both components improve each other’s behaviour and performance.

6 Conclusion

In this work, we have presented the first anytime algorithm to tackle the reachability problem for MDP with uncountable state- and action-spaces, giving both correctness and termination guarantees under general assumptions. The experimental evaluation of our prototype implementation shows both promising results and room for improvements.

On the theoretical side, we conjecture that **Assumption D: Absorption** can be weakened if we complement it with an automatic procedure that finds and treats problematic parts of the state space of a certain kind, similar to the collapsing approach on finite MDP

[19, 9]. Note that as the general problem is undecidable, some form of **Absorption** will remain necessary. On the practical side, we aim for a more sophisticated tool, applying our theoretical foundation to the full range of MDP, including discrete discontinuities. Moreover, we want to combine the tool with existing ways of identifying the Lipschitz constant.

References

- 1 Alessandro Abate, Saurabh Amin, Maria Prandini, John Lygeros, and Shankar Sastry. Computational approaches to reachability analysis of stochastic hybrid systems. In *HSCC*, volume 4416 of *Lecture Notes in Computer Science*, pages 4–17. Springer, 2007. doi:10.1007/978-3-540-71493-4_4.
- 2 Alessandro Abate, Joost-Pieter Katoen, John Lygeros, and Maria Prandini. Approximate model checking of stochastic hybrid systems. *Eur. J. Control*, 16(6):624–641, 2010. doi:10.3166/ejc.16.624-641.
- 3 Alessandro Abate, Maria Prandini, John Lygeros, and Shankar Sastry. Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. *Automatica*, 44(11):2724–2734, 2008. doi:10.1016/j.automatica.2008.03.027.
- 4 Christel Baier, Joachim Klein, Linda Leuschner, David Parker, and Sascha Wunderlich. Ensuring the reliability of your model checker: Interval iteration for Markov decision processes. In *CAV (1)*, volume 10426 of *Lecture Notes in Computer Science*, pages 160–180. Springer, 2017.
- 5 Dimitri Bertsekas. Convergence of discretization procedures in dynamic programming. *IEEE Transactions on Automatic Control*, 20(3):415–419, 1975.
- 6 Dimitri P Bertsekas and Steven Shreve. Stochastic optimal control: the discrete-time case, 1978.
- 7 Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Math. Oper. Res.*, 16(3):580–595, 1991. doi:10.1287/moor.16.3.580.
- 8 Patrick Billingsley. *Probability and Measure*, volume 939. John Wiley & Sons, 2012.
- 9 Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Křetínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of Markov decision processes using learning algorithms. In *ATVA*, volume 8837 of *Lecture Notes in Computer Science*, pages 98–114. Springer, 2014. doi:10.1007/978-3-319-11936-6_8.
- 10 John L. Bresina, Richard Dearden, Nicolas Meuleau, Sailesh Ramakrishnan, David E. Smith, and Richard Washington. Planning under continuous time and resource uncertainty: A challenge for AI. *CoRR*, abs/1301.0559, 2013. arXiv:1301.0559.
- 11 Debasish Chatterjee, Eugenio Cinquemani, and John Lygeros. Maximizing the probability of attaining a target prior to extinction. *Nonlinear Analysis: Hybrid Systems*, 5(2):367–381, 2011.
- 12 Krishnendu Chatterjee, Zuzana Křetínská, and Jan Křetínský. Unifying two views on multiple mean-payoff objectives in Markov decision processes. *Logical Methods in Computer Science*, 13(2), 2017. doi:10.23638/LMCS-13(2:15)2017.
- 13 Anne Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992. doi:10.1016/0890-5401(92)90048-K.
- 14 Zhengzhu Feng, Richard Dearden, Nicolas Meuleau, and Richard Washington. Dynamic programming for structured continuous Markov decision problems. In *UAI*, pages 154–161. AUAI Press, 2004. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1102&proceeding_id=20.
- 15 Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Automated verification techniques for probabilistic systems. In *SFM*, volume 6659 of *Lecture Notes in Computer Science*, pages 53–113. Springer, 2011. doi:10.1007/978-3-642-21455-4_3.

- 16 Hongfei Fu and Krishnendu Chatterjee. Termination of nondeterministic probabilistic programs. In *VMCAI*, volume 11388 of *Lecture Notes in Computer Science*, pages 468–490. Springer, 2019. doi:10.1007/978-3-030-11245-5_22.
- 17 Kush Grover, Jan Kretínský, Tobias Meggendorfer, and Maximilian Weininger. Anytime guarantees for reachability in uncountable markov decision processes. *CoRR*, abs/2008.04824, 2020. arXiv:2008.04824.
- 18 Carlos Guestrin, Milos Hauskrecht, and Branislav Kveton. Solving factored MDPs with continuous and discrete variables. In *UAI*, pages 235–242. AUAI Press, 2004. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1113&proceeding_id=20.
- 19 Serge Haddad and Benjamin Monmege. Interval iteration algorithm for MDPs and IMDPs. *Theor. Comput. Sci.*, 735:111–131, 2018. doi:10.1016/j.tcs.2016.12.003.
- 20 Sofie Haesaert, Sadegh Soudjani, and Alessandro Abate. Temporal logic control of general Markov decision processes by approximate policy refinement. In *ADHS*, volume 51(16) of *IFAC-PapersOnLine*, pages 73–78. Elsevier, 2018. doi:10.1016/j.ifacol.2018.08.013.
- 21 Sofie Haesaert, Sadegh Esmaeil Zadeh Soudjani, and Alessandro Abate. Verification of general Markov decision processes by approximate similarity relations and policy refinement. *SIAM J. Control and Optimization*, 55(4):2333–2367, 2017. doi:10.1137/16M1079397.
- 22 Arnd Hartmanns and Benjamin Lucien Kaminski. Optimistic value iteration. In *CAV (2)*, volume 12225 of *Lecture Notes in Computer Science*, pages 488–511. Springer, 2020.
- 23 Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Logically-constrained neural fitted q-iteration. In *AAMAS*, pages 2012–2014. International Foundation for Autonomous Agents and Multiagent Systems, 2019. URL: <http://dl.acm.org/citation.cfm?id=3331994>.
- 24 Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Certified reinforcement learning with logic guidance. *CoRR*, abs/1902.00778, 2019. arXiv:1902.00778.
- 25 William B. Haskell, Rahul Jain, Hiteshi Sharma, and Pengqian Yu. A universal empirical dynamic programming algorithm for continuous state MDPs. *IEEE Trans. Automat. Contr.*, 65(1):115–129, 2020. doi:10.1109/TAC.2019.2907414.
- 26 Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What’s decidable about hybrid automata? In *STOC*, pages 373–382. ACM, 1995.
- 27 Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What’s decidable about hybrid automata? *J. Comput. Syst. Sci.*, 57(1):94–124, 1998. doi:10.1006/jcss.1998.1581.
- 28 Onésimo Hernández-Lerma and Jean B Lasserre. *Discrete-time Markov control processes: basic optimality criteria*, volume 30. Springer Science & Business Media, 2012.
- 29 Ronald A Howard. *Dynamic programming and Markov processes*. John Wiley, 1960.
- 30 Mingzhang Huang, Hongfei Fu, and Krishnendu Chatterjee. New approaches for almost-sure termination of probabilistic programs. In *Program. Lang. and Sys.*, volume 11275 of *Lecture Notes in Computer Science*, pages 181–201. Springer, 2018. doi:10.1007/978-3-030-02768-1_11.
- 31 Mingzhang Huang, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. Modular verification for almost-sure termination of probabilistic programs. *Proc. ACM Program. Lang.*, 3(OOPSLA):129:1–129:29, 2019. doi:10.1145/3360555.
- 32 Manfred Jaeger, Peter Gjøøl 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. doi:10.1007/978-3-030-31784-3_5.
- 33 Benjamin Lucien Kaminski and Joost-Pieter Katoen. On the hardness of almost-sure termination. In *MFCS*, volume 9234 of *Lecture Notes in Computer Science*, pages 307–318. Springer, 2015. doi:10.1007/978-3-662-48057-1_24.
- 34 Jan Kretínský and Tobias Meggendorfer. Of cores: A partial-exploration framework for Markov decision processes. In *CONCUR*, volume 140 of *LIPICs*, pages 5:1–5:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.CONCUR.2019.5.

- 35 Ratan Lal and Pavithra Prabhakar. Bounded verification of reachability of probabilistic hybrid systems. In *QEST*, volume 11024 of *Lecture Notes in Computer Science*, pages 240–256. Springer, 2018. doi:10.1007/978-3-319-99154-2_15.
- 36 Bernard F Lamond and Abdeslem Boukhtouta. Water reservoir applications of Markov decision processes. In *Handbook of Markov decision processes*, pages 537–558. Springer, 2002.
- 37 Lihong Li and Michael L. Littman. Lazy approximation for solving continuous finite-horizon MDPs. In *AAAI*, pages 1175–1180. AAAI Press / The MIT Press, 2005. URL: <http://www.aaai.org/Library/AAAI/2005/aaai05-186.php>.
- 38 Masoud Mahootchi. *Storage system management using reinforcement learning techniques and nonlinear models*. PhD thesis, University of Waterloo, 2009.
- 39 H. Brendan McMahan, Maxim Likhachev, and Geoffrey J. Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML*, volume 119 of *ACM International Conference Proceeding Series*, pages 569–576. ACM, 2005. doi:10.1145/1102351.1102423.
- 40 Francisco S. Melo, Sean P. Meyn, and M. Isabel Ribeiro. An analysis of reinforcement learning with function approximation. In *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 664–671. ACM, 2008. doi:10.1145/1390156.1390240.
- 41 Goran Peskir and Albert Shiryaev. *Optimal stopping and free-boundary problems*. Springer, 2006.
- 42 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.
- 43 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994. doi:10.1002/9780470316887.
- 44 Tim Quatmann and Joost-Pieter Katoen. Sound value iteration. In *CAV (1)*, volume 10981 of *Lecture Notes in Computer Science*, pages 643–661. Springer, 2018.
- 45 Scott Sanner, Karina Valdivia Delgado, and Leliane Nunes de Barros. Symbolic dynamic programming for discrete and continuous state MDPs. In *UAI*, pages 643–652. AUAI Press, 2011. URL: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2223&proceeding_id=27.
- 46 Claude Elwood Shannon. Communication in the presence of noise. *Proceedings of the IRE*, 37(1):10–21, 1949.
- 47 Hiteshi Sharma, Mehdi Jafarnia-Jahromi, and Rahul Jain. Approximate relative value learning for average-reward continuous state MDPs. In *UAI*, page 341. AUAI Press, 2019. URL: <http://auai.org/uai2019/proceedings/papers/341.pdf>.
- 48 Fedor Shmarov and Paolo Zuliani. Probreach: verified probabilistic delta-reachability for stochastic hybrid systems. In *HSCC*, pages 134–139. ACM, 2015.
- 49 Sadegh Esmail Zadeh Soudjani and Alessandro Abate. Adaptive gridding for abstraction and verification of stochastic hybrid systems. In *QEST*, pages 59–68. IEEE Computer Society, 2011. doi:10.1109/QEST.2011.16.
- 50 Sean Summers and John Lygeros. Verification of discrete time stochastic hybrid systems: A stochastic reach-avoid decision problem. *Automatica*, 46(12):1951–1961, 2010. doi:10.1016/j.automatica.2010.08.006.
- 51 Ilya Tkachev, Alexandru Mereacre, Joost-Pieter Katoen, and Alessandro Abate. Quantitative automata-based controller synthesis for non-autonomous stochastic hybrid systems. In *HSCC*, pages 293–302. ACM, 2013. doi:10.1145/2461328.2461373.
- 52 Ilya Tkachev, Alexandru Mereacre, Joost-Pieter Katoen, and Alessandro Abate. Quantitative model-checking of controlled discrete-time Markov processes. *Inf. Comput.*, 253:1–35, 2017. doi:10.1016/j.ic.2016.11.006.
- 53 Hado van Hasselt. Reinforcement learning in continuous state and action spaces. In *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 207–251. Springer, 2012. doi:10.1007/978-3-642-27645-3_7.


11:20 Anytime Guarantees for Reachability in Uncountable Markov Decision Processes

- 54 Luis Gustavo Rocha Vianna, Scott Sanner, and Leliane Nunes de Barros. Continuous real time dynamic programming for discrete and continuous state MDPs. In *2014 Brazilian Conference on Intelligent Systems, BRACIS 2014, Sao Paulo, Brazil, October 18-22, 2014*, pages 134–139. IEEE Computer Society, 2014. doi:10.1109/BRACIS.2014.34.

Checking Timed Büchi Automata Emptiness Using the Local-Time Semantics

Frédéric Herbreteau ✉ 

Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400, Talence, France

B. Srivathsan ✉ 

Chennai Mathematical Institute, India
CNRS IRL 2000, ReLaX, Chennai, India

Igor Walukiewicz ✉ 

Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400, Talence, France

Abstract

We study the Büchi non-emptiness problem for networks of timed automata. Standard solutions consider the network as a monolithic timed automaton obtained as a synchronized product and build its zone graph on-the-fly under the classical global-time semantics. In the global-time semantics, all processes are assumed to have a common global timeline.

Bengtsson et al. in 1998 have proposed a local-time semantics where each process in the network moves independently according to a local timeline, and processes synchronize their timelines when they do a common action. It has been shown that the local-time semantics is equivalent to the global-time semantics for finite runs, and hence can be used for checking reachability. The local-time semantics allows computation of a local zone graph which has good independence properties and is amenable to partial-order methods. Hence local zone graphs are able to better tackle the state-space explosion due to concurrency.

In this work, we extend the results to the Büchi setting. We propose a local zone graph computation that can be coupled with a partial-order method, to solve the Büchi non-emptiness problem in timed networks. In the process, we develop a theory of regions for the local-time semantics.

2012 ACM Subject Classification Theory of computation → Verification by model checking

Keywords and phrases Timed Büchi automata, local-time semantics, zones, abstraction, partial-order reduction

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.12

Funding *Frédéric Herbreteau*: ANR project Ticktak (ANR-18-CE40-0015)

Igor Walukiewicz: ANR project Ticktac (ANR-18-CE40-0015)

1 Introduction

Timed automata [2] are a popular model for real-time systems. Typically, systems are modeled as a *network of timed automata* that communicate with each other via synchronizing actions. We are interested in verifying Büchi properties of such models: does there exist a run of the network that executes transitions from a given set infinitely often? This is called the Büchi non-emptiness problem. Model checking LTL specifications can be reduced to the Büchi non-emptiness problem. Moreover, verifying Büchi properties can be useful in trouble-shooting the model under consideration, for example, a typo in the benchmark CSMA/CD protocol model was discovered through a Büchi property verification [18]. Recent works go even further and consider synthesis questions for Büchi timed automata [3, 8].

Existing algorithms for the Büchi non-emptiness problem view the network as a single timed automaton obtained by a synchronized product, and build the so-called zone graph of this product automaton on-the-fly [29, 28, 25, 23, 18]. The main challenge lies in guaranteeing



© Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 12; pp. 12:1–12:24

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the termination of the zone graph computation. This is achieved through a *finite abstraction* of the zone graph that preserves the Büchi property. The aim of course is to get as small a graph as possible. This has been the central subject of study in timed automata verification [11, 7, 4, 21, 20]. As a matter of fact, new abstraction methods are usually first studied in the context of reachability verification and then lifted to the Büchi setting. In this paper, we continue this trend by extending a recent work on abstractions based on the *local-time semantics* [15], to the Büchi non-emptiness problem.

The local-time semantics for timed automata networks was proposed by Bengtsson *et al.* [5] with the aim of applying partial-order reduction methods that can exploit the network representation of the model to build a smaller zone graph. In the local-time semantics, each process in the network moves independently according to its local timeline which contrasts with the standard semantics where time elapses synchronously in all the processes. When processes perform a shared action, they synchronize their local timelines. This semantics gives good independence properties: for instance, if a and b are actions performed by processes P_a and P_b , an execution $(a, 2)(b, 1)$ means a happens when the local time of P_a is 2 and b happens when local time of P_b is 1. There is no “happens-before” between $(a, 2)$ and $(b, 1)$. The local-time semantics leads to a local-zone graph computation in which performing ab or ba from a local-zone leads to the same local-zone. This diamond property is essential for applying partial-order reduction methods. In [15] we have proposed abstractions for the local zone graph that can be coupled with partial-order methods to solve the reachability problem. Extending these methods to the Büchi non-emptiness problem, poses certain technical questions and requires some adaptations of the setting. We settle these issues here.

1.1 Contributions

The first question is whether the local-time semantics is sound for Büchi runs: does existence of an infinite Büchi run in the local-time semantics ensure existence of an infinite Büchi run in the usual global-time semantics? Surprisingly, we answer in the affirmative without any extra assumption. This said, let us remark that this is not true if one allows invariants in states. The solution is significantly different from the soundness argument used for reachability where the last valuation of a local run needs to be synchronized.

The next question is whether the local-zone graph is sound for Büchi runs: does an infinite Büchi run in the local-zone graph ensure existence of an infinite Büchi run in the local-time semantics. For every finite prefix, we can get a finite run in the local-time semantics. But the question of whether these prefixes can be glued together to form an infinite run is non-trivial. The same question arises in the global-time semantics as well, and there the solution makes a crucial use of Alur-Dill regions [2]. For the local-time semantics, there is no known notion of a region equivalence. We have shown [15] that in general, there can be no finite time-abstract bisimulation for the local-time semantics and proposed to restrict attention to a class of networks called bounded spread networks. Every network can be converted to a bounded spread network at the cost of reducing concurrency. In this work, we develop a finite region equivalence over the local-time semantics for bounded spread networks.

Finally, we prove that the combination of the abstraction from [15] and partial-order reduction can be suitably applied on the local-zone graph to solve the Büchi non-emptiness problem. For the argument to work we need to assume that the network is deterministic. This is usually not a strong assumption, as a network can be made deterministic by renaming actions. The proof of correctness appropriately combines the guarantees known over finite runs and the region machinery developed above.

We remark that we do not propose here a concrete POR method. Instead, we consider a POR method as an oracle that assigns a subset of edges to be explored from each node of the local-zone graph. Our work can be seen as a theoretical development that allows to plug in any POR method which is correct for the Büchi problem on untimed networks, to timed networks. Most recent works in the POR literature consider a very special case where every process is acyclic and has at most one outgoing action in every state [1, 33, 9, 22]. As a further work we would like to have equally efficient methods for more general settings as considered in works on stubborn/ample/persistent sets [30, 27, 13, 32].

1.2 Related Work

The early abstraction methods studied for timed automata depended on the maximum constant appearing in the automaton [11, 7]. These abstractions were extended to Büchi runs by Tripakis *et al.* [29, 28]. Later, superior abstractions were proposed based on the maximum constant L occurring in lower bound guards ($x > c, x \geq c$) and the maximum constant U occurring in upper bound guards ($x < c, x \leq c$) [4]. Li [25] showed that these LU abstractions can be used to solve Büchi non-emptiness problem.

An abstraction method comes with an operator \mathbf{a} that can be applied on zones Z . For reachability, it is enough to check $\mathbf{a}(Z) \subseteq \mathbf{a}(Z')$ (called a subsumption) to discard further exploration from Z and continue from Z' . On the other hand, for Büchi non-emptiness, we need to check for equality $\mathbf{a}(Z) = \mathbf{a}(Z')$. Subsumptions are instrumental in reducing the size of the graph obtained. For the Büchi problem, a restricted usage of subsumption is possible [23, 18]. However, the gains due to this restricted subsumption are less pronounced in the Büchi setting as compared to the gains achieved for reachability. There is even a concrete argument to support this statement: deciding Büchi non-emptiness starting from a zone graph with subsumption is PSPACE-hard [18]. The moral is that graphs computed for the Büchi problem are in general expected to be much larger than the reachability counterparts. In this situation it is even more interesting to use POR to reduce their size.

POR techniques have been applied for the reachability problem, but over the zone graph computed using the standard global-time semantics. There is much less independence in the global-time zone graph and the POR method needs to be restricted accordingly. Therefore some approaches limit the POR methods to parts where independent actions occur in zero time [26, 24, 6], and other approaches discover which actions remain independent either statically [10] or dynamically [17].

1.3 Outline of the Paper

In the next section we introduce networks of timed automata and their local-time semantics. Standard global-time semantics is a special case of the local-time semantics. We define the Büchi non-emptiness problem over the global-time semantics. In Section 3 we show that for the Büchi non-emptiness problem it is sound to use the local-time semantics instead of the global one. We also recall local-zones, local-zone graphs and their properties from [15]. In Section 4, we develop a theory of regions for the local-time semantics. We employ the concept of a bounded spread network from [15], and show that for such networks the number of regions is finite if we have a bound on the constants used in the guards of the transitions. Finally, in Section 5 we recall the abstraction operation $\mathbf{a}_{\approx_{LU}}^D$ from [15], introduce an abstract notion of a POR method based on a source function, and show how to obtain a finite abstract local-zone graph where we can use a POR method while retaining soundness and completeness for Büchi runs. Missing proofs are presented in appendices.

2 Preliminaries

We write \mathbb{R} , $\mathbb{R}_{\geq 0}$ and \mathbb{N} for the set of reals, non-negative reals and natural numbers, respectively. We will write 2^S for the power set of a set S . Let X be a set of real valued variables. A constraint over X is described by the grammar: $\phi := x\#c \mid \phi \wedge \phi$, where $x \in X$, $c \in \mathbb{N}$ and $\# \in \{<, \leq, =, >, \geq\}$. Let $\Phi(X)$ denote the set of all constraints over X .

A *network of timed automata* \mathcal{N} is a tuple $\langle A_1, A_2, \dots, A_k \rangle$ of k timed automata, each A_i is called a *process* or a *component* of the network. Let $Proc = \{1, \dots, k\}$ denote the set of process identifiers. Process A_i is given by $(Q_i, q_i^{init}, \Sigma_i, X_i, \Delta_i)$ consisting of a finite set Q_i of states, an initial state $q_i^{init} \in Q_i$, a finite alphabet of actions Σ_i , a finite set of clocks X_i , and a finite set of transitions $\Delta_i \subseteq Q_i \times \Sigma_i \times \Phi(X_i) \times 2^{X_i} \times Q_i$.

Transitions in A_i are of the form (p, a, g, R, q) where p and q are the source and target of the transition, $a \in \Sigma_i$ is the action, $g \in \Phi(X_i)$ is a *guard* over local clocks X_i , and $R \subseteq X_i$ is the set of local clocks of X_i that are *reset* along the transition. We assume that $Q_i \cap Q_j = \emptyset$ and $X_i \cap X_j = \emptyset$ for all distinct pairs $i, j \in \{1, \dots, k\}$. We define $\Sigma := \bigcup_{i=1}^k \Sigma_i$, $X := \bigcup_{i=1}^k X_i$ and $Q := \prod_{i=1}^k Q_i$. For $a \in \Sigma$, we write $dom(a) := \{i \in \{1, \dots, k\} \mid a \in \Sigma_i\}$. For $q = (q_1, \dots, q_k)$ in Q , we write $q(i)$ for q_i .

We say that \mathcal{N} is *deterministic* if for every component A_i and for every action a , there is at most one local transition (p, a, g, R, q) from every local state $p \in Q_i$. We will assume deterministic networks in Section 5.

There are two ways to describe the semantics of a network: one in which all the components share a common timeline (global-time semantics), and another where each of them work with a local timeline (local-time semantics). We define the local-time semantics and view global-time semantics as a special case.

2.1 Local-Time Semantics

Fix a network $\mathcal{N} = \langle A_1, \dots, A_k \rangle$ for the rest of the section. We assume that each A_i has a special clock t_i called the *reference clock* of process A_i . Intuitively, it represents the local time of process A_i . Let $T = \{t_1, \dots, t_k\}$ denote the set of all reference clocks. A valuation $v : X \cup T \rightarrow \mathbb{R}$ is a function that maps each variable in $X \cup T$ to a real number under the condition that $v(t_i) \geq v(x_i)$ for all $x_i \in X_i$. The value $v(x_i)$ represents the local time at process A_i when x_i was last reset. This explains why we require $v(t_i) \geq v(x_i)$. The value of clock x_i is then obtained as $v(t_i) - v(x_i)$. This semantics that keeps reset time points instead of clock ages has previously been introduced in [12] in the global-time setting. In the rest of the document, we use the notation $v(x - y)$ for $v(x) - v(y)$. The semantics relies on two operations.

The first one is a *local-time elapse*. Given a valuation v , a delay $\delta_i \in \mathbb{R}$ and a process i , the valuation $v +_i \delta_i$ describes a *local delay* of δ_i units at process i . It is given by $(v +_i \delta_i)(t_i) = v(t_i) + \delta_i$ and $(v +_i \delta_i)(x) = v(x)$ for all other variables x . Notice that $((v +_i \delta_i) +_j \delta_j)$ is the same as $((v +_j \delta_j) +_i \delta_i)$: the order in which we sum the local delays does not matter. We extend this notion to a tuple $\Delta := (\delta_1, \dots, \delta_k) \in \mathbb{R}_{\geq 0}^k$ of delays, one for each process: $v + \Delta$ is the valuation $v +_1 \delta_1 +_2 \delta_2 \cdots +_k \delta_k$.

The next operation is *clock reset*. In the local-time interpretation, resetting a clock $x_i \in X_i$ amounts to updating its value to the local-time of i given by t_i . Given valuation v , and a set of clocks $R \subseteq X$, we write $v[R]$ to be the valuation obtained as $(v[R])(x) = v(t_i)$ when $x \in R \cap X_i$ for some $i \in \{1, \dots, k\}$ and $v[R](x) = v(x)$ otherwise. Valuation v is said to satisfy a constraint $x\#c$ for $x \in X_i$ if $v(t_i - x)\#c$. We write $v \models (x\#c)$ in this case. A valuation satisfies a conjunction of constraints $\phi_1 \wedge \phi_2$ if $v \models \phi_1$ and $v \models \phi_2$.

A *configuration* of a network is a pair (q, v) where $q \in Q$ and v is a valuation. Recall that we have defined Q to be the product of the local states Q_i . A valuation v is said to be *initial* if $v(x) = v(y)$ for all $x, y \in X \cup T$: all timelines are synchronized and the constraint $x = 0$ holds for every clock. A configuration (q, v) is initial if $q = (q_1^{init}, \dots, q_k^{init})$ and v is an initial valuation.

There are two kinds of transitions between configurations: local delays and action transitions. From a configuration (q, v) there is a local delay transition $(q, v) \xrightarrow{\Delta} (q, v + \Delta)$ for each $\Delta \in \mathbb{R}_{\geq 0}^k$. For each $b \in \Sigma$, a b -transition is a tuple $\{(q_i, b, g_i, R_i, q'_i)\}_{i \in \text{dom}(b)}$ of local transitions one from each process in its domain. From (q, v) , we have an action transition $(q, v) \xrightarrow{b} (q', v')$ if there exists a b -transition $\{(q_i, b, g_i, R_i, q'_i)\}_{i \in \text{dom}(b)}$ such that:

- source states match: $q(i) = q_i$ for all $i \in \text{dom}(b)$,
- valuation satisfies guard: $v \models g_i$ for all $i \in \text{dom}(b)$,
- all processes in $\text{dom}(b)$ are synchronized: $v(t_i) = v(t_j)$ for all $i, j \in \text{dom}(b)$,
- resets are performed: $v' = v[\bigcup_{i \in \text{dom}(b)} R_i]$,
- target states are reached: $q'(i) = q'_i$ if $i \in \text{dom}(b)$ and $q'(i) = q(i)$ otherwise.

The important point is that when a common action is performed, the local times of all the processes in its domain are the same.

We will write $(q, v) \xrightarrow{\Delta, b} (q', v')$ for $(q, v) \xrightarrow{\Delta} (q, v) \xrightarrow{b} (q', v')$, a local delay Δ transition followed by an action b . Observe that Δ is determined by v and v' because $\delta_p = v'(t_p) - v(t_p)$ where t_p is the reference clock of process p . A *run* is an infinite sequence $(q_0, v_0) \xrightarrow{\Delta_0, b_0} (q_1, v_1) \xrightarrow{\Delta_1, b_1} \dots$ of transitions starting from an initial configuration (q_0, v_0) . A finite run is defined similarly: $(q_0, v_0) \xrightarrow{\Delta_0, b_0} (q_1, v_1) \xrightarrow{\Delta_1, b_1} \dots (q_n, v_n) \xrightarrow{\Delta_n} (q'_n, v'_n)$. Observe that finite runs have a final delay. We write $(q, v) \xrightarrow{\sigma} (q', v')$ to say that there is a finite run on a sequence of actions σ .

2.2 The Büchi Non-Emptiness Problem

Global-time semantics is a local-time semantics restricted to *synchronized valuations*. A valuation v is said to be *synchronized* if $v(t_p) = v(t_q)$ for all processes p, q . This implies that in every delay $(\delta_1, \dots, \delta_k)$ that is part of the global-time semantics, we have $\delta_1 = \delta_2 = \dots = \delta_k$. The global-time semantics obtained this way is the usual semantics that is used in tools and studies that involve networks of timed automata. To make a distinction, we refer to runs in the local-time semantics as local runs and runs in the global-time semantics as global runs. Clearly, a global run is also a local run.

► **Definition 1** (Büchi non-emptiness problem). *Given a network \mathcal{N} and a set of actions $F \subseteq \Sigma$, decide if \mathcal{N} has a global run with infinitely many occurrences of actions from F .*

In the case of finite runs, the correspondence between local and global semantics is well-known. Indeed there is a local run to a configuration (q, v) with a synchronized valuation v if and only if there exists a global run to (q, v) that follows the same transitions, although in a different order. This is captured by the notion of independent actions and traces.

► **Definition 2** (Independence). *Two actions $a, b \in \Sigma$ are said to be independent if $\text{dom}(a) \cap \text{dom}(b) = \emptyset$. Two sequences of actions $u, w \in \Sigma^*$ are trace equivalent, $u \sim w$, if one of them can be obtained from the other by permuting adjacent independent actions.*

► **Lemma 3** ([16]). *Let v, v' be synchronized valuations, and let $(q, v) \xrightarrow{u} (q', v')$ be a local run. Then there exists a global run $(q, v) \xrightarrow{w} (q', v')$ such that $u \sim w$.*

► **Remark 4.** Putting Büchi conditions on transitions makes them trace invariant which is important if we want to do partial-order methods: if u has infinitely many actions from F , then so does every $w \sim u$, which may not be true with Büchi conditions on states.

Moreover, the local-time semantics enjoys a very nice property: any two independent actions commute as stated by the following lemma.

► **Lemma 5** (Diamond property). [16] *Let $a, b \in \Sigma$ with $\text{dom}(a) \cap \text{dom}(b) = \emptyset$. If $(q, v) \xrightarrow{ab} (q', v')$ then $(q, v) \xrightarrow{ba} (q', v')$.*

Observe that this commutation property does not hold in the global-time semantics since delays are synchronous in all processes. Our motivation for using local time over global time comes from the fact that local time allows to reorder the actions in a run, hence using partial-order reduction techniques becomes possible.

3 Büchi Runs in the Local-Time Semantics

In this section, we show that using the local-time semantics is sound for the Büchi emptiness problem. Hence the local-time semantics looks appealing as it enables to use partial-order reduction techniques. However, verification algorithms cannot work directly from the state-space of the network as it is uncountable. We will then introduce the local zone graph as a symbolic representation of the state-space of timed networks in the local time semantics.

3.1 The Local-Time Semantics is Sound

When state-reachability is considered the soundness of the local-time semantics follows from Lemma 3. However, it is based on the notion of independence, Definition 2, that is not adequate for infinite runs. The point is that for infinite runs we may need an infinite number of permutations to get w from u . The more general definition refers to partial orders defined by runs. These partial orders are often called traces. The other obstacle in repeating Lemma 3 is that the lemma refers to runs ending in synchronized valuations, while for infinite runs we do not have final valuations. For these two reasons the soundness argument is more involved than that of reachability.

We start with an intuition and an illustrating example. To get a global run from a local run, the natural idea would be to re-order the events based on the time-stamps. For example, if we have a finite sequence $(a, 2)(b, 1)(c, 3)(a, 2.5)(b, 1.5)$ where the number represents the local time when the action occurred, then we get a reordered sequence $(b, 1)(b, 1.5)(a, 2)(a, 2.5)(c, 3)$. It can then be argued that this sequence has a global run where each clock can be delayed up to the next action to be read. For the case of infinite runs, consider the following example that consists of two processes A and B .

$$\mathbf{A} : \longrightarrow \textcircled{q} \longleftarrow (x < 1), a \qquad \mathbf{B} : \longrightarrow \textcircled{r} \longleftarrow (y = 1), b, \{y\}$$

Consider an infinite local run:

$$(a, \frac{1}{2}) (b, 1) (a, \frac{1}{2} + \frac{1}{2^2}) (b, 2) \cdots (a, \frac{1}{2} + \cdots + \frac{1}{2^i}) (b, i) \cdots$$

Notice that all the a actions happen before global time 1, and b actions start from 1. Therefore, a re-ordering of the events based on the time-stamps gives an infinite sequence of a 's "followed by" an infinite sequence of b 's. This does not correspond to a global run. However, if a was accepting, we have a global run $(a, \frac{1}{2})(a, \frac{1}{2} + \frac{1}{2^2}) \cdots$ that is accepting. Similarly, if b was

accepting, we have a global run $(b, 1)(b, 2) \dots$. In each of these runs, the other process plays no role except for elapsing time. Observe that the absence of state invariants is crucial here as the infinite sequence of b 's would not be feasible in the global time semantics if state q of process A had invariant $(x < 1)$.

In general, when we have a local run, we can reorder it based on timestamps to get a sequence which may have a block of infinite events (say between 0 and 1), followed by another block of events (say between 1 and 2), and so on. However, the processes that participate infinitely often in some block do not participate in the future blocks. This means there can be only finitely many such blocks. Moreover, if the original run is Büchi accepting, then there is some block (along with some events in blocks before it) that gives a global run which is Büchi accepting. We formalize this intuition below.

For an infinite sequence $w \in \Sigma^\omega$ we define a *trace of w* as a labelled partial order $T_\sigma = \langle \mathbb{N}, \lambda_\sigma, \preceq_\sigma \rangle$ where $\lambda_\sigma(i) = w_i$, and \preceq_σ the smallest transitive relation such that $i \preceq_\sigma j$ if $\text{dom}(w_i) \cap \text{dom}(w_j) \neq \emptyset$ and $i \leq j$. Observe that \preceq_σ is reflexive.

► **Definition 6.** *Two infinite runs $u, w \in \Sigma^\omega$ are trace equivalent, $u \sim w$, if the traces T_u and T_w are isomorphic*

We extend this notion to runs. Consider a local run

$$\sigma = (q_0, v_0) \xrightarrow{\Delta_0} (q_0, v'_0) \xrightarrow{b_0} (q_1, v_1) \xrightarrow{\Delta_1} (q_1, v'_1) \xrightarrow{b_1} (q_2, v_2) \xrightarrow{\Delta_2} \dots$$

A *trace of σ* is a labelled partial order $T_\sigma = \langle \mathbb{N}, \lambda_\sigma, \preceq_\sigma \rangle$ where $\lambda_\sigma(i) = (q_i, v'_i, b_i)$, and as before \preceq_σ the smallest transitive relation such that $i \preceq_\sigma j$ if $\text{dom}(b_i) \cap \text{dom}(b_j) \neq \emptyset$ and $i \leq j$. Observe that we take valuation v'_i in the label as it ensures that b_i is enabled in (q_i, v'_i) .

To connect local and global time semantics we will rearrange actions depending on their local time. We use $\theta_\sigma(i)$ for the local time of execution of action b_i in the trace σ ; it is given by $\theta_\sigma(i) = v'_i(t_p)$ where $p \in \text{dom}(b_i)$. Recall that by definition of an action transition, $v'_i(p) = v'_i(q)$ for all $p, q \in \text{dom}(b_i)$.

► **Lemma 7.** *If $i \preceq_\sigma j$ then $\theta_\sigma(i) \leq \theta_\sigma(j)$.*

A *trace prefix T'* of a trace T_σ is a restriction of T_σ to some \preceq_σ -downward closed subset of \mathbb{N} . A *linearization* of a trace T_σ is a bijection $f : \mathbb{N} \rightarrow \mathbb{N}$ such that if $f(i) \preceq_\sigma f(j)$ then $i \leq j$. This means that the order of elements in a linearization should respect the order in T_σ . Observe that the sequence given by f should use all elements of T_σ , because f is bijective. The next lemma says that every linearization yields a local run.

► **Lemma 8.** *For every T' a prefix of T_σ , every linearization f of T' gives a local run.*

The next result states the desired soundness property of the local-time semantics with respect to the global semantics. As every global run is a local run, one direction of the proposition is easy. The other side consists in finding a prefix of T_σ from which we can build a global run as explained in the example above.

► **Proposition 9.** *Consider a network of timed automata \mathcal{N} . There is a Büchi local run of \mathcal{N} iff there is a Büchi global run of \mathcal{N} .*

3.2 Local-Zone Graphs

Thanks to Proposition 9, we know that we can safely use the local-time semantics to detect Büchi runs. However, we cannot solve the Büchi non-emptiness problem by an exploration of the state-space of timed automata as it is uncountable. Algorithms for timed automata

work with sets of configurations sharing the same discrete state. We copy this approach to our setting. To start off, we lift operations on individual valuations to sets of valuations. For a set of local valuations W , define:

- $\text{local-elapsed}(W) := \{v + \Delta \mid v \in W, \Delta \in \mathbb{R}_{\geq 0}^k\}$,
- $W[R] := \{v[R] \mid v \in W\}$, for a set of clocks $R \subseteq X$.
- $W \cap g := \{v \mid v \models g\}$ for a guard g .

Next, the transition relation on configurations can be lifted to sets. We write $(q, W) \xrightarrow{b} (q', W')$ when there exists a b -transition $\{(q_i, b, g_i, R_i, q'_i)\}_{i \in \text{dom}(b)}$ such that:

- source states match: $q(i) = q_i$ for all $i \in \text{dom}(b)$,
- target states are reached: $q'(i) = q'_i$ for all $i \in \text{dom}(b)$ and $q'(i) = q(i)$ otherwise,
- $W' = \text{local-elapsed}(W_2)$ where $W_2 = W_1 \cup_{p \in \text{dom}(b)} R_p$ with $W_1 = W \cap (\bigwedge_{p \in \text{dom}(b)} g_p \wedge \bigwedge \{t_p = t_q \mid p, q \in \text{dom}(b)\})$, and W' is not empty.

We will call \xrightarrow{b} a symbolic transition. We write $(q, W) \xrightarrow{b_1 \dots b_n} (q_n, W_n)$ if there is a sequence of symbolic transitions $(q, W) \xrightarrow{b_1} (q_1, W_1) \dots \xrightarrow{b_n} (q_n, W_n)$.

Local-zones are special sets of valuations that occur naturally while computing the reachable configurations. A local-zone is a set of valuations described by a conjunction of constraints of the form $x - y \# c$ where $x, y \in X \cup T$, $c \in \mathbb{Z}$ and $\# \in \{<, \leq\}$. Local-zones can be efficiently represented using Difference Bound Matrices (DBMs). It can be shown that for a local-zone Z , the sets $\text{local-elapsed}(Z)$, $Z[R]$ and $Z \cap g$ (intersection with guard) are local-zones [5, 16]. This leads to the definition of a local-zone graph that captures the reachable configurations of a network.

► **Definition 10** (Local-zone graph $\text{LZG}(\mathcal{N})$). *The local-zone graph $\text{LZG}(\mathcal{N})$ of a network \mathcal{N} is a transition system whose nodes are of the form (q, Z) where q is a state of the network, and Z is a local-zone. The initial node is (q_0, Z_0) with $Z_0 = \text{local-elapsed}(V_0)$ where V_0 is the set of initial valuations (given by the local-zone $\bigwedge_{x, y \in X \cup T} x - y = 0$) and $q_0 = (q_1^{\text{init}}, \dots, q_k^{\text{init}})$. The transitions are given by the symbolic transition relation $(q, Z) \xrightarrow{b} (q', Z')$.*

The next lemma relates transitions over valuations and zones. Proof follows from the definition of symbolic transitions. If $Z = \text{local-elapsed}(Z)$ then Z is *time-elapsed*.

► **Lemma 11.** *For every network of timed automata and every action b :*

pre-property: *If $(q, v) \xrightarrow{b} (q', v')$ and $v \in Z$ for some time-elapsed local-zone Z then $(q, Z) \xrightarrow{b} (q', Z')$ and $v' \in Z'$ for some local-zone Z' .*

post-property: *If $(q, Z) \xrightarrow{b} (q', Z')$ and $v' \in Z'$ for local-zones Z, Z' , then $(q, v) \xrightarrow{b} (q', v')$ for some $v \in Z$.*

The initial zone is time-elapsed. By definition of the symbolic transition, every zone reachable by \xrightarrow{b} transitions is also time-elapsed. Using this observation along with the pre- and post-properties of Lemma 11, we get the following theorem.

► **Theorem 12** ([16, 5]). *For a given network \mathcal{N} , there is a run of \mathcal{N} reaching a state q iff there is a path in $\text{LZG}(\mathcal{N})$ from the initial node to a node (q, Z) .*

This shows soundness and completeness of the local-zone graph with respect to state reachability. In particular, soundness follows from the post-property and completeness follows from the pre-property. Interestingly, these two properties are not sufficient to show soundness for infinite runs. In other words, does an infinite run in the local-zone graph imply existence of an infinite run in the local-time semantics? From the post-property, each prefix of the infinite run in the local-zone graph leads to corresponding finite local-time run in the network. However the problem of forming an infinite run from these prefixes is non-trivial.

In the global-time settings, the proof of soundness crucially relies on Alur&Dill's finite region bisimulation. However, there is no finite abstraction that is sound, complete and that preserves all runs on the local zone graph [15]. This last property is crucial to apply partial-order reduction techniques. In the next section, we consider a subclass of timed networks for which we can define a finite region abstraction in the local-time settings.

4 A Region Equivalence for Bounded Spread Valuations

In this section we recall the notion of bounded spread networks of timed automata [15]. We then define a region equivalence \equiv_M^D for such networks.

4.1 Bounded Spread Networks

We consider networks of timed automata where every feasible sequence of actions can be done with bounded desynchronisation between the processes.

► **Definition 13.** *Let $D \in \mathbb{N}$. A valuation v is said to have spread D if $|v(t_p - t_q)| \leq D$ for all processes p, q . A run $(q_0, v_0) \xrightarrow{\Delta_0, b_0} (q_1, v_1) \xrightarrow{\Delta_1, b_1} \dots$ has spread D if v_i and $v_i + \Delta_i$ have spread D for all $i \geq 0$. A network \mathcal{N} has spread D if for every run $(q_0, v_0) \xrightarrow{\Delta_0, b_0} (q_1, v_1) \xrightarrow{\Delta_1, b_1} \dots$, there exists a D -spread run $(q_0, v_0) \xrightarrow{\Delta'_0, b_0} (q_1, v_1) \xrightarrow{\Delta'_1, b_1} \dots$ over the same sequence of actions, but with possibly different delays.*

It has been shown that every network can be converted into a D spread network for any arbitrary $D \geq 1$, by adding extra synchronizations [15]. Moreover, for D -spread networks it is possible to get a finite abstraction of the local zone graph, by making use of simulations.

4.2 A Finite Region Bisimulation

A *time-abstract simulation relation* on the local semantics is a reflexive and transitive relation between configurations having the same control state. Two conditions need to be satisfied when $(q, v) \preceq (q', v')$: (1) for every local delay Δ there exists Δ' such that $(q, v + \Delta) \preceq (q', v' + \Delta')$, and (2) for every transition $(q, v) \xrightarrow{b} (q_1, v_1)$ there exists a transition $(q', v') \xrightarrow{b} (q_1, v'_1)$ such that $(q_1, v_1) \preceq (q_1, v'_1)$. When $\Delta = \Delta'$, we simply call \preceq a simulation relation. For technical convenience, we will use (time-abstract) simulation relations that do not rely on the control state and depend only on the valuations. Hence we will write it as $v \preceq v'$, a relation over valuations and use its straightforward extension to configurations: $(q, v) \preceq (q', v')$ if $v \preceq v'$. The relation \preceq is a (time-abstract) bisimulation relation if both \preceq and \preceq^{-1} are (time-abstract) simulation relations.

The region equivalence of Alur and Dill [2] is a fundamental concept in the global-time semantics that leads to a finite region automaton recognizing the untimed behaviour of the system. This has been cornerstone of several decidability results for timed automata. In the global-time semantics two valuations are made region equivalent with respect to a constant M if for every clock, the values given by the two valuations lie in one of the intervals $[0], (0, 1), [1], \dots, [M], (M, \infty)$ and the ordering of fractional parts of clocks less than M is the same in both valuations. In the local-time setting, there is an additional challenge. While in the global-time semantics, when a clock goes beyond M , its actual value is irrelevant, it is not the case in the local-time semantics. Indeed, if the difference $t_p - t_q > M$ we cannot forget the actual value, since t_q can elapse some local time and bring the difference $t_p - t_q$ to something lesser than M . This is a fundamental difference between the two semantics,

12:10 Checking Timed Büchi Automata Emptiness Using the Local-Time Semantics

and this is the basic reason that leads to the result of [15] that there is no finite simulation for the local-time semantics. This is why we need to restrict to D -spread valuations v where $|v(t_p - t_q)| \leq D$. We show that with this restriction there is an adequate notion of a region equivalence. We proceed in two steps: first we define when two local valuations are “close-enough”, next we factor in the maximum constant M and bounded spread D to get a finite time abstract bisimulation. Some of these results appear in [14].

For $x \in \mathbb{R}$, we write $\lfloor x \rfloor$ for the greatest integer that is lesser than or equal to x . We will write $\{x\}$ for $x - \lfloor x \rfloor$, the fractional part of x starting from $\lfloor x \rfloor$. For example, $\lfloor 4.2 \rfloor = 4$, $\{4.2\} = 0.2$ and $\lfloor -4.2 \rfloor = -5$, $\{-4.2\} = 0.8$. Notice that $0 \leq \{x\} < 1$ for all $x \in \mathbb{R}$.

► **Definition 14.** For local valuations v and v' , we define $v \approx^* v'$ if for all pairs of clocks $x, y \in X \cup T$ (including reference clocks), we have $\lfloor v(x - y) \rfloor = \lfloor v'(x - y) \rfloor$.

Notice that the above definition does not explicitly make use of $\{v(x - y)\}$. Some relation between fractional values gets derived through the definition. For example, suppose $v(x - y) = 1$, then $v(y - x)$ is -1 . This will ensure $v'(x - y) = 1$ and $v'(y - x) = -1$. But if $v(x - y) = 1.5$, then $v(y - x) = -1.5$, and in particular $\lfloor v(y - x) \rfloor = \lfloor v'(y - x) \rfloor = -2$. This will say that $1 < v'(x - y) < 2$ and $-2 < v'(y - x) < -1$. We will precisely derive some useful properties later. Before that, we modify the definition to account for the maximum constant.

► **Definition 15** ((M, D) -equivalence). Let $M : X \rightarrow \mathbb{N} \cup \{-\infty\}$ be a bounds function mapping each process clock to a non-negative constant or $-\infty$ when the value of the clock is irrelevant. Let $D \in \mathbb{N}$ denote a spread. For a local valuation v , let $\text{Bounded}(v) = \bigcup_{p \in \text{Proc}} \{t_p\} \cup \{x \in X_p \mid v(t_p - x) \leq M(x)\}$. Notice that $\text{Bounded}(v)$ contains all the clocks that have a value below bound M in v as well as all reference clocks.

Two D -spread local valuations v, v' are (M, D) -equivalent, denoted as $v \equiv_M^D v'$, if

- $\text{Bounded}(v) = \text{Bounded}(v')$
- $v|_B \approx^* v'|_B$ where $v|_B$ and $v'|_B$ denote the valuations v and v' restricted to clocks in $B = \text{Bounded}(v)$.

For a D -spread valuation v , we write $[v]_M^D$ to denote the equivalence class of v under \equiv_M^D and refer to it as the (M, D) -region of v .

Intuitively, the above definition says that $v \equiv_M^D v'$ if the bounded part of v and v' are close enough. This is in the same spirit as in the global-time semantics. Now the goal is to show that \equiv_M^D is a time-abstract bisimulation. The most difficult part is to prove that for all local delays Δ , there exist local delays Δ' such that $v + \Delta \equiv_M^D v' + \Delta'$. This is shown in the next lemma. We denote by $\xrightarrow{\delta}_p$ a delay of δ time units in process A_p .

► **Lemma 16.** Let $v \approx^* v'$. For every local delay $v \xrightarrow{\delta}_p u$, there exists a δ' such that $v' \xrightarrow{\delta'}_p u'$ where $u \approx^* u'$.

The next task is to show that if M is appropriately chosen, the \equiv_M^D equivalence also preserves actions. We say that a network \mathcal{N} conforms to bounds function M if every constraint $x \sim c$ in \mathcal{N} satisfies $c \leq M(x)$.

► **Lemma 17.** Let v, v' be D -spread valuations such that $v \equiv_M^D v'$. Let \mathcal{N} be a network that conforms to M . For every action transition $(q, v) \xrightarrow{b} (q_1, v_1)$ we have $(q, v') \xrightarrow{b} (q_1, v'_1)$ such that $v_1 \equiv_M^D v'_1$.

► **Corollary 18.** *Let \mathcal{N} be a network that conforms to M and let $v \equiv_M^D v'$ be D -spread valuations. For every $(q, v) \xrightarrow{\Delta} \xrightarrow{b} (q_1, v_1)$ such that $v + \Delta$ is D -spread, there exists a Δ' such that $v' + \Delta'$ is D -spread and $(q, v') \xrightarrow{\Delta'} \xrightarrow{b} (q_1, v'_1)$ with $v_1 \equiv_M^D v'_1$.*

The corollary shows that \equiv_M^D is a time-abstract bisimulation on the local semantics restricted to D -spread configurations. This also motivates the following definition of a region graph obtained as a quotient of the \equiv_M^D equivalence. Recall that the initial valuations are given by $\{v \mid v(x) = v(y) \text{ for all clocks } x, y\}$. Hence by definition of \equiv_M^D equivalence all of them fall in one equivalence class.

► **Definition 19** ((M, D) -region graph). *Let \mathcal{N} be a network. A node of an (M, D) -region graph of \mathcal{N} is of the form $(q, [v]_M^D)$ where q is a state of \mathcal{N} and v is a D -spread valuation. There is a transition $(q, [v]_M^D) \xrightarrow{b} (q_1, [v_1]_M^D)$ if $(q, v) \xrightarrow{\Delta} \xrightarrow{b} (q_1, v_1)$ for some local delay Δ such that $v + \Delta$ is D -spread. The initial node is $(q_0, [v_0]_M^D)$ where v_0 is any initial valuation and q_0 is the tuple of initial states.*

► **Theorem 20.** *Let \mathcal{N} be a network that conforms to bounds function M . Then:*

- *For every D -spread local run $(q_0, v_0) \xrightarrow{\Delta_0, b_0} (q_1, v_1) \cdots$, there exists a run $(q_0, [v_0]_M^D) \xrightarrow{b} (q_1, [v_1]_M^D) \cdots$ in the (M, D) -region graph.*
- *For every run $(q_0, [v_0]_M^D) \xrightarrow{b_0} (q_1, [v_1]_M^D) \cdots$ in the (M, D) -region graph, there exists a D -spread local run $(q_0, v'_0) \xrightarrow{\Delta_0, b_0} (q_1, v'_1) \cdots$ such that $v'_i \in [v_i]_M^D$ for all $i \geq 0$.*
- *The (M, D) -region graph is finite.*

5 Abstraction and Partial-Order Reduction for Büchi Runs

The goal of this section is to make use of the local zone graph (Definition 10) for solving the Büchi non-emptiness problem. We will start by showing that the local zone graph $\text{LZG}(\mathcal{N})$ is sound and complete for infinite runs of bounded spread networks (Proposition 21). This is only the beginning because: (i) the local zone graph is still potentially infinite, and (ii) the statement does not talk about partial-order reduction. In the next step we introduce a quasi-abstraction α_{LZG}^D , and a partial-order approach. Then, we show that their combination maintains correctness for Büchi runs.

► **Proposition 21.** *Let \mathcal{N} be a D -spread network. There is an infinite D -spread local run $(q_0, v_0) \xrightarrow{\Delta_0, b_0} (q_1, v_1) \xrightarrow{\Delta_1, b_1} \cdots$ in \mathcal{N} iff there is an infinite path $(q_0, Z_0) \xrightarrow{b_0} (q_1, Z_1) \xrightarrow{b_1} \cdots$ in $\text{LZG}(\mathcal{N})$.*

Proof. Left-to-right direction follows from the pre-property of local zone graph, Lemma 11. We focus on the right-to-left direction.

Let S_i be the set of all D -spread valuations $u_i \in Z_i$ such that there is a D -spread run as below leading to u_i :

$$(q_0, u_0) \xrightarrow{\Delta'_0} \xrightarrow{b_0} (q_1, u_1) \xrightarrow{\Delta'_1} \xrightarrow{b_1} \cdots \xrightarrow{\Delta'_{i-1}} \xrightarrow{b_{i-1}} (q_i, u_i)$$

with u_0 an initial local valuation. The set S_i need not contain all D -spread valuations of Z_i . Consider some D -spread valuation v of Z_i . Due to the post-property, it has some run leading to it, not necessarily D -spread. As the network is D -spread, there is a corresponding D -spread run over the same sequence of actions. However this run may not end up in the same valuation v .

Let us come back to the D -spread run given above. Due to the pre-property of local zones, we have $u_k \in Z_k$ for all $0 \leq k \leq i$. As \mathcal{N} is D -spread, S_i is indeed non-empty, for all $i \geq 0$. In fact, each u_k in the above run belongs to S_k as the prefix with actions $b_0 \cdots b_{k-1}$ is a D -spread run leading to (q_k, u_k) . Therefore, for every $u_{i+1} \in S_{i+1}$, there exists a $u_i \in S_i$ such that $(q_i, u_i) \xrightarrow{\Delta} \xrightarrow{b_i} (q_{i+1}, u_{i+1})$ for some local delay Δ'_i .

Construct a graph with nodes $(i, q_i, [u_i]_M^D)$ for each $u_i \in S_i$. Add an edge $(i, q_i, [u_i]_M^D) \rightarrow (i+1, q_{i+1}, [u_{i+1}]_M^D)$ if $(q_i, u_i) \xrightarrow{\Delta} \xrightarrow{b_i} (q_{i+1}, u_{i+1})$. Due to the discussion in the previous paragraph, every node has a predecessor. Moreover, by Theorem 20, there are finitely many (M, D) -regions, so this graph is finitely branching. Hence there is an infinite path in this graph. This path corresponds to an infinite path in the (M, D) -region graph. Thanks to Theorem 20, this can be instantiated into an infinite D -spread local run \blacktriangleleft

5.1 Abstractions and Partial-Order Methods

We recall some notions from [15]. A quasi-abstraction \mathbf{a} is a function that maps each zone Z to a set of valuations $\mathbf{a}(Z)$ such that $\mathbf{a}(\mathbf{a}(Z)) = \mathbf{a}(Z)$. A finite quasi-abstraction function $\mathbf{a}_{\preceq_{LU}^D}^D$ has been studied in the context of reachability [15]. It is based on a preorder relation between local valuations.

► **Definition 22** (The \preceq_{LU}^* -preorder). *Let $L : X \rightarrow \{-\infty\} \cup \mathbb{N}$ and $U : X \rightarrow \{-\infty\} \cup \mathbb{N}$ be two functions. For two valuations v and v' , we say $v \preceq_{LU}^* v'$ if:*

- $v(t_p - t_q) = v'(t_p - t_q)$ for all processes p, q
- for all processes p and all $x \in X_p$,
 - $v(t_p - x) \leq U(x)$ implies $v'(t_p - x) \leq v(t_p - x)$,
 - $v(t_p - x) \leq L(x)$ implies $v'(t_p - x) \geq v(t_p - x)$,
 - $v(t_p - x) > L(x)$ implies $v'(t_p - x) > L(x)$

Notice that if v is a D -spread valuation and if $v \preceq_{LU}^* v'$, valuation v' is also D -spread. Here is a known result about \preceq_{LU}^* . We say that a network \mathcal{N} conforms to bound functions L and U if for every process clock x we have $L(x) \geq c$ for every lower bound guard $x \geq c, x > c$ occurring in \mathcal{N} , and $U(x) \geq c$ for every upper bound guard $x \leq c, x < c$ in \mathcal{N} .

► **Lemma 23** ([15]). *Let \mathcal{N} be a D -spread network that conforms to given LU bounds. The \preceq_{LU}^* pre-order is a simulation on the local semantics of \mathcal{N} : if $v \preceq_{LU}^* v'$ and $(q, v) \xrightarrow{\Delta, b} (q_1, v_1)$ then $(q, v') \xrightarrow{\Delta, b} (q_1, v'_1)$ and $v_1 \preceq_{LU}^* v'_1$.*

The \preceq_{LU}^* relation is now lifted to zones, but restricted to D -spread valuations.

► **Definition 24** ($\mathbf{a}_{\preceq_{LU}^D}^D$ -quasi-abstraction). *For a zone Z , we define $\text{spread}_D(Z) = \{v \in Z \mid v \text{ has spread } D\}$. We define $\mathbf{a}_{\preceq_{LU}^D}^D(Z) = \{v \mid \exists v' \in \text{spread}_D(Z) \text{ such that } v \preceq_{LU}^* v'\}$.*

The $\mathbf{a}_{\preceq_{LU}^D}^D$ operator can be used to give a finite abstraction of the local zone graph $\text{LZG}(\mathcal{N})$, by truncating exploration of Z if $\mathbf{a}_{\preceq_{LU}^D}^D(Z) \subseteq \mathbf{a}_{\preceq_{LU}^D}^D(Z')$ and continuing the exploration from Z' . The operation $\mathbf{a}_{\preceq_{LU}^D}^D(Z) \subseteq \mathbf{a}_{\preceq_{LU}^D}^D(Z')$ is known as subsumption in the literature [23, 18]. For Büchi non-emptiness, subsumptions cannot be used directly since we need to find cycles [23, 18]. We will define an abstraction of the local zone graph that makes use of equality with respect to $\mathbf{a}_{\preceq_{LU}^D}^D$. Notice that the equality $\mathbf{a}_{\preceq_{LU}^D}^D(Z) = \mathbf{a}_{\preceq_{LU}^D}^D(Z')$ can be checked efficiently in time $\mathcal{O}((|X| + |T|)^2)$ [15]. We will first present our view of partial-order methods and then combine this with the $\mathbf{a}_{\preceq_{LU}^D}^D$ operator in our new abstraction of the local zone graph.

We describe a generic approach to partial-order methods on local zone graphs. Then our main result will say that once we have a method that works on $\text{LZG}(\mathcal{N})$, we can use the same method on a finite abstraction of $\text{LZG}(\mathcal{N})$ obtained using $\mathfrak{a}_{\approx LU}^D$.

We formalize what it means to have a partial-order method on $\text{LZG}(\mathcal{N})$ using a notion of a source function. Let $\text{enabled}(q, Z)$ denote the set of actions $b \in \Sigma$ that are enabled from the node (q, Z) , i.e. such that there exists an edge $(q, Z) \xrightarrow{b} (q', Z')$ for some (q', Z') .

► **Definition 25.** *A source function for a timed network \mathcal{N} is a function $\text{src} : Q \times \mathcal{P}(\Sigma) \rightarrow \mathcal{P}(\Sigma)$. An action b is source enabled in (q, Z) if $b \in \text{src}(q, \text{enabled}(q, Z))$. A source path is a path taking only source enabled actions. A source function is trace faithful if for every node (q, Z) of $\text{LZG}(\mathcal{N})$, and an infinite path u from (q, Z) in $\text{LZG}(\mathcal{N})$ there is a source path $w \sim u$ from (q, Z) in $\text{LZG}(\mathcal{N})$.*

Observe that this definition of a source function allows to store some information in the state (like which process moved just before, etc.). Indeed such information is important for certain partial-order reduction approaches.

5.2 Local Zone Graph with Abstraction and Partial-Order

We are now in a position to define a local zone graph for the Büchi non-emptiness problem. This zone graph will use $\mathfrak{a}_{\approx LU}^D$ for finiteness and an arbitrary source function for partial-order reduction.

► **Definition 26** ($\text{eLZG}_{LU}^{D,src}(\mathcal{N})$). *Let \mathcal{N} be a D -spread network conforming to a given LU -bounds. Let $\text{src} : Q \times \mathcal{P}(\Sigma) \rightarrow \mathcal{P}(\Sigma)$ be a trace faithful source function. The graph $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$ is a subset of nodes and edges of $\text{LZG}(\mathcal{N})$ together with some new edges called equality edges. Each node is labeled either covered or uncovered. The graph must satisfy the following conditions:*

- *The initial node of $\text{LZG}(\mathcal{N})$ belongs to the graph.*
- *For every uncovered node (q, Z) of $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$ and for every $b \in \text{src}(q, \text{enabled}(Z))$ the transition $(q, Z) \xrightarrow{b} (q', Z')$ present in $\text{LZG}(\mathcal{N})$ should be in $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$.*
- *For every covered node (q, Z') there exists an uncovered node (q, Z) with $\mathfrak{a}_{\approx LU}^D(Z) = \mathfrak{a}_{\approx LU}^D(Z')$; moreover there is an explicit equality edge $(q, Z) \rightarrow_e (q, Z')$ in $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$.*
- *Every node of the graph is reachable from the initial node by a path of \Rightarrow edges.*

We write $\overset{b}{\rightsquigarrow}$ to mean a, possibly empty, sequence of equality edges followed by \xrightarrow{b} edge. Similarly $\overset{\sigma}{\rightsquigarrow}$ stands for a sequence of $\overset{\sigma}{\Rightarrow}$ edges possibly with equality edges in between.

Let M be defined as $M(x) = \max(L(x), U(x))$ for every process clock x . The next lemma gives a useful property of the $\mathfrak{a}_{\approx LU}^D$ abstraction which entails that the above local zone graph is a finite object.

► **Lemma 27.** *For every zone Z , the abstraction $\mathfrak{a}_{\approx LU}^D(Z)$ is a union of (M, D) -regions. The graph $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$ is finite for D -spread networks \mathcal{N} .*

Our goal is to show soundness and completeness of $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$ for Büchi non-emptiness. We have already seen in Proposition 21 that $\text{LZG}(\mathcal{N})$ is sound and complete. Therefore we will now relate paths in $\text{LZG}(\mathcal{N})$ with paths in $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$.

5.2.1 Soundness

For soundness, we want to show that every infinite path in $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$ corresponds to an infinite path in $\text{LZG}(\mathcal{N})$. The main difficulty in the argument comes from equality edges. Consider two reachable nodes (q_1, Z_1) and (q_1, Z'_1) such that $\mathbf{a}_{\approx_{LU}^D}^D(Z_1) = \mathbf{a}_{\approx_{LU}^D}^D(Z'_1)$. Let us say there is an equality edge $(q_1, Z_1) \rightarrow_e (q_1, Z'_1)$. Hence, $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$ does not contain paths from (q_1, Z_1) . We must thus show that for every path from (q_1, Z_1) in $\text{LZG}(\mathcal{N})$ there is a similar path from (q_1, Z'_1) . One may guess that as \approx_{LU}^* is a simulation, for every $(q_1, Z_1) \xRightarrow{b} (q_2, Z_2)$, we have $(q_1, Z'_1) \xRightarrow{b} (q_2, Z'_2)$ with $\mathbf{a}_{\approx_{LU}^D}^D(Z_1) = \mathbf{a}_{\approx_{LU}^D}^D(Z'_2)$. But this is not true. Notice that the $\mathbf{a}_{\approx_{LU}^D}^D$ operator restricts to D -spread valuations. So it can say nothing about the other valuations of Z_1 and Z'_1 and these non D -spread valuations may lead to D -spread valuations in Z_2 and Z'_2 . We have no control on such valuations just by using the fact that \approx_{LU}^* is a simulation. Nevertheless, we are able to show soundness of $\text{eLZG}_{LU}^D(\mathcal{N})$, albeit with a more involved reasoning that additionally uses the fact that \mathcal{N} is a D -spread system and the network \mathcal{N} is deterministic.

► **Lemma 28.** *Let \mathcal{N} be a deterministic D -spread network conforming to LU -bounds. Let (q_1, Z_1) be a node reachable from (q_0, Z_0) , namely $(q_0, Z_0) \xRightarrow{\sigma_1} (q_1, Z_1)$ for some $\sigma_1 \in \Sigma^*$. Let (q_1, Z'_1) be a reachable node of $\text{LZG}(\mathcal{N})$ that satisfies $\mathbf{a}_{\approx_{LU}^D}^D(Z_1) = \mathbf{a}_{\approx_{LU}^D}^D(Z'_1)$.*

For every finite or infinite sequence of transitions σ_2 : if $(q_1, Z_1) \xRightarrow{\sigma_2}$ in $\text{LZG}(\mathcal{N})$, then $(q_1, Z'_1) \xRightarrow{\sigma_2}$ in $\text{LZG}(\mathcal{N})$. Moreover, if σ_2 is finite then $\text{enabled}(q_2, Z_2) = \text{enabled}(q_2, Z'_2)$, where $(q_1, Z_1) \xRightarrow{\sigma_2} (q_2, Z_2)$ and $(q_1, Z'_1) \xRightarrow{\sigma_2} (q_2, Z'_2)$.

Proof. Suppose σ_2 is finite. Consider the sequence $(q_0, Z_0) \xRightarrow{\sigma_1} (q_1, Z_1) \xRightarrow{\sigma_2} (q_2, Z_2)$. By post-property, there exists a local run $(q_0, v_0) \xrightarrow{\sigma_1} (q_1, v_1) \xrightarrow{\sigma_2} (q_2, v_2)$ with $v_0 \in Z_0, v_1 \in Z_1$ and $v_2 \in Z_2$. As \mathcal{N} is D -spread, we can assume this run to be D -spread. Thus, $v_1 \in \text{spread}_D(Z_1)$.

As $\mathbf{a}_{\approx_{LU}^D}^D(Z_1) = \mathbf{a}_{\approx_{LU}^D}^D(Z'_1)$, there exists $v'_1 \in \text{spread}_D(Z'_1)$ such that $v_1 \approx_{LU}^* v'_1$. Hence there exists a run $(q_1, v'_1) \xrightarrow{\sigma_2} (q_2, v'_2)$. By pre-property, there exists a sequence of symbolic transitions $(q_1, Z'_1) \xRightarrow{\sigma_2} (q_2, Z'_2)$.

If σ_2 is infinite, then the above argument says that for every finite prefix σ_3 of σ_2 there is a sequence $(q_2, Z'_2) \xRightarrow{\sigma_3}$. Since \mathcal{N} is deterministic, the local zone graph $\text{LZG}(\mathcal{N})$ is deterministic. Hence we get the presence of the infinite path σ_2 from (q_2, Z'_2) , that is, $(q_2, Z'_2) \xRightarrow{\sigma_2}$.

For the last statement consider $(q_1, Z_1) \xRightarrow{\sigma_2} (q_2, Z_2)$ and $(q_1, Z'_1) \xRightarrow{\sigma_2} (q_2, Z'_2)$. Say b is enabled from (q_2, Z_2) . Using the first statement of the lemma $\sigma_2 b$ is possible from (q_1, Z'_1) , thus b is possible from (q_2, Z'_2) , as the transition system is deterministic. The case of b from (q_2, Z'_2) is the same by exchanging the roles of Z_2 and Z'_2 . ◀

► **Lemma 29.** *Let \mathcal{N} be a deterministic D -spread network that conforms to a bounds LU -bounds.*

Let $(q_0, Z_0) \xrightarrow{\sigma_p} (q, Z) \xrightarrow{\sigma_c} (q, Z)$ be a path in $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$ which could potentially contain equality edges. Then, there exists an infinite D -spread local run over the sequence $\sigma_p(\sigma_c)^\omega$.

5.2.2 Completeness

We now move on to showing that the graph that is computed is complete for Büchi non-emptiness. Recall that covered nodes and successors via actions that are outside the src are not explored in $\text{eLZG}_{LU}^{D,src}$. We will now make use of Lemma 28 to show that source paths in $\text{LZG}(\mathcal{N})$ are preserved in $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$. Later, for the final result, we will use the fact that src is trace faithful.

► **Lemma 30.** *Let \mathcal{N} be a D -spread network that conforms to a bounds function LU . Let $(q_0, Z_0) \xrightarrow{b_0} (q_1, Z_1) \xrightarrow{b_1} \dots$ be an infinite source path in $\text{LZG}(\mathcal{N})$. Then there is an infinite path $(q_0, Z_0) \xrightarrow{b_0} (q_1, Z_1) \xrightarrow{b_1} \dots$ in $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$.*

► **Theorem 31.** *Let \mathcal{N} be a deterministic D -spread network that conforms to LU -bounds and let F be a set of accepting actions. Then, there is an infinite global run visiting F infinitely often, iff there is a reachable cycle in $\text{eLZG}_{LU}^D(\mathcal{N})$ containing an edge over an action in F .*

Proof. By Proposition 9, there is an infinite global run visiting F infinitely often iff there is an infinite local run visiting F infinitely often. Since \mathcal{N} is D -spread, there is an infinite D -spread run with the same sequence of actions. Therefore it remains to prove that there is a D -spread local run visiting F infinitely often iff there is a reachable cycle in $\text{eLZG}_{LU}^D(\mathcal{N})$ with an action in F .

Suppose there is such a local run. Proposition 21 says that there is an infinite Büchi path in $\text{LZG}(\mathcal{N})$ from (q_0, Z_0) . Since the source function is assumed to be trace faithful, there is a source path that visits F infinitely often. Lemma 30, gives us a Büchi source path in $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$. The $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$ graph is finite (Lemma 27). Hence the infinite path leads to a cycle. As the infinite path contains F infinitely often, the cycle contains an action in F .

For the other direction, suppose there is a reachable cycle containing F in $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$. Lemma 29 gives a D -spread local run with the same sequence of actions and control states. Hence, this local run visits F infinitely often. ◀

6 Conclusions

We have developed a setting allowing to use partial-order methods for solving the Büchi non-emptiness problem for timed systems. Partial-order methods exploit commutation of independent actions. This is why we use local-time semantics for networks of timed automata. For a given network \mathcal{N} we define a finite local-zone graph $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$ such that there is a Büchi run in \mathcal{N} if there is a Büchi path in $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$. Moreover, if we have a partial order method that works on the, potentially infinite, local-zone graph $\text{LZG}(\mathcal{N})$, this method can be used for exploring $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$. We find this a satisfying formulation since in $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$ independent actions do not necessarily commute, due to equality edges.

We did not present here a concrete partial-order method that can be used in our setting. In principle, any ample/persistent/stubborn set method can be used to calculate what we call here source sets. These methods become quite complicated when dealing with infinitary conditions, and these complications limit the efficiency of partial-order reductions. As a first step, it would be reasonable to assume some structural properties, like may-termination [31], but we do not have a satisfying solution at this point.

We did not address the question of Zeno runs. Often one is not just interested in existence of a Büchi run but also wants it to be non-Zeno, that is, a run where time diverges. While there is no consensus on what kind of infinite runs can be considered realistic, it is rather clear that Zeno runs are not realistic. It is always possible to convert a network to a strongly non-Zeno network [29] and encode the non-Zeno requirement in a Büchi condition. Sometimes this construction can produce a blow-up than can be alleviated with more complicated approach [19]. It remains to be seen if this construction can be adapted to the local-time semantics.

References

- 1 Parosh Aziz Abdulla, Stavros Aronis, Bengt Jonsson, and Konstantinos Sagonas. Source sets: A foundation for optimal dynamic partial order reduction. *J. ACM*, 64(4):25:1–25:49, 2017. doi:10.1145/3073408.
- 2 Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.
- 3 Étienne André, Jaime Arias, Laure Petrucci, and Jaco van de Pol. Iterative bounded synthesis for efficient cycle detection in parametric timed automata. In Jan Friso Groote and Kim Guldstrand Larsen, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings, Part I*, volume 12651 of *Lecture Notes in Computer Science*, pages 311–329. Springer, 2021. doi:10.1007/978-3-030-72016-2_17.
- 4 Gerd Behrmann, Patricia Bouyer, Kim Guldstrand Larsen, and Radek Pelánek. Lower and upper bounds in zone-based abstractions of timed automata. *Int. J. Softw. Tools Technol. Transf.*, 8(3):204–215, 2006.
- 5 Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 485–500, 1998.
- 6 Frederik Meyer Bønneland, Peter Gjøøl Jensen, Kim Guldstrand Larsen, Marco Muñoz, and Jiri Srba. Stubborn set reduction for two-player reachability games. *Log. Methods Comput. Sci.*, 17(1), 2021. URL: <https://lmcs.episciences.org/7278>.
- 7 Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods Syst. Des.*, 24(3):281–320, 2004.
- 8 Damien Busatto-Gaston, Benjamin Monmege, Pierre-Alain Reynier, and Ocan Sankur. Robust controller synthesis in timed büchi automata: A symbolic approach. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 572–590. Springer, 2019. doi:10.1007/978-3-030-25540-4_33.
- 9 Krishnendu Chatterjee, Andreas Pavlogiannis, and Viktor Toman. Value-centric dynamic partial order reduction. *Proc. ACM Program. Lang.*, 3(OOPSLA):124:1–124:29, 2019. doi:10.1145/3360550.
- 10 Dennis Dams, Rob Gerth, Bart Knaack, and Ruurd Kuiper. Partial-order reduction techniques for real-time model checking. *Formal Aspects Comput.*, 10(5-6):469–482, 1998. doi:10.1007/s001650050028.
- 11 Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In *TACAS*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 1998.
- 12 Laurent Fribourg. A closed-form evaluation for extended timed automata. Technical report, CNRS and École Normale Supérieure de Cachan, 1998.
- 13 Patrice Godefroid and Pierre Wolper. A partial approach to model checking. *Inf. Comput.*, 110(2):305–326, 1994. doi:10.1006/inco.1994.1035.
- 14 R. Govind. *Partial-order reduction for timed systems*. PhD thesis, Université de Bordeaux, France and Chennai Mathematical Institute, India (cotutelle), 2021.
- 15 R. Govind, Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Abstractions for the Local-time Semantics of Timed Automata: a Foundation for Partial-order Methods. To appear at 37th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2022. URL: <https://hal.archives-ouvertes.fr/hal-03644039>.
- 16 R. Govind, Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Revisiting local time semantics for networks of timed automata. In *CONCUR*, volume 140 of *LIPICs*, pages 16:1–16:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

- 17 Henri Hansen, Shang-Wei Lin, Yang Liu, Truong Khanh Nguyen, and Jun Sun. Diamonds are a girl's best friend: Partial order reduction for timed automata with abstractions. In *CAV*, volume 8559 of *Lecture Notes in Computer Science*, pages 391–406. Springer, 2014.
- 18 Frédéric Herbreteau, B. Srivathsan, Thanh-Tung Tran, and Igor Walukiewicz. Why liveness for timed automata is hard, and what we can do about it. *ACM Trans. Comput. Log.*, 21(3):17:1–17:28, 2020.
- 19 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Efficient emptiness check for timed büchi automata. *Formal Methods Syst. Des.*, 40(2):122–146, 2012.
- 20 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Lazy abstractions for timed automata. In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 990–1005. Springer, 2013.
- 21 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. *Inf. Comput.*, 251:67–90, 2016. doi:10.1016/j.ic.2016.07.004.
- 22 Michalis Kokologiannakis, Iason Marmanis, Vladimir Gladstein, and Viktor Vafeiadis. Truly stateless, optimal dynamic partial order reduction. *Proc. ACM Program. Lang.*, 6(POPL), January 2022. doi:10.1145/3498711.
- 23 Alfons Laarman, Mads Chr. Olesen, Andreas Engelbrecht Dalsgaard, Kim Guldstrand Larsen, and Jaco van de Pol. Multi-core emptiness checking of timed büchi automata using inclusion abstraction. In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 968–983. Springer, 2013.
- 24 Kim G. Larsen, Marius Mikucionis, Marco Muñoz, and Jiri Srba. Urgent partial order reduction for extended timed automata. In Dang Van Hung and Oleg Sokolsky, editors, *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings*, volume 12302 of *Lecture Notes in Computer Science*, pages 179–195. Springer, 2020. doi:10.1007/978-3-030-59152-6_10.
- 25 Guangyuan Li. Checking timed büchi automata emptiness using lu-abstractions. In *FORMATS*, volume 5813 of *Lecture Notes in Computer Science*, pages 228–242. Springer, 2009.
- 26 Jesper B. Møller, Jakob Lichtenberg, Henrik Reif Andersen, and Henrik Hulgaard. Fully symbolic model checking of timed systems using difference decision diagrams. *Electron. Notes Theor. Comput. Sci.*, 23(2):88–107, 1999. doi:10.1016/S1571-0661(04)80671-6.
- 27 Doron A. Peled. All from one, one for all: on model checking using representatives. In Costas Courcoubetis, editor, *Computer Aided Verification, 5th International Conference, CAV '93, Elounda, Greece, June 28 - July 1, 1993, Proceedings*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 1993. doi:10.1007/3-540-56922-7_34.
- 28 Stavros Tripakis. Checking timed büchi automata emptiness on simulation graphs. *ACM Trans. Comput. Log.*, 10(3):15:1–15:19, 2009.
- 29 Stavros Tripakis, Sergio Yovine, and Ahmed Bouajjani. Checking timed Büchi automata emptiness efficiently. *Form. Methods Syst. Des.*, 26(3):267–292, 2005.
- 30 Antti Valmari. Stubborn sets for reduced state space generation. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1990 [10th International Conference on Applications and Theory of Petri Nets, Bonn, Germany, June 1989, Proceedings]*, volume 483 of *Lecture Notes in Computer Science*, pages 491–515. Springer, 1989. doi:10.1007/3-540-53863-1_36.
- 31 Antti Valmari. Stop it, and be stubborn! *ACM Trans. Embed. Comput. Syst.*, 16(2):46:1–46:26, 2017. doi:10.1145/3012279.
- 32 Antti Valmari and Henri Hansen. Stubborn set intuition explained. *Trans. Petri Nets Other Model. Concurr.*, 12:140–165, 2017. doi:10.1007/978-3-662-55862-1_7.
- 33 Naling Zhang, Markus Kusano, and Chao Wang. Dynamic partial order reduction for relaxed memory models. In David Grove and Stephen M. Blackburn, editors, *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, pages 250–259. ACM, 2015. doi:10.1145/2737924.2737956.

A Appendix for Section 3

► **Lemma 8.** *For every T' a prefix of T_σ , every linearization f of T' gives a local run.*

Proof. We will suppose that T' is infinite. For finite T' the argument is essentially the same. Consider a linearization of a prefix of T_σ given by a 1-1 function $f : \mathbb{N} \rightarrow \mathbb{N}$. This means that $\text{rng}(f)$ is \preceq_σ -downwards closed: if $j \in \text{rng}(f)$ and $i \preceq_\sigma j$ then $i \in \text{rng}(f)$. We construct a local run:

$$(\bar{q}_{f(0)}, \bar{v}_{f(0)}) \xrightarrow{\Delta_{f(0), b_{f(0)}}} (\bar{q}_{f(1)}, \bar{v}_{f(1)}) \xrightarrow{\Delta_{f(1), b_{f(1)}}} \dots$$

In order to state the invariant we define a function $\text{flast}(i, p)$ giving the last action before i with an action of process p : $\text{flast}(i, p) = f(l)$ where l the largest number strictly smaller than i with $b_{f(l)}$ an action of p ; we keep $\text{flast}(i, p)$ undefined if there is no such l . Recall that a state $q = (q^1, \dots, q^k)$ is a tuple of states of component processes. We use q^p for the p -th component of q . The invariants are:

- for every process p , $\bar{q}_{f(i)}^p = q_{\text{flast}(i, p)+1}^p$ or $\bar{q}_{f(i)}^p = q_0^p$ if $\text{flast}(i, p)$ undefined.
- for every process p , $\bar{v}_{f(i)}(t_p) = \theta(f(i))$ if $p \in \text{dom}(b_{f(i)})$; otherwise $\bar{v}_{f(i)}(t_p) = \theta(\text{flast}(i, p))$ or $\bar{v}_{f(i)}(t_p) = v_0(t_p)$ if $\text{flast}(i, p)$ is not defined.
- for every $x \neq t_p$ a clock of process p : if $\text{flast}(i, p)$ is defined then $\bar{v}_{f(i)}(x) = v(x)$ where $v(x) = v_{\text{flast}(i, p)}[R]$ and R are resets of $b_{\text{flast}(i, p)}$; if $\text{flast}(i, p)$ not defined then $\bar{v}_{f(i)}(x) = v_0(x)$.

We set $\bar{q}_{f(0)} = q_{f(0)}$ and $\bar{v}_{f(0)}(t_p) = \theta(f(0))$ for $p \in \text{dom}(b_{f(0)})$ and $\bar{v}_{f(0)}(t_p) = v_0(t_p)$ otherwise. Finally, we set $\bar{v}_{f(0)} = v_0(x)$ for all other clocks. This satisfies the invariant as $b_{f(0)}$ is \preceq -minimal action.

Suppose that we have constructed a run up to $(\bar{q}_{f(i)}, \bar{v}_{f(i)})$. The invariants guarantee that for every process $p \in \text{dom}(b_{f(i)})$ we have:

- $\bar{q}_{f(i)}^p = q_{f(i)}^p$,
- $\bar{v}_{f(i)}(t_p) = v_{f(i)}(t_p)$ and $\bar{v}_{f(i)}(x) = v_{f(i)}(x)$ for every clock x of p .

Since $b_{f(i)}$ is enabled from $(q_{f(i)}, v_{f(i)})$ this shows that it is also enabled from $(\bar{q}_{f(i)}, \bar{v}_{f(i)})$.

Consider $(\bar{q}_{f(i)}, \bar{v}_{f(i)}) \xrightarrow{b_{f(i)}} (q_b, v_b)$. We need to show that $q_b = \bar{q}_{f(i+1)}$ and that there is Δ giving $v_b + \Delta = \bar{v}_{f(i+1)}$.

We start with q_b . Take a process $p \in \text{dom}(b_{f(i)})$ then $\text{flast}(i+1, p) = f(i)$ and we get $q_b^p = q_{\text{flast}(i+1, p)+1}^p$. For $p \notin \text{dom}(b_{f(i)})$, we have $q_b^p = \bar{q}_{f(i)}^p = q_{\text{flast}(i, p)+1}^p = q_{\text{flast}(i+1, p)+1}^p$. The last equality is because $\text{flast}(i, p) = \text{flast}(i+1, p)$ when $p \notin \text{dom}(b_{f(i)})$.

Now we look at reference clocks t_p . If $p \in \text{dom}(b_{f(i)}) \setminus \text{dom}(b_{f(i+1)})$ then we have $v_b(t_p) = \theta(f(i)) = \theta(\text{flast}(i, p))$ as required. If $p \notin \text{dom}(b_{f(i)}) \cup \text{dom}(b_{f(i+1)})$ then $v_b(t_p) = \theta(\text{flast}(i, p)) = \theta(\text{flast}(i+1, p))$. If $p \in \text{dom}(b_{f(i+1)})$ then $v_b(t_p) = \theta(\text{flast}(i+1, p)) \leq \theta(f(i+1))$ so we take $\delta_p = \theta(f(i+1)) - v_b(t_p)$ to reestablish the invariant.

Finally, to check the third invariant take a clock $x \neq t_p$ of a process p . Recall that $v_b = v_{f(i)}[R_b]$ where R_b are resets of action $b_{f(i)}$. If $p \in \text{dom}(b_{f(i)})$ then v in the invariant is $v_{f(i)}[R]$ because $\text{flast}(i+1, p) = f(i)$. This is as required. If $p \notin \text{dom}(b_{f(i)})$ then $\text{flast}(i+1, p) = \text{flast}(i, p)$ and $v_b(x) = \bar{v}_{f(i)}$, so we are done in this case too.

Hence, we have prolonged the run to $(\bar{q}_{f(i+1)}, \bar{v}_{f(i+1)})$ and all invariants are satisfied. By induction we obtain the desired local run corresponding to a linearization f . ◀

► **Proposition 9.** *Consider a network of timed automata \mathcal{N} . There is a Büchi local run of \mathcal{N} iff there is a Büchi global run of \mathcal{N} .*

Proof. Since a global run is also a local run, one direction is easy.

For the other direction, let us take a local run with infinitely many occurrences of actions from F , and construct a global run. Recall that $\theta(i)$ is the time of the execution of action b_i , namely $v'_i(t_p)$ for $p \in \text{dom}(b_i)$. Consider an order $i < j$ when $\theta(i) < \theta(j)$ or $\theta(i) = \theta(j)$ and $i < j$.

This order is a linear order \leq_σ and moreover if $i < j$ then it is not the case that $j \leq i$ (Lemma 7). Yet the linear order $<$ may not have type ω meaning that it can be a transfinite sequence. We find a prefix T of T_σ such that $<$ is a linearization of T of type ω and T has infinitely many occurrences of actions from F .

Before doing this let us see how this gives us a desired global run. The $<$ -linearization of some prefix T of T_σ gives us a local run (Lemma 8):

$$\sigma = (q_0, v_0) \xrightarrow{\Delta_0} (q_0, v'_0) \xrightarrow{b_0} (q_1, v_1) \xrightarrow{\Delta_1} \dots$$

with $\theta(0) \leq \theta(1) \leq \dots$. We define $\bar{v}'_i(t_p) = \theta(i)$ for all process p and $\bar{v}'_i(x) = v_i(x)$ for all other clocks x . Clearly all \bar{v}'_i are synchronized valuations. We claim that

$$\sigma = (q_0, \bar{v}_0) \xrightarrow{\delta_0} (q_0, \bar{v}'_0) \xrightarrow{b_0} (q_1, \bar{v}_1) \xrightarrow{\delta_1} (q_1, \bar{v}'_1) \xrightarrow{b_1} \dots$$

is a global run (where $\delta_i = \theta(i) - \theta(i-1)$, and \bar{v}_{i+1} is determined by \bar{v}'_i and resets of b_i). For this we verify that every transition $(q_i, \bar{v}'_i) \xrightarrow{b_i} (q_{i+1}, \bar{v}_{i+1}) \xrightarrow{\delta_{i+1}} (q_{i+1}, \bar{v}'_{i+1})$ exists. We know $(q_i, v'_i) \xrightarrow{b_i} (q_{i+1}, v_{i+1})$. We also have by assumption that $\bar{v}_i(t_p) = \theta(i) = v'_i(t_p)$ for all $p \in \text{dom}(b_i)$, as well as $\bar{v}'_i(x) = v'_i(x)$ for all clocks x that are not reference clocks. Hence $(q_i, \bar{v}'_i) \xrightarrow{b_i} (q_{i+1}, v_b)$ exists. We have $v_b(x) = v_{i+1}(x)$ for all clocks that are non-reference clocks. Additionally $v_b(t_p) = v'_i(t_p) = \theta(i)$ for every process p . Hence, if we take $\delta_{i+1} = \theta(i+1) - \theta(i)$ we get $\bar{v}'_{i+1} = v_b + \delta_{i+1}$ as required. Repeating this reasoning we construct a desired run.

We come back to the problem of finding a desired linearization. If $<$ gives a linearization of T_σ of type ω then we are done. Otherwise, consider the set $I = \{i : i \text{ has finitely many } < \text{-smaller elements}\}$. We have that $<$ is an order of type ω on I . Moreover I is a \leq_σ -downward closed as it is impossible to have $j \leq_\sigma i$ and $i < j$ at the same time. If I contains infinitely many occurrences of actions from F then I defines a prefix we are looking for. Otherwise $\mathbb{N} \setminus I$ is infinite and there are infinitely many actions from F in $\mathbb{N} \setminus I$. Consider the set of processes P_ω such that there is an action $b \in \mathbb{N} \setminus I$ with p in its domain. For every $p \in P$ find the $<$ -smallest $j_p \in \mathbb{N} \setminus I$ such that $p \in \text{dom}(b_{j_p})$. We claim that there are finitely many $i \in I$ with $i \leq_\sigma j_p$. Indeed $i \leq_\sigma j_p$ implies $i \leq j$ in the standard order on natural numbers. Let I_0 contain all such i for all $p \in P$. We claim that for $i \in I \setminus I_0$ we have $i \not\leq_\sigma j$ for every $j \in \mathbb{N} \setminus I$. Hence $I_0 \cup (\mathbb{N} \setminus I)$ is a prefix of T_σ . Moreover, it contains infinitely many occurrences of actions from F since I contains only finitely many of those and \mathbb{N} has infinitely many. If $<$ -linearization of $I_0 \cup (\mathbb{N} \setminus I)$ has type ω then we are done. If not then we repeat the argument. Observe that this time we have fewer processes such that there are infinitely many actions involving the process (none of the processes involved in actions from $I \setminus I_0$ are there). Thus the argument must terminate giving us the desired prefix. ◀

B Appendix for Section 4

► **Lemma 32.** For all $x \in \mathbb{R} \setminus \mathbb{Z}$, we have $\{-x\} = 1 - \{x\}$.

12:20 Checking Timed Büchi Automata Emptiness Using the Local-Time Semantics

Proof. We have $x = \lfloor x \rfloor + \{x\}$ and $-x = \lfloor -x \rfloor + \{-x\}$. Therefore $-(\lfloor x \rfloor + \{x\}) = \lfloor -x \rfloor + \{-x\}$. Secondly, $\lfloor -x \rfloor = -\lfloor x \rfloor - 1$ for all $x \in \mathbb{R} \setminus \mathbb{Z}$. Plugging this into the previous equation gives the required conclusion. ◀

► **Lemma 33.** For $x, y, z \in \mathbb{R}$ such that $z - x \in \mathbb{R} \setminus \mathbb{Z}$, we have $\{z - x\} \leq \{z - y\}$ iff $\lfloor x - y \rfloor = \lfloor x - z \rfloor + \lfloor z - y \rfloor + 1$.

Proof.

$$\begin{aligned} x - y &= x - z + z - y \\ &= \lfloor x - z \rfloor + \lfloor z - y \rfloor + \{x - z\} + \{z - y\} \\ &= \lfloor x - z \rfloor + \lfloor z - y \rfloor + 1 - \{z - x\} + \{z - y\} \end{aligned}$$

This gives the statement of the lemma. ◀

► **Lemma 34.** Let $x, y, z \in \mathbb{R}$, such that $\{x - z\} > 0$ and $\{y - z\} > 0$. Then, $\{x - z\} \leq \{y - z\}$ iff $\lfloor z - x \rfloor \geq \lfloor z - y \rfloor$.

Proof. Follows by using $\{x - z\} = 1 - \{z - x\}$ and $\{y - z\} = 1 - \{z - y\}$ (Lemma 32). ◀

► **Lemma 35.** Let $v \approx^* v'$. Then, for variables $x, y, z \in X \cup X^t$, we have $\{v(z - x)\} \leq \{v(z - y)\}$ iff $\{v'(z - x)\} \leq \{v'(z - y)\}$.

Proof. Follows from Lemma 33 and Definition 14. ◀

► **Lemma 16.** Let $v \approx^* v'$. For every local delay $v \xrightarrow{\delta}_p u$, there exists a δ' such that $v' \xrightarrow{\delta'}_p u'$ where $u \approx^* u'$.

Proof. We assume that $0 < \delta < 1$. If $\delta \geq 1$ then we can decompose it into its integral part and fractional part and repeat the reasoning.

We divide the variable differences into three sets:

$$\begin{aligned} C^+ &= \{t_p - z \mid z \in X \setminus \{t_p\}\} \\ C^- &= \{z - t_p \mid z \in X \setminus \{t_p\}\} \\ C^0 &= \{x - y \mid x, y \in X \setminus \{t_p\}\} \end{aligned}$$

A local delay of δ increases the value of differences in C^+ , decreases the ones in C^- and leaves the C^0 differences unaltered. Consider an element $\phi \in C^+$. Based on the relation between δ and $1 - \{v(\phi)\}$, its value either stays in the same integer interval, or moves to the next integer point, or to the next integer interval. A symmetric change happens in C^- . We now make this idea more precise.

$$\begin{aligned} u(t_p - z) &= v(t_p - z) + \delta \\ &= \lfloor v(t_p - z) \rfloor + \{v(t_p - z)\} + \delta \\ &= \lfloor v(t_p - z) \rfloor + 1 - \{v(z - t_p)\} + \delta \quad \text{when } v(t_p - z) \neq 0 \\ u(z - t_p) &= v(z - t_p) - \delta \\ &= \lfloor v(z - t_p) \rfloor + \{v(z - t_p)\} - \delta \end{aligned}$$

From the above calculations, we observe some properties:

- When $\{v(t_p - z)\} \neq 0$:

$$\lfloor u(t_p - z) \rfloor = \lfloor v(t_p - z) \rfloor + 1 \text{ iff } \delta \geq \{v(z - t_p)\} \quad (1)$$

$$\lfloor u(z - t_p) \rfloor = \lfloor v(z - t_p) \rfloor - 1 \text{ iff } \delta > \{v(z - t_p)\} \quad (2)$$

- When $\{v(t_p - z)\} = 0$, as $\delta < 1$ we have:

$$\lfloor u(t_p - z) \rfloor = \lfloor v(t_p - z) \rfloor \quad (3)$$

$$\lfloor u(z - t_p) \rfloor = \lfloor v(z - t_p) \rfloor - 1 \quad (4)$$

Note that the difference in the inequalities (\geq in (1) and $>$ in (2)) is expected, since for any $x \in \mathbb{R}$ we have $\lfloor -x \rfloor = -\lfloor x \rfloor$ if $\{x\} = 0$, and $\lfloor -x \rfloor = -\lfloor x \rfloor - 1$ otherwise. Among the ordering of fractional parts of differences in C^- for v , consider $(z_1 - t_p), (z_2 - t_p)$ that are consecutive in this ordering such that $\{v(z_1 - t_p)\} \leq \delta < \{v(z_2 - t_p)\}$. Replace $\{v(z_1 - t_p)\}$ with 0 if no such z_1 exists, and replace $\{v(z_2 - t_p)\}$ with 1 if no such z_2 exists.

We now propose a δ' as required. From Lemmas 34 and 35, we know that the fractional parts of differences in C^- are ordered in the same way in v and v' . We take any δ' with $\{v'(z_1 - t_p)\} \leq \delta' < \{v'(z_2 - t_p)\}$, such that in addition $\delta' = \{v'(z_1 - t_p)\}$ if $\delta = \{v(z_1 - t_p)\}$. Let $u' = v' + \delta'$. Since we started with $v \approx^* v'$, from (1) to (4) we get $u \approx^* u'$. ◀

The following lemma shows that \equiv_M^D equivalence is preserved by choosing appropriate local delays.

► **Lemma 36.** *Let v, v' be D -spread valuations such that $v \equiv_M^D v'$. For every local delay Δ such that $v + \Delta$ is D -spread, there exists a local delay Δ' such that $v' + \Delta'$ is D -spread and $v + \Delta \equiv_M^D v' + \Delta'$.*

Proof. Let $\Delta = \{\delta_p\}_{p \in Proc}$. We can break the local delay Δ into a sequence of local delays $\xrightarrow{\delta_{p_1}} \xrightarrow{\delta_{p_2}} \dots$ happening one process at a time. Therefore it is sufficient to prove the lemma for a local delay of one process, say δ_p at process p .

Consider the given valuations v, v' which satisfy $v \equiv_M^D v'$. By definition, we have $\text{Bounded}(v) = \text{Bounded}(v')$ and $v|_B \approx^* v'|_B$, where $B = \text{Bounded}(v)$. From Lemma 16, for every local delay δ_p , there exists a delay δ'_p such that $(v +_p \delta_p)|_B \approx^* (v' +_p \delta'_p)|_B$. Clocks outside B are unbounded both in $v +_p \delta_p$ and $v' +_p \delta'_p$. Finally, we are interested only in delays δ_p such that $v +_p \delta_p$ is D -spread. Since all the reference clocks are present in B , we have $\lfloor (v +_p \delta_p)(t_r - t_s) \rfloor = \lfloor (v' +_p \delta'_p)(t_r - t_s) \rfloor$. This shows that $v' +_p \delta'_p$ is D -spread. All these observations lead to $v +_p \delta_p \equiv_M^D v' +_p \delta'_p$. ◀

► **Lemma 17.** *Let v, v' be D -spread valuations such that $v \equiv_M^D v'$. Let \mathcal{N} be a network that conforms to M . For every action transition $(q, v) \xrightarrow{b} (q_1, v_1)$ we have $(q, v') \xrightarrow{b} (q_1, v'_1)$ such that $v_1 \equiv_M^D v'_1$.*

Proof. As $(q, v) \xrightarrow{b} (q_1, v_1)$, we have $v(t_p - t_q) = 0$ for all $p, q \in \text{dom}(b)$. Since $v \equiv_M^D v'$, we also have $v'(t_p - t_q) = 0$ for $p, q \in \text{dom}(b)$. Secondly, v satisfies the guard g present in the b -transition. As \mathcal{N} conforms to M , every constraint in g is of the form $x < c, x \leq c$ or $x > c, x \geq c$ with $0 \leq c \leq M(x)$. Hence, by definition of $v \equiv_M^D v'$, valuation v' satisfies g too. This shows that b is enabled at (q, v') . Finally, resetting R from v sets differences $t_p - x$ with $x \in X_p \cap R$ to 0. It does not change the values of differences between reference clocks. Hence both $[R]v$ and $[R]v'$ are D -spread. Moreover, $\text{Bounded}([R](v)) = \text{Bounded}(v) \cup R$, which equals $\text{Bounded}(v') \cup R$ and hence $\text{Bounded}([R](v'))$. We need to show that $[R](v)|_{B_1} \approx^* [R](v')|_{B_1}$ where $B_1 = \text{Bounded}([R]v)$. So, we need to show that $\lfloor [R](v)(x - y) \rfloor = \lfloor [R](v')(x - y) \rfloor$.

This is direct when both $x, y \in R$, or when both $x, y \notin R$. Suppose $x \in R, y \notin R$. We have $[R]v(x - y) = v(t_p - y)$ when $x \in X_p$. Since t_p and y are already in $\text{Bounded}(v)$, we have $[v(t_p - y)] = [v'(t_p - y)]$ and hence $[[R](v)(x - y)] = [[R](v')(x - y)]$. Symmetric reasoning works when $x \notin R, y \in R$. \blacktriangleleft

► **Theorem 20.** *Let \mathcal{N} be a network that conforms to bounds function M . Then:*

- *For every D -spread local run $(q_0, v_0) \xrightarrow{\Delta_0, b_0} (q_1, v_1) \cdots$, there exists a run $(q_0, [v_0]_M^D) \xrightarrow{b} (q_1, [v_1]_M^D) \cdots$ in the (M, D) -region graph.*
- *For every run $(q_0, [v_0]_M^D) \xrightarrow{b_0} (q_1, [v_1]_M^D) \cdots$ in the (M, D) -region graph, there exists a D -spread local run $(q_0, v'_0) \xrightarrow{\Delta_0, b_0} (q_1, v'_1) \cdots$ such that $v'_i \in [v_i]_M^D$ for all $i \geq 0$.*
- *The (M, D) -region graph is finite.*

Proof.

- Follows from definition of region graph.
- Given a transition $(q_i, [v_i]_M^D) \xrightarrow{b_i} (q_{i+1}, [v_{i+1}]_M^D)$, for every valuation $u_i \in [v_i]_M^D$, there is a transition $(q_i, u_i) \xrightarrow{\Delta_i, b_i} (q_{i+1}, u_{i+1})$ with $u_{i+1} \in [v_{i+1}]_M^D$. This holds due to Corollary 18 and is sufficient to extract a run starting from some arbitrary initial configuration.
- We claim that an (M, D) region is specified by the following information:
 - a subset $B \subseteq \bigcup_{p \in \text{Proc}} X_p$ of bounded clocks,
 - for every process clock $x \in X_p$ that is bounded, whether $t_p - x = c$ or $e - 1 < t_p - x < e$ for $c \in \{0, \dots, M(x)\}$ and $e \in \{1, \dots, M(x)\}$,
 - for every pair of reference clocks t_p, t_q , whether $t_p - t_q = c$ or $e - 1 < t_p - t_q < e$ for $c \in \{-D, \dots, D\}$ and $e \in \{-D + 1, \dots, D\}$
 - for a pair of bounded process clocks $x, y \in B$, whether $x - y = c$ or $e - 1 < x - y < e$ for $c \in \{-M(x) - D, \dots, M(y) + D\}$ and $e \in \{-M(x) - D + 1, \dots, M(y) + D\}$.

The claim gives a finite bound on the number of regions. It remains to prove the claim. Consider an (M, D) -region $[v]_M^D$. We have $\text{Bounded}(v) = \text{Bounded}(v')$ for every valuation $v' \in [v]_M^D$. Hence the set B in the first item above is given by $\text{Bounded}(v)$. The next three items follow from $v|_B \approx^* v'|_B$ and noticing the bounds on the differences: we have $0 \leq v(t_p - x) \leq M(x)$ for $x \in B \cap X_p$ by definition of bounded clocks; we have $-D \leq v(t_p - t_q) \leq D$ since v has spread D ; finally for $x, y \in B$, we have $v(x - y) \leq v(x - t_p) + v(t_p - t_q) + v(t_q - y)$ assuming $x \in X_p, y \in X_q$. Now, we use the inequalities: $-M(x) \leq v(x - t_p) \leq 0$, $-D \leq v(t_p - t_q) \leq D$ and $0 \leq v(t_q - y) \leq M(y)$ to get $-M(x) - D \leq v(x - y) \leq M(y) + D$. \blacktriangleleft

C Appendix for Section 5

► **Lemma 37.** *Let \mathcal{N} be a deterministic D -spread network conforming to LU -bounds. For every path of the form $(q_0, Z_0) \xrightarrow{\sigma_1} (q_1, Z'_1) \xrightarrow{e} (q_1, Z_1) \xrightarrow{\sigma_2} (q_2, Z_2)$ in $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$ there exists a path $(q_1, Z'_1) \xrightarrow{\sigma_2} (q_2, Z'_2)$ in $\text{LZG}(\mathcal{N})$.*

Proof. Since (q_1, Z_1) is a node of $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$, it is reachable from (q_0, Z_0) , and so is (q_1, Z'_1) . Since $\mathbf{a}_{\leq LU}^D(Z_1) = \mathbf{a}_{\leq LU}^D(Z'_1)$, Lemma 28 gives us a path $(q_1, Z'_1) \xrightarrow{\sigma_2} (q_2, Z'_2)$ in $\text{LZG}(\mathcal{N})$. \blacktriangleleft

► **Lemma 29.** *Let \mathcal{N} be a deterministic D -spread network that conforms to a bounds LU -bounds.*

Let $(q_0, Z_0) \xrightarrow{\sigma_p} (q, Z) \xrightarrow{\sigma_c} (q, Z)$ be a path in $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$ which could potentially contain equality edges. Then, there exists an infinite D -spread local run over the sequence $\sigma_p(\sigma_c)^\omega$.

Proof. Let M be defined as $M(x) = \max(L(x), U(x))$ for every process clock x . Let $k \in \mathbb{N}$ be larger than the number of (M, D) -regions. Consider the finite path in $\text{eLZG}_{LU}^D(\mathcal{N})$ obtained by the sequence $\sigma_p(\sigma_c)^k$. By repeated use of Lemma 37, there is a path $\sigma_p(\sigma_c)^k$ in $\text{LZG}(\mathcal{N})$. By post-property of $\text{LZG}(\mathcal{N})$, there is a local run $(q_0, v_0) \xrightarrow{\sigma_p} (q, v_1) \xrightarrow{\sigma_c} (q, v_2) \xrightarrow{\sigma_c} \dots \xrightarrow{\sigma_c} (q, v_{k+1})$. As \mathcal{N} is D -spread, this run can be assumed to be D -spread. Due to Theorem 20, there is a path $\sigma_p(\sigma_c)^k$ in the (M, D) -region graph: $(q_0, [v_0]_M^D) \xrightarrow{\sigma_p} (q, [v_1]_M^D) \xrightarrow{\sigma_c} \dots \xrightarrow{\sigma_c} (q, [v_{k+1}]_M^D)$. As k is larger than the number of regions, there exist i, j such that $[v_i]_M^D = [v_j]_M^D$. This gives a path $\sigma_p \sigma_c^{i-1} \sigma_c^{j-i} \sigma_c^{k+1-j}$ in the region graph where the part σ_c^{j-i} is a cycle, which can be iterated infinitely often. Hence there is a path $\sigma_p(\sigma_c)^\omega$ in the region graph. By Theorem 20, there is an infinite local run over the sequence $\sigma_p(\sigma_c)^\omega$, whose intermediate valuations are all D -spread. \blacktriangleleft

► Lemma 30. *Let \mathcal{N} be a D -spread network that conforms to a bounds function LU . Let $(q_0, Z_0) \xrightarrow{b_0} (q_1, Z_1) \xrightarrow{b_1} \dots$ be an infinite source path in $\text{LZG}(\mathcal{N})$. Then there is an infinite path $(q_0, Z_0) \xrightarrow{b_0} (q_1, Z_1) \xrightarrow{b_1} \dots$ in $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$.*

Proof. Let $w_i = b_i b_{i+1} \dots$. By induction on i we construct a path in $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$

$$(q_0, Z_0) \xrightarrow{b_0} (q_1, Z'_1) \dots \xrightarrow{b_i} (q_i, Z'_i)$$

such that in $\text{LZG}(\mathcal{N})$ there is a path

$$(q_i, Z'_i) \xrightarrow{b_{i+1}} (q_{i+1}, Z_{i+1}^i) \xrightarrow{b_{i+2}} (q_{i+2}, Z_{i+2}^i) \dots \quad \text{and } \text{enabled}(q_j, Z_j^i) = \text{enabled}(q_j, Z_j).$$

The second item implies that the later path is a source path in $\text{LZG}(\mathcal{N})$.

If (q_i, Z'_i) is not covered in $\text{eLZG}_{LU}^{D,src}$ then the induction step is direct.

If (q_i, Z'_i) is covered in $\text{eLZG}_{LU}^{D,src}$ then there exists $(q_i, Z'_i) \rightarrow_e (q_i, Z''_i)$ with (q_i, Z''_i) uncovered and $\alpha_{\preccurlyeq LU}^D(Z'_i) = \alpha_{\preccurlyeq LU}^D(Z''_i)$. As (q_i, Z''_i) is reachable from (q_0, Z_0) in $\text{LZG}(\mathcal{N})$ by the definition of $\text{eLZG}_{LU}^{D,src}$, we can use Lemma 28. The lemma gives us a path $(q_i, Z''_i) \xrightarrow{b_{i+1}} (q_{i+1}, Z_{i+1}^{i+1}) \xrightarrow{b_{i+2}} (q_{i+2}, Z_{i+2}^{i+1}) \dots$ such that $\text{enabled}(q_j, Z_j^i) = \text{enabled}(q_j, Z_j^{i+1})$. Thus we also have $\text{enabled}(q_j, Z_j) = \text{enabled}(q_j, Z_j^{i+1})$. We can prolong the finite prefix $(q_0, Z_0) \xrightarrow{b_0} (q_1, Z'_1) \dots \xrightarrow{b_i} (q_i, Z'_i)$ by $(q_i, Z'_i) \rightarrow_e (q_i, Z''_i) \xrightarrow{b_{i+1}} (q_{i+1}, Z_{i+1}^{i+1})$. \blacktriangleleft

The local zone graph $\text{eLZG}_{LU}^{D,src}(\mathcal{N})$ is finite. We will prove that for every zone Z , the abstraction $\alpha_{\preccurlyeq LU}^D(Z)$ is a union of (M, D) -regions. To prove this statement, we will need to reason about *canonical* representations of zones. Zones are typically represented using Difference-Bound-Matrices (DBMs) or distance graphs [21]. We will use the distance graph representation for our analysis. A constraint $x - y \leq c$ of the zone is represented as an edge $y \xrightarrow{\leq c} x$. An arithmetic over weights of (\leq, c) can be suitably defined (see [21], [15]) for more details. A canonical graph is one where the shortest path from y to x is given by the direct edge $y \rightarrow x$. We will write Z_{yx} to denote the weight of the $y \rightarrow x$ edge in the canonical distance graph representing Z .

For convenience of presentation, we define two sets of clocks for a given local valuation v :

$$L\text{-bounded}(v) := T \cup \bigcup_{p \in \text{Proc}} \{x \in X_p \mid v(t_p - x) \leq L_x\}$$

$$U\text{-bounded}(v) := T \cup \bigcup_{p \in \text{Proc}} \{x \in X_p \mid v(t_p - x) \leq U_x\}$$

Notice that the reference clocks T are present in both L -bounded(v) and U -bounded(v).

Define $\langle v \rangle^* := \{v' \mid v \preceq_{LU}^* v'\}$. We will now recall the distance graph representation of $\langle v \rangle^*$ and an important property of the intersection $\langle v \rangle^* \cap Z$ for some arbitrary zone Z .

► **Definition 38** (Distance graph H^v [15]). *Let $x, y \in X \cup X^t$ be two clocks, possibly reference clocks. Assume that $y \neq x$ and $y \in X_q \cup \{t_q\}$ for some process q . The weight of the edge $x \rightarrow y$ in the distance graph H^v is given by:*

$$\left\{ \begin{array}{ll} (\leq, v(y-x)) & \text{if } x \in U\text{-bounded}(v), \\ & y \in L\text{-bounded}(v) \\ (\leq, v(t_q-x)) + (<, -L_y) & \text{if } x \in U\text{-bounded}(v), \\ & y \notin L\text{-bounded}(v), L_y \neq -\infty \\ (\leq, v(t_q-x)) & \text{if } x \in U\text{-bounded}(v), \\ & y \notin L\text{-bounded}(v), L_y = -\infty \\ (<, \infty) & \text{otherwise} \end{array} \right.$$

► **Proposition 39.** [15] *The intersection $\langle v \rangle^* \cap Z$ is empty iff there are two variables $x, y \in X \cup T$ s.t. $x \in U$ -bounded(v), $L_y \neq -\infty$ when y is a process clock, and $H_{xy}^v + Z_{yx} < (\leq, 0)$.*

The above proposition gives a simple characterization for when the upward closure of a valuation v wrt to the \preceq_{LU}^* simulation does not intersect zone Z . Using this, we can show that when two valuations belong to the same (M, D) -region, then one of them satisfies this characterization iff the other does so. Here, we will make use of the fact that our atomic constraints involve integer constants, and hence all zones that appear in the local zone graph computation will only involve integer constants.

► **Lemma 40.** *Let L, U and M be bound functions such that for every process clock x , we have $M(x) \geq L(x)$ and $M(x) \geq U(x)$. For every zone Z , the set $\mathbf{a}_{\preceq_{LU}^*}^*(\text{spread}_D(Z))$ is a finite union of (M, D) -regions.*

Proof. We first remark that every valuation in $\mathbf{a}_{\preceq_{LU}^*}^*(\text{spread}_D(Z))$ is D -spread. Let v be a D -spread valuation. We have $v \in \mathbf{a}_{\preceq_{LU}^*}^*(\text{spread}_D(Z))$ iff $\langle v \rangle^* \cap Z$ is non-empty. Let $v \equiv_M^D v'$. We will show that $\langle v \rangle^* \cap Z$ is empty iff $\langle v' \rangle^* \cap Z$ is empty. This will prove the lemma. Let H^v and $H^{v'}$ be the canonical distance graphs representing $\langle v \rangle^*$ and $\langle v' \rangle^*$ respectively.

From Proposition 39, $\langle v \rangle^* \cap Z$ is empty iff there exist two variables x, y such that $x \in U$ -bounded(v), and $L_y \neq -\infty$ when y is a process clock, such that $H_{xy}^v + Z_{yx} < (\leq, 0)$. As $x \in U$ -bounded(v), we also have $x \in \text{Bounded}(v)$, as $M(x) = \max(L(x), U(x))$. Since $v \equiv_M^D v'$, we have $x \in U$ -bounded(v) iff $x \in U$ -bounded(v'). When $y \in \text{Bounded}(v)$, we have $\lfloor H_{xy}^v \rfloor = \lfloor H_{xy}^{v'} \rfloor$. Since Z_{yx} is of the form (\leq, c) with c an integer, we have $H_{xy}^v + Z_{yx} < (\leq, 0)$ iff $H_{xy}^{v'} + Z_{yx} < (\leq, 0)$. When $y \notin \text{Bounded}(v)$, then in particular, y is a process clock, $y \notin L$ -bounded(v) and we have $H_{xy}^v = (\leq, v(t_q - x)) + (<, -L_y)$ where q is the process containing y . But, t_q belongs to both $\text{Bounded}(v)$ and $\text{Bounded}(v')$. Hence $\lfloor v(t_q - x) \rfloor = \lfloor v'(t_q - x) \rfloor$ and the lemma follows for this case using the previous argument. ◀

Simulations for Event-Clock Automata

S. Akshay  



Department of CSE, Indian Institute of Technology Bombay, Mumbai, India

Paul Gastin  

Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, 91190, Gif-sur-Yvette, France
CNRS, ReLaX, IRL 2000, Siruseri, India

R. Govind  

Department of CSE, Indian Institute of Technology Bombay, Mumbai, India

B. Srivathsan  

Chennai Mathematical Institute, India
CNRS, ReLaX, IRL 2000, Siruseri, India

Abstract

Event-clock automata are a well-known subclass of timed automata which enjoy admirable theoretical properties, e.g., determinizability, and are practically useful to capture timed specifications. However, unlike for timed automata, there exist no implementations for event-clock automata. A main reason for this is the difficulty in adapting zone-based algorithms, critical in the timed automata setting, to the event-clock automata setting. This difficulty was studied in [19, 20], where the authors also proposed a solution using zone extrapolations.

In this paper, we propose an alternative zone-based algorithm, *using simulations* for finiteness, to solve the reachability problem for event-clock automata. Our algorithm exploits the \mathcal{G} -simulation framework, which is the coarsest known simulation relation for reachability, and has been recently used for advances in other extensions of timed automata.

2012 ACM Subject Classification Theory of computation \rightarrow Timed and hybrid models; Theory of computation \rightarrow Quantitative automata; Theory of computation \rightarrow Logic and verification

Keywords and phrases Event-clock automata, verification, zones, simulations, reachability

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.13

Related Version *Full Version*: <https://arxiv.org/abs/2207.02633>

Funding This work was supported by DST/CEFIPRA/INRIA Project EQuaVE.

S. Akshay: Supported in part by DST/SERB Matrix Grant MTR/2018/000744.

Paul Gastin: Partially supported by ANR project Ticktac (ANR-18-CE40-0015).

1 Introduction

Timed automata (TA) [4] are a well-established model for real-time systems and form the basis for employing model-checking techniques. The most popular property that has been considered in these systems is control state reachability. Reachability in timed automata is a well-studied problem and was shown to be decidable (and PSPACE-complete) using the so-called region construction [4]. This construction was primarily of theoretical interest, as the number of regions, which are collections of reachable configurations, explodes both in theory and in practice. On the other hand, timed automata have been implemented in several tools: UPPAAL [26, 6], KRONOS [10], PAT [29], RED [31], TChecker [21], Theta [30], LTS-Min [24], Symrob [28], MCTA [25], etc. Most of these tools have a common underlying algorithm which is an explicit enumeration of reachable configurations stored as *zones* [7]. Since the late 90s, a substantial effort has been invested in improving zone enumeration techniques, the common challenge being how to get a sound and complete enumeration while exploring as few zones as possible.



© S. Akshay, Paul Gastin, R. Govind, and B. Srivathsan;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 13; pp. 13:1–13:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The more general model checking problem of whether the system represented by TA \mathcal{A} satisfies the specification given by TA \mathcal{B} reduces to the language inclusion problem $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$. There are two challenges here: first, the inclusion problem is undecidable in its full generality, and second, having clocks, though excellent for timed implementations, are often less than ideal for modeling timed specifications. This has led to the introduction of event-clocks and the corresponding model of *event-clock automaton (ECA)* [5]. Event-clock automata make use of special clocks that track the time since the last occurrence of an event (history clocks) or the time until the next occurrence of an event (prophecy clocks). On one hand this makes writing timed specifications more natural. Indeed, the role of prophecy clocks is in the same spirit as future modalities in temporal logics. This has led to several extensions of temporal logics with event-clocks [15, 1, 27], which are often used as specification languages and can be converted into ECA. On the other hand, ECA can be determinized and hence complemented. Observe that model-checking event-clock specifications over TA models can be reduced to the reachability problem on the product of the TA with an ECA. This product contains usual clocks, history clocks and prophecy clocks. The usual clocks can be treated in the same way as history clocks for the zone analysis. Therefore, if we solve ECA reachability (with history and prophecy clocks) using zones, we can incorporate usual clocks into the procedure seamlessly. The bottomline is that the well-motivated problem of model-checking event-clock specifications over TA models can be reduced to an ECA reachability problem.

Thus, in this paper, we focus on the core problem of building efficient, zone-based algorithms for reachability in ECA. This problem turns out to be significantly different compared to zone based reachability algorithms in usual TA, precisely due to prophecy clocks. Our goal is to align the zone-based reachability algorithms for ECA with recent approaches for TA that have shown significant gains.

As mentioned earlier, the core of an efficient TA reachability algorithm is an enumeration of zones, where the central challenge is that naïve enumeration does not terminate. One approach to guarantee termination is to make use of an *extrapolation* operation on zones: each new zone that is enumerated is extrapolated to a bigger zone. Any freshly enumerated zone that is contained in an existing zone is discarded. More recently, a new *simulation* approach to zone enumeration has been designed, where enumerated zones are left unchanged. Instead, with each fresh zone it is checked whether the fresh zone is simulated by an already seen zone. If yes, the fresh zone is discarded. Otherwise, it is kept for further exploration. Different simulations have been considered: the *LU-simulation* [22] which is based on *LU*-bounds, or the *G-simulation* [18], which is based on a carefully-chosen set of constraints. Coarser simulations lead to fewer zones being enumerated. The *G-simulation* is currently the coarsest-known simulation that can be efficiently applied in the simulation approach. The simulation based approach offers several gains over the extrapolation approach: (1) since concrete zones are maintained, one could use dynamic simulation parameters and dynamic simulations, starting from a coarse simulation and refining whenever necessary [23], (2) the simulation approach has been extended to richer models like timed automata with diagonal constraints [17, 16], updatable timed automata [18], weighted timed automata [9] and pushdown timed automata [3]. In these richer models, extrapolation has either been shown to be impossible [8] or is unknown.

Surprisingly, for ECA, an arguably more basic and well-known model, it turns out that there are no existing simulation-based approaches. However, an extrapolation approach using maximal constants has been studied for ECA in [19, 20]. In this work, the authors start by showing that prophecy clocks exhibit fundamental differences as compared to usual clocks. To begin with, it was shown that there is no finite time-abstract bisimulation for ECA in

general. This is in stark contrast to TA where the region equivalence forms a finite time-abstract bisimulation. The correctness of extrapolation is strongly dependent on the region equivalence. Therefore, in order to get an algorithm, the authors define a weak semantics for ECA and a corresponding notion of weak regions which is a finite time-abstract bisimulation for the weak semantics and show that the weak semantics is sound for reachability. Building on this, they define an extrapolation operation for the zone enumeration.

Contributions. Given the advantages of using simulations with respect to extrapolations in the TA setting described above, we extend the \mathcal{G} -simulation approach to ECA. Here are the technical contributions leading to the result.

- We start with a slightly modified presentation of zones in ECA and provide a clean algebra for manipulating weights in the graph representation for such ECA-zones. This simplifies the reasoning and allows us to adapt many ideas for simulation developed in the TA setting directly to the ECA setting.
- The \mathcal{G} -simulation is parameterized by a set of constraints at each state of the automaton. We adapt the constraint computation and the definition of the simulation to the context of ECA, the main challenge being the handling of prophecy clocks.
- We give a simulation test between two zones that runs in time quadratic in the number of clocks. This is an extension of the similar test that exists for timed automata, but now it incorporates new conditions that arise due to prophecy clocks.
- Finally, we show that the reachability algorithm using the \mathcal{G} -simulation terminates for ECA: for every sequence Z_0, Z_1, \dots of zones that are *reachable* during a zone enumeration of an ECA, there exist $i < j$ such that Z_j is simulated by Z_i . This is a notable difference to the existing methods in TA, where finiteness is guaranteed for all zones, not only the reachable zones. In the ECA case, this is not true: we can construct an infinite sequence of zones which are incomparable with respect to the new \mathcal{G} -simulation. However, we show that finiteness does hold when restricting to reachable zones, and this is sufficient to prove termination of the zone enumeration algorithm. Our argument involves identifying some crucial invariants in reachable zones, specially, involving the prophecy clocks.

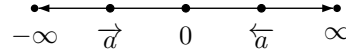
The fundamental differences in the behaviour of prophecy clocks as compared to usual clocks constitute the major challenge in developing efficient procedures for the analysis of ECAs. In our work, we have developed methods to incorporate prophecy clocks alongside the usual clocks. We prove a surprising property: in all *reachable* ECA-zones, the constraints involving *prophecy* clocks come from a finite set. A direct consequence of this observation is that the event zone graph of an ECA containing only prophecy clocks (known as Event-Predicting Automata EPA) is always finite. We wish to emphasize that, in this work, we are moving a step towards implementability, and at the same time towards more expressivity, since simulation approaches are amenable to extensions, e.g., with diagonal constraints.

Organization of the paper. Section 2 recalls ECA and describes a slightly modified presentation of the ECA semantics. Section 3 introduces event zones, event zone graph and the simulation based reachability framework. Section 4 introduces the new algebra for representing event zones and describes some operations needed to build the zone graph. Section 5 introduces the \mathcal{G} -simulation for event-clock automata and gives the simulation test. Section 6 proves finiteness of the simulation when restricted to reachable zones. All the missing proofs can be found in the full version of the paper [2].

2 Event Clock Automata and Valuations

Let X be a finite set of variables called *clocks*. Let $\Phi(X)$ denote a set of clock constraints generated by the following grammar: $\varphi ::= x \triangleleft c \mid c \triangleleft x \mid \varphi \wedge \varphi$ where $x \in X$, $c \in \overline{\mathbb{Z}} = \mathbb{Z} \cup \{-\infty, +\infty\}$ and $\triangleleft \in \{<, \leq\}$. The base constraints of the form $x \triangleleft c$ and $c \triangleleft x$ will be called *atomic constraints*. Constraints $x < -\infty$ and $+\infty < x$ are equivalent to *false* and constraints $-\infty \leq x$ and $x \leq +\infty$ are equivalent to *true*.

Given a finite alphabet Σ , we define a set $X_H = \{\overleftarrow{a} \mid a \in \Sigma\}$ of *history clocks* and a set $X_P = \{\overrightarrow{a} \mid a \in \Sigma\}$ of *prophecy clocks*. Together, history and prophecy clocks are called *event clocks*. In this paper, all clocks will be event clocks, thus we set $X = X_H \cup X_P$.



■ **Figure 1** Range of valuations of event clocks. A valuation maps history clocks to $\mathbb{R}_{\geq 0} \cup \{+\infty\}$ and prophecy clocks to $\mathbb{R}_{\leq 0} \cup \{-\infty\}$.

► **Definition 1 (Valuation).** *A valuation of event clocks is a function $v: X \mapsto \overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ which maps history clocks to $\mathbb{R}_{\geq 0} \cup \{+\infty\}$ and prophecy clocks to $\mathbb{R}_{\leq 0} \cup \{-\infty\}$. We say a history clock \overleftarrow{a} , for some $a \in \Sigma$ is undefined (resp. defined) when $v(\overleftarrow{a}) = +\infty$ (resp. $v(\overleftarrow{a}) < +\infty$) and a prophecy clock \overrightarrow{a} is undefined (resp. defined) when $v(\overrightarrow{a}) = -\infty$ (resp. $-\infty < v(\overrightarrow{a})$). We denote by $\mathbb{V}(X)$ or simply by \mathbb{V} the set of valuations over X .*

We remark that the history clock and the prophecy clock of an event a are symmetric notions. In the semantics that we introduce in this paper, history clock \overleftarrow{a} stores the amount of time elapsed after seeing the last a , measuring how far ahead in the future we are w.r.t. the last occurrence of a . Before we see an a for the first time, \overleftarrow{a} is set to $+\infty$. The prophecy clock \overrightarrow{a} stores the negative of the amount of time that needs to be elapsed before seeing the next a . In other words, $-\overrightarrow{a}$ tells us how far behind in the past we are w.r.t. the next occurrence of a . If no more a 's are going to be seen, then the prophecy clock of a is set to $-\infty$, i.e., $\overrightarrow{a} = -\infty$. See Figure 1 for a pictorial representation of valuations of event clocks.

Notice that for history (resp. prophecy) clocks, *useful* constraints use non-negative (resp. non-positive) constants. Also, $\overleftarrow{a} < 0$ and $0 < \overrightarrow{a}$ are equivalent to *false* whereas $0 \leq \overleftarrow{a}$, $\overleftarrow{a} \leq \infty$, $\overrightarrow{a} \leq 0$ and $-\infty \leq \overrightarrow{a}$ are equivalent to *true*. A constraint $c \triangleleft \overleftarrow{a}$ does not imply that the history clock \overleftarrow{a} is defined, whereas a constraint $\overleftarrow{a} \triangleleft c$ with $(\triangleleft, c) \neq (\leq, \infty)$ does. The same applies to prophecy clocks where a constraint $c \triangleleft \overrightarrow{a}$ with $(c, \triangleleft) \neq (-\infty, \leq)$ implies that \overrightarrow{a} is defined, whereas $\overrightarrow{a} \triangleleft c$ does not; in fact, $\overrightarrow{a} \leq -\infty$ states that \overrightarrow{a} is undefined.

► **Remark 2.** In the earlier works on ECA [5, 20], prophecy clocks assumed non-negative values and decreased along with time. This allowed to write guards on prophecy clocks with non-negative constants, e.g., $\overrightarrow{a} \leq 5$ means that the next a occurs in at most 5 time units. In our convention, this would be written as $-5 \leq \overrightarrow{a}$. Secondly, an undefined clock (history or prophecy) was assigned a special symbol \perp in earlier works. We have changed this to use $-\infty$ and $+\infty$ for undefined prophecy and history clocks respectively. We adopt these new conventions as they allow to treat both history clocks and prophecy clocks in a symmetric fashion, and a clean integration of undefined values when we describe zones and simulations.

We say that a valuation v satisfies a constraint φ , denoted as $v \models \varphi$, if φ evaluates to *true*, when each variable x in φ is replaced by its value $v(x)$. We write $[\overleftarrow{a}]v$ to denote the valuation v' obtained from v by resetting the history clock \overleftarrow{a} to 0, keeping the value of other clocks unchanged. We denote by $[\overrightarrow{a}]v$ the set of valuations v' obtained from v by setting

the prophecy clock \vec{a} non-deterministically to some value in $[-\infty, 0]$, keeping the value of other clocks unchanged. We denote by $v + \delta$ the valuation obtained by increasing the value of all clocks from the valuation v by $\delta \in \mathbb{R}_{\geq 0}$. Not every time elapse may be possible from a valuation, since prophecy clocks need to stay at most 0. For example, if there are two events a, b , then a valuation with $v(\vec{a}) = -3$ and $v(\vec{b}) = -2$ can elapse at most 2 time units.

► **Definition 3** (Event-clock automata [5]). *An event-clock automaton (ECA) \mathcal{A} is given by a tuple $(Q, \Sigma, X, T, q_0, F)$, where Q is a finite set of states, Σ is a finite alphabet of actions, X is the set of event clocks for Σ , $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of accepting states and $T \subseteq Q \times \Sigma \times \Phi(X) \times Q$ is a finite set of transitions.*

The semantics of an ECA $\mathcal{A} = (Q, \Sigma, X, T, q_0, F)$ is given by a transition system $S_{\mathcal{A}}$ whose states are configurations (q, v) of \mathcal{A} , where $q \in Q$ and v is a valuation. A configuration (q, v) is initial if $q = q_0$, $v(x) = \infty$ for all $x \in X_H$ and $-\infty \leq v(x) \leq 0$ for all $x \in X_P$. A configuration (q, v) is accepting if $q \in F$, and $v(x) = -\infty$ for all $x \in X_P$ and $0 \leq v(x) \leq \infty$ for all $x \in X_H$. Transitions of $S_{\mathcal{A}}$ are of two forms:

- **Delay transition:** $(q, v) \xrightarrow{\delta} (q, v + \delta)$, if $(v + \delta)(x) \leq 0$ for all $x \in X_P$.
- **Action transition:** $(q, v) \xrightarrow{t} (q', [\overleftarrow{a}]v')$ if $t = (q, a, g, q')$ is a transition in \mathcal{A} , $v(\vec{a}) = 0$, $v' \in [\vec{a}]v$ and $v' \models g$.

A transition with action a can be taken when the value of the prophecy clock \vec{a} is 0, then a new value in $[-\infty, 0]$ for \vec{a} is non-deterministically guessed so that the resulting valuation v' satisfies the guard g , and finally, the history clock \overleftarrow{a} is reset to 0.

An ECA is called an event recording automaton (ERA) if it only contains history clocks and event predicting automaton (EPA) if it only contains prophecy clocks. A run of an event-clock automaton is a finite sequence of transitions from an initial configuration of $S_{\mathcal{A}}$. A run is said to be *accepting* if its last configuration is accepting. We are interested in the *reachability problem* of an event clock automaton. Formally,

► **Definition 4** (Reachability problem for ECA). *The reachability problem for an event-clock automaton \mathcal{A} is to decide whether \mathcal{A} has an accepting run.*

Different solutions based on regions and zones have been proposed in [5, 19, 20]. For ERA, the standard region and zone based algorithms for timed automata work directly. However, for EPA (and ECA), this is not the case. In fact, [19] show that the standard region abstraction is not possible, as there exists no finite bisimulation due to the behavior of prophecy clocks. Also, the standard definition of zones used for timed automata is not sufficient to handle valuations with undefined clocks. The papers [19, 20] make use of special symbols \perp and $?$ for this purpose. In this work, we use a different formulation of zones by making use of $+\infty$ and $-\infty$. Instead of using $x = \perp$ (resp. $x \neq \perp$) to state that a clock is undefined (resp. defined) as in [19, 20], we write $+\infty \leq x$ or $x \leq -\infty$ or (resp. $x < +\infty$ or $-\infty < x$) depending on whether x is a history clock or a prophecy clock. This distinction between being undefined for history and prophecy clocks plays an important role.

3 Event zones and simulation based reachability

The most widely used approach for checking reachability in a timed automaton is based on reachability in a graph called the *zone graph* of a timed automaton [13]. Roughly, *zones* [7] are sets of valuations that can be represented efficiently using difference constraints between clocks. In this section, we introduce an analogous notion for event-clock automata. We consider *event zones*, which are sets of valuations of event-clock automata.

► **Definition 5** (Event zones). *An event zone is a set of valuations satisfying a conjunction of constraints of the form $c \triangleleft x$, $x \triangleleft c$ or $x - y \triangleleft c$, where $x, y \in X$ and $c \in \overline{\mathbb{Z}} = \mathbb{Z} \cup \{-\infty, +\infty\}$. Constraints of the form $x - y \triangleleft c$ are called diagonal constraints. To evaluate such constraints, we extend addition on real numbers with the convention that $(+\infty) + \alpha = +\infty$ for all $\alpha \in \overline{\mathbb{R}}$ and $(-\infty) + \beta = -\infty$, as long as $\beta \neq +\infty$. We simply write $v(x - y)$ for $v(x) - v(y)$.*

Let W be a set of valuations and q a state. For transition $t := (q, a, g, q_1)$, we write $(q, W) \xrightarrow{t} (q_1, W_1)$ if $W_1 = \{v_1 \mid (q, v) \xrightarrow{t, \delta} (q_1, v_1) \text{ for some } \delta \in \mathbb{R}_{\geq 0}\}$. As is usual with timed automata, zones are closed under the time elapse operation. We will show in the next section that starting from an event zone Z , the successors are also event zones: $(q, Z) \xrightarrow{t} (q_1, Z_1)$ implies Z_1 is an event zone too. We use this feature to define an event zone graph.

► **Definition 6** (Event zone graph). *Nodes are of the form (q, Z) where q is a state and Z is an event zone. The initial node is (q_0, Z_0) where q_0 is the initial state and Z_0 is given by $\bigwedge_{a \in \Sigma} (\infty \leq \overleftarrow{a}) \wedge (\overrightarrow{a} \leq 0)$. This is the set of all initial valuations, which is already closed under time elapse. For every node (q, Z) and every transition $t := (q, a, g, q_1)$ there is a transition $(q, Z) \xrightarrow{t} (q_1, Z_1)$ in the event zone graph. A node (q, Z) is accepting if $q \in F$ and $Z \cap Z_f$ is non-empty where the final zone Z_f is defined by $\bigwedge_{a \in \Sigma} \overrightarrow{a} \leq -\infty$.*

Two examples of ECA and their event zone graphs are given in Figure 3 and Figure 4 of Appendix A.

Similar to the case of timed automata, the event zone graph can be used to decide reachability. The next lemma follows by a straightforward adaptation of the corresponding proof [13] from timed automata.

► **Proposition 7.** *The event zone graph of an ECA is sound and complete for reachability.*

However, as in the case of zone graphs for timed automata, the event zone graph for an ECA is also not guaranteed to be finite. We will now define what a simulation is and see how it can be used to get a finite truncation of the event zone graph, which is still sound and complete for reachability.

► **Definition 8** (Simulation). *A simulation relation on the semantics of an ECA is a reflexive, transitive relation $(q, v) \preceq (q, v')$ relating configurations with the same control state and (1) for every $(q, v) \xrightarrow{\delta} (q, v + \delta)$, we have $(q, v') \xrightarrow{\delta} (q, v' + \delta)$ and $(q, v + \delta) \preceq (q, v' + \delta)$, (2) for every transition t , if $(q, v) \xrightarrow{t} (q_1, v_1)$ for some valuation v_1 , then $(q, v') \xrightarrow{t} (q_1, v'_1)$ for some valuation v'_1 with $(q_1, v_1) \preceq (q_1, v'_1)$.*

For two event zones Z, Z' , we say $(q, Z) \preceq (q, Z')$ if for every $v \in Z$ there exists $v' \in Z'$ such that $(q, v) \preceq (q, v')$. The simulation \preceq is said to be finite if for every sequence $(q_1, Z_1), (q_2, Z_2), \dots$ of reachable nodes, there exists $j > i$ such that $(q_j, Z_j) \preceq (q_i, Z_i)$.

The reachability algorithm enumerates the nodes of the event zone graph and uses \preceq to truncate nodes that are smaller with respect to the simulation.

► **Definition 9** (Reachability algorithm). *Let \mathcal{A} be an ECA and \preceq a finite simulation for \mathcal{A} . Add the initial node of the event zone graph (q_0, Z_0) to a Waiting list. Repeat the following until Waiting list is empty:*

- *Pop a node (q, Z) from the Waiting list and add it to the Passed list.*
- *For every $(q, Z) \xrightarrow{t} (q_1, Z_1)$: if there exists a (q_1, Z'_1) in the Passed or Waiting lists such that $(q_1, Z_1) \preceq (q_1, Z'_1)$, discard (q_1, Z_1) ; else add (q_1, Z_1) to the Waiting list.*

If some accepting node is reached, the algorithm terminates and returns a Yes. Else, it continues until there are no further nodes to be explored and returns a No answer.

The correctness of the reachability algorithm follows once again from the correctness of the simulation approach in timed automata [22]. Moreover, termination is guaranteed when the simulation used is finite.

► **Theorem 10.** *An ECA has an accepting run iff the reachability algorithm returns Yes.*

We have now presented the framework for the simulation approach in its entirety. However, to make it functional, we will need the following.

1. An efficient representation for event zones and algorithms to compute successors.
2. A concrete simulation relation \preceq for ECA with an efficient simulation test $(q, Z) \preceq (q, Z')$.
3. A proof that \preceq is finite, to guarantee termination of the reachability algorithm.

In the rest of the paper, we show how these can be achieved. To start with, for standard timed automata, zones are represented using Difference-Bound-Matrices (DBMs) [14]. For such a representation to work on event zones, we will need to incorporate the fact that valuations can now take $+\infty$ and $-\infty$. In Section 4, we propose a way to merge $+\infty$ and $-\infty$ seamlessly into the DBM technology. In the subsequent Section 5, we define a simulation for ECA based on \mathcal{G} -simulation, develop some technical machinery and present an efficient simulation test. Finally, in Section 6, we deal with the main problem of showing finiteness. For this, we prove some non-trivial invariants on the event zones that are reachable in ECA and use them to show a surprising property regarding prophecy clocks. More precisely, we show that constraints involving prophecy clocks in reachable event zones come from a finite set depending on the maximum constant of the ECA only.

4 Computing with event zones and distance graphs

We now show that event zones can be represented using Difference-Bound-Matrices (DBMs) and the operations required for the reachability algorithm can be implemented using DBMs. Each entry in a DBM encodes a constraint of the form $x - y \triangleleft c$. For timed automata analysis, the entries are (\triangleleft, c) where $c \in \mathbb{R}$ and $\triangleleft \in \{<, \leq\}$, or $(\triangleleft, c) = (<, \infty)$. In our case, we will need to deal with valuations having $+\infty$ or $-\infty$. For this purpose, we first extend weights to include $(\leq, -\infty)$ and (\leq, ∞) and define an arithmetic that admits the new entries in a natural way.

► **Definition 11 (Weights).** *Let $\mathcal{C} = \{(\leq, -\infty)\} \cup \{(\triangleleft, c) \mid c \in \mathbb{R} \cup \{\infty\} \text{ and } \triangleleft \in \{\leq, <\}\}$, called the set of weights.*

- **Order.** *Define $(\triangleleft_1, c_1) < (\triangleleft_2, c_2)$ when either (1) $c_1 < c_2$, or (2) $c_1 = c_2$ and \triangleleft_1 is $<$ while \triangleleft_2 is \leq . This is a total order, in particular $(\leq, -\infty) < (\triangleleft, c) < (<, \infty) < (\leq, \infty)$ for all $c \in \mathbb{R}$.*
- **Sum.** *Let $\alpha, \beta, \gamma, (\triangleleft_1, c_1), (\triangleleft_2, c_2) \in \mathcal{C}$ with $\beta \neq (\leq, \infty)$, $\gamma \notin \{(\leq, -\infty), (\leq, \infty)\}$ and $c_1, c_2 \in \mathbb{R}$. We define the operation of sum on weights as follows.*

$$\begin{aligned} (\leq, \infty) + \alpha &= (\leq, \infty) & (\leq, -\infty) + \beta &= (\leq, -\infty) & (<, \infty) + \gamma &= (<, \infty) \\ (\triangleleft_1, c_1) + (\triangleleft_2, c_2) &= (\triangleleft, c_1 + c_2) & \text{with } \triangleleft &= \leq \text{ if } \triangleleft_1 = \triangleleft_2 = \leq \text{ and } \triangleleft &= < \text{ otherwise.} \end{aligned}$$

The intuition behind the above definition of order is that when $(\triangleleft, c) < (\triangleleft', c')$, the set of valuations that satisfies a constraint $x - y \triangleleft c$ is contained in the solution set of $x - y \triangleleft' c'$. For the sum, we have the following lemma which gives the idea behind our choice of definition.

► **Lemma 12.** *Let v be a valuation, x, y, z be event clocks and $(\triangleleft_1, c_1), (\triangleleft_2, c_2) \in \mathcal{C}$. If $v \models x - y \triangleleft_1 c_1$ and $v \models y - z \triangleleft_2 c_2$, then $v \models x - z \triangleleft c$ where $(\triangleleft, c) = (\triangleleft_1, c_1) + (\triangleleft_2, c_2)$.*

Equipped with the weights and the arithmetic over it, we will work with a graph representation of zones (as so-called distance graphs), instead of matrices (i.e., DBMs), since this makes the analysis more convenient. We wish to highlight that our definition of weights, order and sum have been chosen to ensure that this notion of distance graphs remains identical to the one for usual TA. As a consequence, we are able to adapt many of the well-known properties about distance graphs for ECA.

► **Definition 13** (Distance graphs). *A distance graph is a weighted directed graph, with vertices being $X_P \cup X_H \cup \{0\}$ where 0 is a special vertex that plays the role of constant 0. Edges are labeled with weights from \mathcal{C} . An edge $x \xrightarrow{c} y$ represents the constraint $y - x \triangleleft c$. For a graph \mathbb{G} , we define $\llbracket \mathbb{G} \rrbracket := \{v \mid v \models y - x \triangleleft c \text{ for all edges } x \xrightarrow{c} y \text{ in } \mathbb{G}\}$. Further,*

- *The weight of a path in a distance graph \mathbb{G} is the sum of the weight of its edges. A cycle in \mathbb{G} is said to be negative if its weight is strictly less than $(\leq, 0)$.*
- *A graph \mathbb{G} is said to be in canonical form if it has no negative cycles and for each pair of vertices x, y , the weight of $x \rightarrow y$ is not greater than the weight of any path from x to y .*
- *For two graphs $\mathbb{G}_1, \mathbb{G}_2$, we write $\min(\mathbb{G}_1, \mathbb{G}_2)$ for the distance graph obtained by setting the weight of each edge to the minimum of the corresponding weights in \mathbb{G}_1 and \mathbb{G}_2 .*

For an event zone Z , we write $\mathbb{G}(Z)$ for the canonical distance graph that satisfies $\llbracket \mathbb{G}(Z) \rrbracket = Z$. We denote by Z_{xy} the weight of the edge $x \rightarrow y$ in $\mathbb{G}(Z)$.

We will make use of an important property, which has been shown when weights come from $\mathcal{C} \setminus \{(\leq, +\infty), (\leq, -\infty)\}$, but continues to hold even with the new weights added.

► **Lemma 14.** *For every distance graph \mathbb{G} , we have $\llbracket \mathbb{G} \rrbracket = \emptyset$ iff \mathbb{G} has a negative cycle.*

Successor computation. To implement the computation of transitions $(q, Z) \xrightarrow{t} (q_1, Z_1)$ in an event zone graph, we will make use of some operations on event zones that we define below. Using distance graphs, we show that these operations preserve event zones, that is, starting from an event zone and applying any of the operations leads to an event zone again. Thanks to the algebra over the new weights that we have defined, the arguments are very similar to the case of standard timed automata.

► **Definition 15** (Operations on event zones). *Let g be a guard and Z an event zone.*

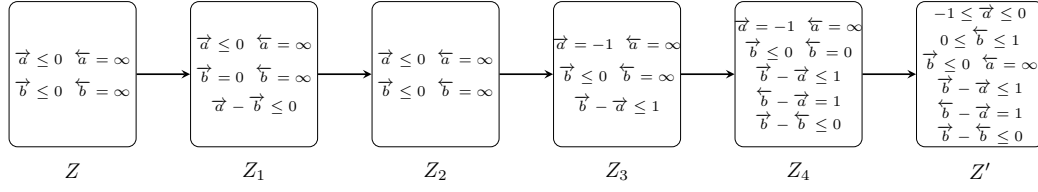
- *Guard intersection: $Z \wedge g := \{v \mid v \in Z \text{ and } v \models g\}$*
- *Release: $[\vec{a}]Z = \bigcup_{v \in Z} [\vec{a}]v$*
- *Reset: $[\overleftarrow{a}]Z = \{[\overleftarrow{a}]v \mid v \in Z\}$*
- *Time elapse: $\vec{Z} = \{v + \delta \mid v \in Z, \delta \in \mathbb{R}_{\geq 0} \text{ s.t. } v + \delta \models \bigwedge_{a \in \Sigma} \vec{a} \leq 0\}$*

A guard g can be seen as yet another event zone and hence guard intersection is just an intersection operation between two event zones. By definition, for a transition $t := (q, a, g, q')$ and a node (q, Z) the successor $(q, Z) \xrightarrow{t} (q', Z')$ can be computed in the following sequence:

$$Z_1 := Z \cap (0 \leq \vec{a}) \quad Z_2 := [\vec{a}]Z_1 \quad Z_3 := Z_2 \cap g \quad Z_4 := [\overleftarrow{a}]Z_3 \quad Z' := \vec{Z}_4$$

As an example, in Figure 2, suppose an action b with guard $\vec{a} = -1$ ($\vec{a} \leq -1 \wedge -1 \leq \vec{a}$) is fired from Zone Z as depicted, applying the above sequence in order gives Z_1, Z_2, Z_3, Z_4 resulting in the successor zone Z' , as depicted in the figure.

We are now ready to state Theorem 16 that says that the operations on event zones translate easily to operations on distance graphs and that the successor of an event zone is an event zone. Except for the release operation $[\vec{a}]$, the rest of the operations are standard in timed automata, but need to be extended to cope with the new weights $(\leq, +\infty), (\leq, -\infty)$.



■ **Figure 2** Successor computation from event zone Z on an action b with guard $\vec{a} = -1$.

We show that we can perform all these operations in the new algebra with quadratic complexity, as in timed automata without diagonal constraints [32].

► **Theorem 16.** *Let Z be an event zone and \mathbb{G} be its canonical distance graph. Let g be a guard. We can compute, in $\mathcal{O}(|X_P \cup X_H|^2)$ time, distance graphs \mathbb{G}_g , $[\vec{a}]\mathbb{G}$, $[\overleftarrow{a}]\mathbb{G}$ and $\overrightarrow{\mathbb{G}}$ in canonical form, such that $Z \wedge g = \llbracket \mathbb{G}_g \rrbracket$, $[\vec{a}]Z = \llbracket [\vec{a}]\mathbb{G} \rrbracket$, $[\overleftarrow{a}]Z = \llbracket [\overleftarrow{a}]\mathbb{G} \rrbracket$, and $\overrightarrow{Z} = \llbracket \overrightarrow{\mathbb{G}} \rrbracket$.*

Proof (sketch). The distance graphs \mathbb{G}_g , $[\vec{a}]\mathbb{G}$, $[\overleftarrow{a}]\mathbb{G}$ and $\overrightarrow{\mathbb{G}}$ are computed as follows:

- Guard intersection: a distance graph \mathbb{G}_g is obtained from \mathbb{G} as follows,
 - for each atomic constraint $x \triangleleft c$ in g , replace weight of edge $0 \rightarrow x$ with the minimum of its weight in \mathbb{G} and (\triangleleft, c) ,
 - for each atomic constraint $d \triangleleft y$ in g , replace weight of edge $y \rightarrow 0$ with the minimum of its weight in \mathbb{G} and $(\triangleleft, -d)$,
 - canonicalize the resulting graph.
- Release: a distance graph $[\vec{a}]\mathbb{G}$ is obtained from \mathbb{G} by
 - removing all edges involving \vec{a} and then
 - adding the edges $0 \xrightarrow{(\leq, 0)} \vec{a}$ and $\vec{a} \xrightarrow{(\leq, \infty)} 0$, and then
 - canonicalizing the resulting graph.
- Reset: a distance graph $[\overleftarrow{a}]\mathbb{G}$ is obtained from \mathbb{G} by
 - removing all edges involving \overleftarrow{a} and then
 - adding the edges $0 \xrightarrow{(\leq, 0)} \overleftarrow{a}$ and $\overleftarrow{a} \xrightarrow{(\leq, 0)} 0$, and then
 - canonicalizing the resulting graph.
- Time elapse: the distance graph $\overrightarrow{\mathbb{G}}$ is obtained by the following transformation:
 - if \overleftarrow{x} is defined, i.e., the weight of $0 \rightarrow \overleftarrow{x}$ is not (\leq, ∞) , then replace it with (\leq, ∞) ,
 - if \overrightarrow{x} is defined, i.e., the weight of $0 \rightarrow \overrightarrow{x}$ is not $(\leq, -\infty)$, then replace it with $(\leq, 0)$,
 - canonicalize the resulting graph.

It is not hard to prove that they correspond to the operations on event zones. Other than canonicalization, it can be easily checked that these operations can be computed in quadratic time. Though canonicalization is cubic time in general, in each of the special cases above, it can be implemented in quadratic time. ◀

5 A concrete simulation relation for ECAs

We fix an event-clock automaton $\mathcal{A} = (Q, \Sigma, X, T, q_0, F)$ for this section. We will define a simulation relation $\preceq_{\mathcal{A}}$ on the configurations of the ECA. We first define a map \mathcal{G} from Q to sets of atomic constraints. The map \mathcal{G} is obtained as the least fixpoint of the set of equations:

$$\mathcal{G}(q) = \{ \vec{b} \leq 0, 0 \leq \overrightarrow{b} \mid b \in \Sigma \} \cup \bigcup_{(q, a, g, q') \in T} \text{split}(g) \cup \text{pre}(a, \mathcal{G}(q'))$$

13:10 Simulations for Event-Clock Automata

where $\text{split}(g)$ is the set of atomic constraints occurring in g and, for a set of atomic constraints G , $\text{pre}(a, G)$ is defined as the set of constraints in G except those on \vec{a} or \overleftarrow{a} . Notice that constraints in $\mathcal{G}(q)$ use the constant 0 and constants used in constraints of \mathcal{A} .

Let G be a set of atomic constraints. The preorder \preceq_G is defined on valuations by

$$v \preceq_G v' \quad \text{if } \forall \varphi \in G, \forall \delta \geq 0, \quad v + \delta \models \varphi \implies v' + \delta \models \varphi.$$

Notice that in the condition above, we *do not* restrict δ to those such that $v + \delta$ is a valuation: we may have $v(\vec{a}) + \delta > 0$ for some $a \in \Sigma$. This is crucial for the proof of Theorem 17 below. It also allows to get a clean characterizations of the simulation (Lemma 18) which in turn is useful for deriving the simulation test and in showing finiteness. Based on \preceq_G and the $\mathcal{G}(q)$ computation, we can define a preorder $\preceq_{\mathcal{A}}$ between configurations of ECA \mathcal{A} as $(q, v) \preceq_{\mathcal{A}} (q', v')$ if $q = q'$ and $v \preceq_{\mathcal{G}(q)} v'$.

► **Theorem 17.** *The relation $\preceq_{\mathcal{A}}$ is a simulation on the transition system $S_{\mathcal{A}}$ of ECA \mathcal{A} .*

When $G = \{\varphi\}$ is a singleton, we simply write \preceq_{φ} for $\preceq_{\{\varphi\}}$. The definition of the \preceq_G simulation above in some sense declares what is expected out of the simulation. Below, we give a constructive characterization of the simulation in terms of the constants used and the valuations. For example, if $v(\overleftarrow{a}) = 3$ and $\overleftarrow{a} \leq 5$ is a constraint in G , point 2 below says that all v' with $v'(\overleftarrow{a}) \leq 3$ simulate v . The next lemma is a generalization of Lemma 8 from [18] to our setting containing prophecy clocks and the undefined values $+\infty$ and $-\infty$.

► **Lemma 18.** *Let v, v' be valuations and G a set of atomic constraints. We have*

1. $v \preceq_G v'$ iff $v \preceq_{\varphi} v'$ for all $\varphi \in G$.
2. $v \preceq_{x \triangleleft c} v'$ iff $v(x) \not\triangleleft c$ or $v'(x) \leq v(x)$ or $(\triangleleft, c) = (\leq, \infty)$ or $(\triangleleft, c) = (<, \infty) \wedge v'(x) < \infty$.
3. $v \preceq_{c \triangleleft x} v'$ iff $c \triangleleft v'(x)$ or $v(x) \leq v'(x)$ or $(c, \triangleleft) = (\infty, <)$ or $(c, \triangleleft) = (\infty, \leq) \wedge v(x) < \infty$.

We now state some useful properties that get derived from Lemma 18.

► **Remark 19.** *Let v, v' be valuations and G a set of atomic constraints.*

1. For all $a \in \Sigma$, if $\{0 \leq \vec{a}, \vec{a} \leq 0\} \subseteq G$ and $v \preceq_G v'$ then $v(\vec{a}) = v'(\vec{a})$.
2. Let $x \triangleleft_1 c_1$ and $x \triangleleft_2 c_2$ be constraints with $(\triangleleft_1, c_1) \leq (\triangleleft_2, c_2) < (<, \infty)$ (we say that $x \triangleleft_1 c_1$ is subsumed by $x \triangleleft_2 c_2$). If $v \preceq_{x \triangleleft_2 c_2} v'$ then $v \preceq_{x \triangleleft_1 c_1} v'$.
Indeed, from $(\triangleleft_2, c_2) < (<, \infty)$ and $v \preceq_{x \triangleleft_2 c_2} v'$ we get $v'(x) \leq v(x)$ or $v(x) \not\triangleleft_2 c_2$, which implies $v(x) \not\triangleleft_1 c_1$ since $(\triangleleft_1, c_1) \leq (\triangleleft_2, c_2)$.
3. Let $c_1 \triangleleft_1 x$ and $c_2 \triangleleft_2 x$ be constraints with $(c_1, \triangleleft_1) \leq (c_2, \triangleleft_2) < (\infty, \leq)$ (we say that $c_1 \triangleleft_1 x$ is subsumed by $c_2 \triangleleft_2 x$). If $v \preceq_{c_2 \triangleleft_2 x} v'$ then $v \preceq_{c_1 \triangleleft_1 x} v'$.
Indeed, from $(c_2, \triangleleft_2) < (\infty, \leq)$ and $v \preceq_{c_2 \triangleleft_2 x} v'$ we get $v(x) \leq v'(x)$ or $c_2 \triangleleft_2 v'(x)$, which implies $c_1 \triangleleft_1 v'(x)$ since $(c_1, \triangleleft_1) \leq (c_2, \triangleleft_2)$.

The ordering between *lower weights* is defined by $(c_1, \triangleleft_1) < (c_2, \triangleleft_2)$ if $c_1 < c_2$ or $c_1 = c_2$, $\triangleleft_1 = \leq$ and $\triangleleft_2 = <$. We have $(c_1, \triangleleft_1) < (c_2, \triangleleft_2)$ iff $(\triangleleft_2, -c_2) < (\triangleleft_1, -c_1)$.

Before lifting the simulation to event zones, we present a central technical object that will be used from time to time in the next set of results.

Distance graph for valuations that simulate a valuation v . For a valuation v , we let $\uparrow_G v = \{v' \in \mathbb{V} \mid v \preceq_G v'\}$, i.e., the set of valuations v' which simulate v . We will define a distance graph, denoted $\mathbb{G}_G(v)$, such that $\llbracket \mathbb{G}_G(v) \rrbracket = \uparrow_G v$. We remark that $\llbracket \mathbb{G}_G(v) \rrbracket$ is not really a zone since it may use constants that are not integers.

We assume that G contains $\{0 \leq \vec{a}, \vec{a} \leq 0 \mid a \in \Sigma\}$ so that $v \preceq_G v'$ implies $v(\vec{a}) = v'(\vec{a})$ for all prophecy clocks \vec{a} with $a \in \Sigma$. We remove from G constraints equivalent to true, such as $x \leq \infty$, $-3 < \overleftarrow{a}$ or $0 \leq \overleftarrow{a}$, or equivalent to false, such as $\overleftarrow{a} < 0$ or $\infty < x$. Also, by

Remark 19, we may remove from G constraints that are subsumed by other constraints in G , while not changing the simulation relation. Hence, for history clocks, we have at most one upper-bound constraint $\overleftarrow{a} \triangleleft c$ with $(\leq, 0) \leq (\triangleleft, c) < (<, \infty)$, and at most one lower-bound constraint $c \triangleleft \overleftarrow{a}$ with $(0, \leq) < (c, \triangleleft) < (\infty, \leq)$. From now on, we always assume that the sets G of atomic constraints that we consider satisfy the above conditions.

The definition of the distance graph $\mathbb{G}_G(v)$ which defines $\uparrow_G v$ is based on Lemma 18.

- For each prophecy clock \overrightarrow{a} , we have the edges $\overrightarrow{a} \xrightarrow{(\leq, -v(\overrightarrow{a}))} 0$ and $0 \xrightarrow{(\leq, v(\overrightarrow{a}))} \overrightarrow{a}$.
- For each history clock \overleftarrow{a} , we have the edge $0 \rightarrow \overleftarrow{a}$ with weight
 - $(\leq, v(\overleftarrow{a}))$ if $\overleftarrow{a} \triangleleft c \in G$ with $(\triangleleft, c) < (<, \infty)$ and $v(\overleftarrow{a}) \triangleleft c$,
 - $(<, \infty)$ if we are not in the case above and $\overleftarrow{a} < \infty \in G$, $v(\overleftarrow{a}) < \infty$,
 - (\leq, ∞) otherwise.
- For each history clock \overleftarrow{a} , we have the edge $\overleftarrow{a} \rightarrow 0$ with weight
 - $(\leq, -\infty)$ if $\infty \leq \overleftarrow{a} \in G$ and $v(\overleftarrow{a}) = \infty$, and if we are not in this case:
 - $(\triangleleft, -c)$ if $c \triangleleft \overleftarrow{a} \in G$ with $(c, \triangleleft) < (\infty, \leq)$ and $c \triangleleft v(\overleftarrow{a})$,
 - $(\leq, -v(\overleftarrow{a}))$ if $c \triangleleft \overleftarrow{a} \in G$ with $(c, \triangleleft) < (\infty, \leq)$ and $c \not\triangleleft v(\overleftarrow{a})$,
 - $(\leq, 0)$ otherwise.

With this definition, while $\mathbb{G}_G(v)$ is not in canonical form, it has the desired property:

► **Lemma 20.** *We have $v \preceq_G v'$ iff v' satisfies all the constraints of $\mathbb{G}_G(v)$.*

Simulation for event zones and an efficient algorithmic check. Let Z, Z' be two event zones and G be a set of atomic constraints. We say that Z is G -simulated by Z' , denoted $Z \preceq_G Z'$, if for all $v \in Z$ there exists $v' \in Z'$ such that $v \preceq_G v'$. Finally, we define $(q, Z) \preceq_{\mathcal{A}} (q', Z')$ if $q = q'$ and $Z \preceq_{G(q)} Z'$. In the rest of this section, we show how to check this relation efficiently. We let $\downarrow_G Z = \{v \in \mathbb{V} \mid v \preceq_G v' \text{ for some } v' \in Z\}$. Notice that $Z \preceq_G Z'$ iff $Z \subseteq \downarrow_G Z'$ iff $\downarrow_G Z = \downarrow_G Z'$.

► **Lemma 21.** *For event zones Z, Z' , we have $Z \not\preceq_G Z'$ iff $\exists v \in Z$ with $\uparrow_G v \cap Z' = \emptyset$.*

To check $Z \not\preceq_G Z'$, we require a valuation $v \in Z$ with a witness that $\uparrow_G v \cap Z'$ is empty. In the language of distance graphs, the witness will be a negative cycle in $\min(\uparrow_G v, Z')$. We show that if $\uparrow_G v \cap Z'$ is empty, then there is a small witness, i.e., a negative cycle containing at most three edges, and belonging to one of three specific forms.

► **Lemma 22.** *Let v be a valuation, Z' a non-empty reachable event zone with canonical distance graph $\mathbb{G}(Z')$ and G a set of atomic constraints. Then, $\uparrow_G v \cap Z'$ is empty iff there is a negative cycle in one of the following forms:*

1. $0 \rightarrow x \rightarrow 0$ with $0 \rightarrow x$ from $\mathbb{G}_G(v)$ and $x \rightarrow 0$ from $\mathbb{G}(Z')$,
2. $0 \rightarrow y \rightarrow 0$ with $0 \rightarrow y$ from $\mathbb{G}(Z')$ and $y \rightarrow 0$ from $\mathbb{G}_G(v)$, and
3. $0 \rightarrow x \rightarrow y \rightarrow 0$, with weight of $x \rightarrow y$ from $\mathbb{G}(Z')$ and the others from $\mathbb{G}_G(v)$. Moreover, this negative cycle has finite weight.

Proof. Since $Z' \neq \emptyset$, the distance graph $\mathbb{G}(Z')$ has no negative cycle. The same holds for $\mathbb{G}_G(v)$ since $v \in \uparrow_G v \neq \emptyset$. We know that $\uparrow_G v \cap Z' = \emptyset$ iff there is a (simple) negative cycle using edges from $\mathbb{G}_G(v)$ and from $\mathbb{G}(Z')$. Since $\mathbb{G}(Z')$ is in canonical form, we may restrict to negative cycles which do not use two consecutive edges from $\mathbb{G}(Z')$. Now all edges of $\mathbb{G}_G(v)$ are adjacent to node 0. Hence, if a simple cycle uses an edge from $\mathbb{G}(Z')$ which is adjacent to 0, it consists of only two edges $0 \rightarrow x \rightarrow 0$, one from $\mathbb{G}(Z')$ and one from $\mathbb{G}_G(v)$. Otherwise, the simple cycle is of the form $0 \rightarrow x \rightarrow y \rightarrow 0$ where the edge $x \rightarrow y$ is from $\mathbb{G}(Z')$ and the other two edges are from $\mathbb{G}_G(v)$. It remains to show that the two clock negative cycle $0 \rightarrow x \rightarrow y \rightarrow 0$ can be considered to have finite weight, i.e., weight is not $(\leq, -\infty)$.

13:12 Simulations for Event-Clock Automata

For the cycle to have weight $(\leq, -\infty)$, one of the edges should have weight $(\leq, -\infty)$ and the others should have a weight different from (\leq, ∞) . We will show that for every such combination, there is a smaller negative cycle with a single clock and 0. Hence we can ignore negative cycles of the form $0 \rightarrow x \rightarrow y \rightarrow 0$ with weight $(\leq, -\infty)$.

Suppose $Z'_{xy} = (\leq, -\infty)$. Then, for every valuation in $u \in Z'$, we have $u(y) - u(x) \leq -\infty$, which implies $u(y) = -\infty$ or $u(x) = +\infty$. If $u(x) = +\infty$ for some valuation $u \in Z'$, then the value of x is $+\infty$ for every valuation in Z' . This follows from the successor computation: initially, history clocks are undefined, and then an action a defines \overleftarrow{a} , and from that point onwards, \overleftarrow{a} is always $< \infty$. Now, if x is not an undefined history clock in Z' , then we need to have $u(y) = -\infty$ for all valuations of Z' . Therefore, either x is a history clock that is undefined in Z' or y is a prophecy clock that is undefined in Z' . In the former case, $Z'_{x0} = (\leq, -\infty)$ and in the latter case $Z'_{0y} = (\leq, -\infty)$. This gives a smaller negative cycle $0 \rightarrow x \xrightarrow{Z'_{x0}} 0$ or $0 \xrightarrow{Z'_{0y}} y \rightarrow 0$ with the remaining edge $0 \rightarrow x$ or $y \rightarrow 0$ coming from $\mathbb{G}_G(v)$, since by our hypothesis of a negative cycle, these edges have weight different from (\leq, ∞) .

Suppose the weight of $0 \rightarrow x$ is $(\leq, -\infty)$. This can happen only when x is a prophecy clock, $v(x) = -\infty$ and weight of $0 \rightarrow x$ is $(\leq, v(x))$. Since $Z'_{xy} \neq (\leq, \infty)$, we infer $Z'_{x0} \neq (\leq, \infty)$ by \dagger_1 of Lemma 26. Hence $0 \xrightarrow{(\leq, v(x))} x \xrightarrow{Z'_{x0}} 0$ is also a negative cycle.

Suppose $y \rightarrow 0$ has weight $(\leq, -\infty)$. This can happen only when y is a history clock and $v(y) = +\infty$. Since $Z'_{xy} \neq (\leq, \infty)$, we obtain $Z'_{0y} \neq (\leq, \infty)$ and hence $0 \xrightarrow{Z'_{0y}} y \xrightarrow{(\leq, -v(y))} 0$ is a negative cycle. \blacktriangleleft

We now have all the results required to state our inclusion test. Using the above lemma, and relying on a careful analysis (as shown in the full version [2]), we obtain the following theorem.

► Theorem 23. *Let Z, Z' be non-empty reachable zones, and G a set of atomic constraints containing $\overrightarrow{a} \leq 0$ and $0 \leq \overrightarrow{a}$ for every prophecy clock \overrightarrow{a} . Then, $Z \not\leq_G Z'$ iff one of the following conditions holds:*

1. $Z'_{x0} < Z_{x0}$ for some prophecy clock x , or for some history clock x with
 - $(x < \infty) \in G$ and $Z'_{x0} = (\leq, -\infty)$, or
 - $(x \triangleleft_1 c) \in G$ for $c \in \mathbb{N}$ and $(\leq, 0) \leq Z_{x0} + (\triangleleft_1, c)$.
2. $Z'_{0y} < Z_{0y}$ for some prophecy clock y , or for some history clock y with
 - $(\infty \leq y) \in G$ and $Z_{0y} = (\leq, \infty)$, or
 - $(d \triangleleft_2 y) \in G$ for $d \in \mathbb{N}$ and $Z'_{0y} + (\triangleleft_2, -d) < (\leq, 0)$
3. $Z'_{xy} < Z_{xy}$ and Z'_{xy} is finite for two distinct (prophecy or history) clocks x, y with $(x \triangleleft_1 c), (d \triangleleft_2 y) \in G$ for $c, d \in \mathbb{N}$ and $(\leq, 0) \leq Z_{x0} + (\triangleleft_1, c)$ and $Z'_{xy} + (\triangleleft_2, -d) < Z_{x0}$.

From Theorem 23, we can see that the inclusion test requires iteration over clocks x, y and checking if the conditions are satisfied by the respective weights.

► Corollary 24. *Checking if $(q, Z) \preceq_{\mathcal{A}} (q', Z')$ can be done in time $\mathcal{O}(|X|^2) = \mathcal{O}(|\Sigma|^2)$.*

6 Finiteness of the simulation relation

In this section, we will show that the simulation relation $\preceq_{\mathcal{A}}$ defined in Section 5 is finite, which implies that the reachability algorithm of Definition 9 terminates. Recall that given an event clock automaton \mathcal{A} , we have an associated map \mathcal{G} from states of \mathcal{A} to sets of atomic constraints. Let $M = \max\{|c| \mid c \in \mathbb{Z} \text{ is used in some constraint of } \mathcal{A}\}$, the maximal constant of \mathcal{A} . We have $M \in \mathbb{N}$ and constraints in the sets $\mathcal{G}(q)$ use constants in $\{-\infty, \infty\} \cup \{c \in \mathbb{Z} \mid |c| \leq M\}$.

Recall that the simulation relation $\preceq_{\mathcal{A}}$ was defined on nodes of the event zone graph $\text{EZG}(\mathcal{A})$ by $(q, Z) \preceq_{\mathcal{A}} (q', Z')$ if $q = q'$ and $Z \preceq_{\mathcal{G}(q)} Z'$. This simulation relation $\preceq_{\mathcal{A}}$ is *finite* if for any infinite sequence $(q, Z_0), (q, Z_1), (q, Z_2), \dots$ of *reachable* nodes in $\text{EZG}(\mathcal{A})$ we find $i < j$ with $(q, Z_j) \preceq_{\mathcal{A}} (q, Z_i)$, i.e., $Z_j \preceq_{\mathcal{G}(q)} Z_i$. Notice that we restrict to *reachable* zones in the definition above. Our goal now is to prove that the relation $\preceq_{\mathcal{A}}$ is finite. The structure of the proof is as follows.

1. We prove in Lemma 26 that for any *reachable* node (q, Z) of $\text{EZG}(\mathcal{A})$, the distance graph $\mathbb{G}(Z)$ in canonical form satisfies a set of (\dagger) conditions which depend only on the maximal constant M of \mathcal{A} .
2. We introduce an equivalence relation \sim_M of *finite index* on valuations (depending on M only) and show in Lemma 28 that, if G is a set of atomic constraints using constants in $\{c \in \mathbb{Z} \mid |c| \leq M\} \cup \{-\infty, \infty\}$ and if Z is a zone such that its distance graph $\mathbb{G}(Z)$ in canonical form satisfies (\dagger) conditions, then $\downarrow_G Z$ is a union of \sim_M equivalence classes.

We start with a lemma which highlights an important property of *prophecy* clocks in reachable event zones. This property is essential for the proof of the (\dagger) conditions. The proof follows from the observation that the property is true in the initial zone, and is invariant under the zone operations, namely, guard intersection, reset, release and time elapse.

► **Lemma 25.** *Let Z be a reachable event zone. For every valuation $v \in Z$, and for every prophecy clock \vec{x} , if $-\infty < v(\vec{x}) < -M$, then $v[\vec{x} \mapsto \alpha] \in Z$ for every $-\infty < \alpha < -M$.*

There is no similar version of the above lemma for history clocks. A reset of a history clock makes its value exactly equal to 0 in every valuation and creates non-trivial diagonal constraints with other clocks. Moreover repeated resets can generate arbitrarily large diagonal constraints, for e.g., a loop with guard $x = 1$ and reset x . This is why simulations are particularly needed to control history clocks. Notice that in our simulation $v \preceq_G v'$, we have $v(\vec{a}) = v'(\vec{a})$: there is no abstraction of the value of prophecy clocks and the simulation relation by itself does not have any means to show finiteness. However, as we show below, the reachable zones themselves take care of finiteness with respect to prophecy clocks. The challenge is then to combine this observation on prophecy clocks along with the non-trivial simulation happening for history clocks to prove that we still get a finite simulation. This is the purpose of the above mentioned item 2.

Now, we give the (\dagger) conditions and prove that they are satisfied by distance graphs of reachable zones. In particular, the (\dagger) conditions imply that the weight of edges of the form $0 \rightarrow \vec{x}$, $\vec{x} \rightarrow 0$ and $\vec{x} \rightarrow \vec{y}$ belong to the finite set $\{(\leq, -\infty), (<, \infty), (\leq, \infty)\} \cup \{(<, c) \mid c \in \mathbb{Z} \wedge -M \leq c \leq M\}$. For an example, see Figure 4. Thus, we obtain as a corollary that, for EPA, we do not even need simulation to obtain finiteness.

► **Lemma 26.** *Let (q, Z) be a reachable node in $\text{EZG}(\mathcal{A})$ with $Z \neq \emptyset$. Then, the distance graph $\mathbb{G}(Z)$ in canonical form satisfies the (\dagger) conditions:*

- †₁ *If $Z_{\vec{x}0} = (\leq, \infty)$ then $Z_{\vec{x}y} = (\leq, \infty)$ for all $y \neq \vec{x}$.*
- †₂ *If $Z_{\vec{x}0} = (<, \infty)$ then for all $y \neq \vec{x}$, either y is a prophecy clock which is undefined in Z and $Z_{\vec{x}y} = Z_{0y} = (\leq, -\infty)$ or $Z_{\vec{x}y} \in \{(<, \infty), (\leq, \infty)\}$.*
- †₃ *If $Z_{\vec{x}0} < (<, \infty)$ then $(\leq, 0) \leq Z_{\vec{x}0} \leq (\leq, M)$.*
- †₄ *If $Z_{\vec{x}\vec{y}} < (<, \infty)$ then $(\leq, 0) \leq Z_{\vec{x}0} \leq (\leq, M)$.*
- †₅ *Either $Z_{0\vec{y}} = (\leq, -\infty)$ (\vec{y} is undefined in Z), or $Z_{x0} + (<, -M) \leq Z_{x\vec{y}}$ for all $x \neq \vec{y}$ (including $x = 0$).*
- †₆ *Either $Z_{0\vec{x}} = (\leq, -\infty)$ or $(<, -M) \leq Z_{0\vec{x}} \leq (\leq, 0)$.*
- †₇ *Either $Z_{\vec{x}\vec{y}} \in \{(\leq, -\infty), (<, \infty), (\leq, \infty)\}$ or $(<, -M) \leq Z_{\vec{x}\vec{y}} \leq (\leq, M)$.*

Proof sketch. \dagger_4 follows immediately from $\dagger_1, \dagger_2, \dagger_3$ and \dagger_6, \dagger_7 can be inferred from \dagger_5 and the other conditions. So here, we focus on $\dagger_1, \dagger_2, \dagger_3$ and partially the case of \dagger_5 , leaving other details to the full version [2].

For \dagger_1 , since $Z_{\vec{x}0} = (\leq, \infty)$, there is a valuation $v \in Z$ with $v(\vec{x}) = -\infty$. Therefore, for every clock $y \neq \vec{x}$, we have $v(y - \vec{x}) = +\infty$. Since $v \in Z$, it satisfies the constraint on $y - \vec{x}$ given by $Z_{\vec{x}y}$. This is possible only when $Z_{\vec{x}y} = (\leq, \infty)$.

For \dagger_2 , assume that $Z_{\vec{x}0} = (<, \infty)$ and let $y \neq \vec{x}$. Consider first the case $Z_{0y} = (\leq, -\infty)$, i.e., y is a prophecy clock which is undefined in Z . Then, since $\mathbb{G}(Z)$ is in canonical form, we have $Z_{\vec{x}y} \leq Z_{\vec{x}0} + Z_{0y} = (<, \infty) + (\leq, -\infty) = (\leq, -\infty)$. The second case is when $Z_{0y} \neq (\leq, -\infty)$. This implies $Z_{\vec{x}y} \neq (\leq, -\infty)$ since otherwise we would get $Z_{0y} \leq Z_{0\vec{x}} + Z_{\vec{x}y} = (\leq, -\infty)$. We claim that there is a valuation $v \in Z$ with $-\infty < v(y)$ and $-\infty < v(\vec{x}) < -M$. Consider the distance graph \mathbb{G}' obtained from $\mathbb{G}(Z)$ by setting the weight of edge $y \rightarrow 0$ to $\min(Z_{y0}, (<, \infty))$ and of edge $0 \rightarrow \vec{x}$ to $\min(Z_{0\vec{x}}, (<, -M))$. We show that there are no negative cycles in this graph. Since $Z \neq \emptyset$, the candidates for being negative must use the new weight $(<, -M)$ of $0 \rightarrow \vec{x}$ or the new weight $(<, \infty)$ of $y \rightarrow 0$ or both. This gives the cycle $0 \rightarrow \vec{x} \rightarrow 0$ with weight $(<, -M) + Z_{\vec{x}0} = (<, \infty)$ since $Z_{\vec{x}0} = (<, \infty)$, the cycle $0 \rightarrow y \rightarrow 0$ with weight $Z_{0y} + (<, \infty)$ which is not negative since $Z_{0y} \neq (\leq, -\infty)$, and the cycle $y \rightarrow 0 \rightarrow \vec{x} \rightarrow y$ with weight $(<, \infty) + (<, -M) + Z_{\vec{x}y}$ which is not negative since $Z_{\vec{x}y} \neq (\leq, -\infty)$. Since \mathbb{G}' has no negative cycle, Lemma 14 implies $[\mathbb{G}'] \neq \emptyset$. Note that $[\mathbb{G}'] \subseteq [\mathbb{G}(Z)] = Z$. Finally, for all $v \in \mathbb{G}'$, we have $-\infty < v(y)$ and $-\infty < v(\vec{x}) < -M$, which proves the claim. By Lemma 25, $v_\alpha = v[\vec{x} \mapsto \alpha] \in Z$ for all $-\infty < \alpha < -M$. Now, $v_\alpha(y - \vec{x}) = v(y) - \alpha$ satisfies the constraint $Z_{\vec{x}y}$. We deduce that $Z_{\vec{x}y}$ is either $(<, \infty)$ or (\leq, ∞) .

Next, we turn to \dagger_3 . Suppose $Z_{\vec{x}0} = (\triangleleft, c)$ for some integer $c > M$. Then, there exists a valuation $v \in Z$ with $v(\vec{x}) = -c$ or $v(\vec{x}) = -c + \frac{1}{2}$ depending on whether \triangleleft is \leq or $<$. Since c, M are integers, we get $-\infty < v(\vec{x}) < -M$. By Lemma 25, $v[\vec{x} \mapsto \alpha] \in Z$ for all $-\infty < \alpha < -M$. In particular, $v' = v[\vec{x} \mapsto -c - 1] \in Z$. For this valuation, we have $v'(\vec{x}) = -c - 1$. This violates $Z_{\vec{x}0}$ which says $0 - v'(\vec{x}) \triangleleft c$, or seen differently, $-c \triangleleft v'(\vec{x})$.

Finally, for \dagger_5 , if $Z_{x0} = (\leq, -\infty)$ the condition is trivially true. If $Z_{x0} \in \{(<, \infty), (\leq, \infty)\}$ then x is a prophecy clock and \dagger_5 follows from \dagger_1, \dagger_2 . Therefore, we assume $Z_{x0} = (\triangleleft, c)$ for $c \in \mathbb{Z}$. The left hand side of the condition is $Z_{x0} + (<, -M) = (<, c - M)$, with $c - M \in \mathbb{Z}$. Let $Z_{x\vec{y}} = (\triangleleft', e)$ with $e \in \mathbb{Z} \cup \{-\infty, +\infty\}$. To show \dagger_5 it then suffices to show $c - M \leq e$. This involves more arguments in the same spirit as in \dagger_2 case above, and we leave these technical details to the full version [2]. \blacktriangleleft

We turn to the second step of the proof and define an equivalence relation of finite index \sim_M on valuations. First, we define \sim_M on $\alpha, \beta \in \overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$ by $\alpha \sim_M \beta$ if $(\alpha \triangleleft c \iff \beta \triangleleft c)$ for all (\triangleleft, c) with $\triangleleft \in \{<, \leq\}$ and $c \in \{-\infty, \infty\} \cup \{d \in \mathbb{Z} \mid |d| \leq M\}$. In particular, if $\alpha \sim_M \beta$ then $(\alpha = -\infty \iff \beta = -\infty)$ and $(\alpha = \infty \iff \beta = \infty)$.

Next, for valuations $v_1, v_2 \in \mathbb{V}$, we define $v_1 \sim_M v_2$ by two conditions: $v_1(x) \sim_M v_2(x)$ and $v_1(x) - v_1(y) \sim_{2M} v_2(x) - v_2(y)$ for all clocks $x, y \in X$. Notice that we use $2M$ for differences of values. Clearly, \sim_M is an equivalence relation of finite index on valuations.

The next result relates the equivalence relation \sim_M and the simulation relation \leq_G when the finite constants used in the constraints are bounded by M . Recall from Section 5 the definition of the distance graph $\mathbb{G}_G(v)$ for the set of valuations $\uparrow_G v$.

► Lemma 27. *Let $v_1, v_2 \in \mathbb{V}$ be valuations with $v_1 \sim_M v_2$ and let G be a set of atomic constraints using constants in $\{-\infty, \infty\} \cup \{c \in \mathbb{Z} \mid |c| \leq M\}$. By replacing the weights $(\leq, v_1(x))$ (resp. $(\leq, -v_1(x))$) by $(\leq, v_2(x))$ (resp. $(\leq, -v_2(x))$) in the graph $\mathbb{G}_G(v_1)$ we obtain the graph $\mathbb{G}_G(v_2)$.*

Next we state the central lemma that says that $\downarrow_G Z$ is a union of \sim_M equivalence classes.

► **Lemma 28.** *Let $v_1, v_2 \in \mathbb{V}$ be valuations with $v_1 \sim_M v_2$ and let G be a set of atomic constraints using constants in $\{-\infty, \infty\} \cup \{c \in \mathbb{Z} \mid |c| \leq M\}$. Let Z be a zone with a canonical distance graph $\mathbb{G}(Z)$ satisfying (\dagger) . Then, $v_1 \in \downarrow_G Z$ iff $v_2 \in \downarrow_G Z$.*

Proof sketch. We will consider two valuations v_1, v_2 such that $v_1 \sim_M v_2$ and $v_1 \in \downarrow_G Z$ and show that the assumption that $v_2 \notin \downarrow_G Z$ leads to a contradiction. Roughly the proof proceeds as follows. Firstly, $v_2 \notin \downarrow_G Z$ implies that $\uparrow_G v_2 \cap Z = \emptyset$. Further, recall from Lemma 22 that if $\uparrow_G v_2 \cap Z = \emptyset$, then we can find a negative cycle C_2 using one edge from $\mathbb{G}(Z)$ and one or two edges from $\mathbb{G}_G(v_2)$. From Lemma 27, there exists a cycle C_1 involving the corresponding edges from $\mathbb{G}(Z)$ and $\mathbb{G}_G(v_1)$. Since $\uparrow_G v_1 \cap Z \neq \emptyset$, we know that C_1 is not negative. We will show that this implies that the C_2 (which was a witness for emptiness of $\uparrow_G v_2 \cap Z$) also cannot be negative, which leads to a contradiction. The central part of the proof involves a careful case analysis of the various forms that the cycle C_2 can take, using different \dagger conditions. We detail two cases here. The remaining eight can be found in the full version [2].

- Cycle $C_2 = 0 \xrightarrow{(\leq, v_2(\vec{x}))} \vec{x} \xrightarrow{Z_{\vec{x}0}} 0$. We have $C_1 = 0 \xrightarrow{(\leq, v_1(\vec{x}))} \vec{x} \xrightarrow{Z_{\vec{x}0}} 0$.
Let $Z_{\vec{x}0} = (\triangleleft, c)$. Since C_2 is negative, we deduce that $c \neq \infty$. From (\dagger_3) , we infer $Z_{\vec{x}0} \leq (\leq, M)$ and $0 \leq c \leq M$.
Since C_1 is not negative, we get $(\leq, 0) \leq (\triangleleft, c + v_1(\vec{x}))$, which is equivalent to $-c \leq v_1(\vec{x})$.
Using $v_1 \sim_M v_2$ and $0 \leq c \leq M$ we deduce that $-c \leq v_2(\vec{x})$. This is equivalent to $(\leq, 0) \leq (\triangleleft, c + v_2(\vec{x}))$, a contradiction with C_2 being a negative cycle.
- Cycle $C_2 = 0 \xrightarrow{Z_{0\vec{x}}} \vec{x} \xrightarrow{(\leq, -v_2(\vec{x}))} 0$. We have $C_1 = 0 \xrightarrow{Z_{0\vec{x}}} \vec{x} \xrightarrow{(\leq, -v_1(\vec{x}))} 0$.
Let $Z_{0\vec{x}} = (\triangleleft, c)$. Since C_2 is negative, we deduce that $-v_2(\vec{x}) \neq \infty$. Using $v_1 \sim_M v_2$, we infer $-v_1(\vec{x}) \neq \infty$. Since C_1 is not negative, we get $Z_{0\vec{x}} \neq (\leq, -\infty)$. From (\dagger_6) , we infer $(\triangleleft, -M) \leq Z_{0\vec{x}}$ and $-M \leq c \leq 0$.
Since C_1 is not a negative cycle, we get $(\leq, 0) \leq (\triangleleft, c - v_1(\vec{x}))$, which is equivalent to $v_1(\vec{x}) \leq c$. Using $v_1 \sim_M v_2$ and $-M \leq c \leq 0$, we deduce that $v_2(\vec{x}) \leq c$. This is equivalent to $(\leq, 0) \leq (\triangleleft, c - v_2(\vec{x}))$, a contradiction with C_2 being a negative cycle. ◀

Finally, from Lemmas 26 and 28, we obtain our main theorem of the section.

► **Theorem 29.** *The simulation relation $\preceq_{\mathcal{A}}$ is finite.*

Proof. Let $(q, Z_0), (q, Z_1), (q, Z_2), \dots$ be an infinite sequence of *reachable* nodes in $\text{EZG}(\mathcal{A})$. By Lemma 26, for all i , the distance graph $\mathbb{G}(Z_i)$ in canonical form satisfies conditions (\dagger) .

The atomic constraints in $G = \mathcal{G}(q)$ use constants in $\{-\infty, \infty\} \cup \{c \in \mathbb{Z} \mid |c| \leq M\}$. From Lemma 28 we deduce that for all i , $\downarrow_G Z_i$ is a union of \sim_M -classes. Since \sim_M is of finite index, there are only finitely many unions of \sim_M -classes. Therefore, we find $i < j$ with $\downarrow_G Z_i = \downarrow_G Z_j$, which implies $Z_j \preceq_G Z_i$. ◀

Note that the number of enumerated zones is bounded by 2^r , where r is the number of regions. This is similar to the exponential blow up that happens in normal timed automata. Indeed, despite this blow up the interest in zone algorithms is that, at least in the timed setting, they work significantly better in practice. We hope the above zone-based approach for ECA will also pave the way for fast implementations for ECA.

7 Conclusion

In this paper, we propose a simulation based approach for reachability in ECAs. The main difficulty and difference from timed automata is the use of prophecy clocks and undefined values. We believe that the crux of our work has been in identifying the new representation

for prophecy clocks and undefined values. With this as the starting point, we have been able to adapt the zone graph computation and the \mathcal{G} -simulation technique to the ECA setting. This process required us to closely study the mechanics of prophecy clocks in the zone computations and we discovered this surprising property that prophecy clocks by themselves do not create a problem for finiteness.

The final reachability algorithm looks almost identical to the timed automata counterpart and hence provides a mechanism to transfer timed automata technology to the ECA setting. The performance benefits observed for the LU and \mathcal{G} -simulation-based reachability procedures for timed automata encourages us to believe that an implementation of our algorithm would also yield good results, thereby providing a way to efficiently check event-clock specifications on timed automata models. We also hope that our framework can be extended to other verification problems, like liveness and to extended models like ECA with diagonal constraints that have been studied in the context of timeline based planning [11, 12].

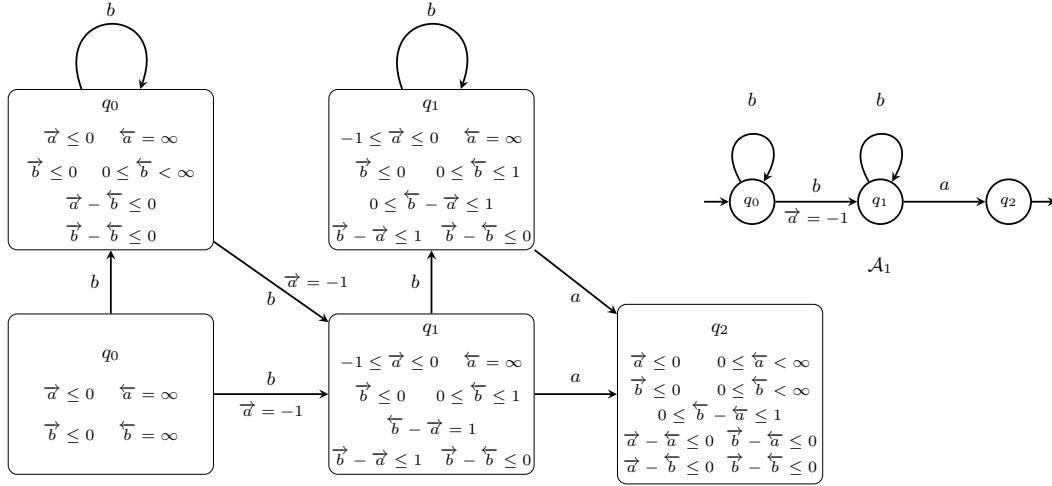
References

- 1 S. Akshay, Benedikt Bollig, and Paul Gastin. Event clock message passing automata: a logical characterization and an emptiness checking algorithm. *Formal Methods Syst. Des.*, 42(3):262–300, 2013.
- 2 S. Akshay, Paul Gastin, R. Govind, and B. Srivathsan. Simulations for event-clock automata. *CoRR*, abs/2207.02633, 2022.
- 3 S. Akshay, Paul Gastin, and Karthik R. Prakash. Fast zone-based algorithms for reachability in pushdown timed automata. In *CAV (1)*, volume 12759 of *Lecture Notes in Computer Science*, pages 619–642. Springer, 2021.
- 4 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.
- 5 Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theor. Comput. Sci.*, 211(1-2):253–273, 1999.
- 6 Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Hakansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. UPPAAL 4.0. In *QEST*, pages 125–126. IEEE Computer Society, 2006.
- 7 Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In *ACPN 2003*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003.
- 8 Patricia Bouyer. Forward analysis of updatable timed automata. *Formal Methods Syst. Des.*, 24(3):281–320, 2004.
- 9 Patricia Bouyer, Maximilien Colange, and Nicolas Markey. Symbolic optimal reachability in weighted timed automata. In *CAV (1)*, volume 9779 of *Lecture Notes in Computer Science*, pages 513–530. Springer, 2016.
- 10 Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis, and Sergio Yovine. Kronos: A model-checking tool for real-time systems. In *CAV*, volume 1427 of *Lecture Notes in Computer Science*, pages 546–550. Springer, 1998.
- 11 Laura Bozzelli, Angelo Montanari, and Adriano Peron. Taming the complexity of timeline-based planning over dense temporal domains. In *FSTTCS*, volume 150 of *LIPICs*, pages 34:1–34:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.
- 12 Laura Bozzelli, Angelo Montanari, and Adriano Peron. Complexity issues for timeline-based planning over dense time under future and minimal semantics. *Theor. Comput. Sci.*, 901:87–113, 2022.
- 13 Conrado Daws and Stavros Tripakis. Model checking of real-time reachability properties using abstractions. In *TACAS*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 1998.

- 14 David L. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 197–212. Springer, 1989.
- 15 Deepak D’Souza and Nicolas Tabareau. On timed automata with input-determined guards. In *FORMATS/FTRTFT*, volume 3253 of *Lecture Notes in Computer Science*, pages 68–83. Springer, 2004.
- 16 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Reachability in timed automata with diagonal constraints. In *CONCUR*, volume 118 of *LIPICs*, pages 28:1–28:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- 17 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Fast algorithms for handling diagonal constraints in timed automata. In *CAV (1)*, volume 11561 of *Lecture Notes in Computer Science*, pages 41–59. Springer, 2019.
- 18 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Reachability for updatable timed automata made faster and more effective. In *FSTTCS*, volume 182 of *LIPICs*, pages 47:1–47:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.
- 19 Gilles Geeraerts, Jean-François Raskin, and Nathalie Sznajder. Event clock automata: From theory to practice. In *FORMATS*, volume 6919 of *Lecture Notes in Computer Science*, pages 209–224. Springer, 2011.
- 20 Gilles Geeraerts, Jean-François Raskin, and Nathalie Sznajder. On regions and zones for event-clock automata. *Formal Methods Syst. Des.*, 45(3):330–380, 2014.
- 21 Frédéric Herbreteau and Gerald Point. TChecker. <https://github.com/fredher/tchecker>, v0.2 – April 2019.
- 22 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better abstractions for timed automata. In *LICS*, pages 375–384. IEEE Computer Society, 2012.
- 23 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Lazy abstractions for timed automata. In *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 990–1005. Springer, 2013.
- 24 Gijs Kant, Alfons Laarman, Jeroen Meijer, Jaco van de Pol, Stefan Blom, and Tom van Dijk. LTSmin: High-performance language-independent model checking. In *TACAS*, volume 9035 of *Lecture Notes in Computer Science*, pages 692–707. Springer, 2015.
- 25 Sebastian Kupferschmid, Martin Wehrle, Bernhard Nebel, and Andreas Podelski. Faster than UPPAAL? In *CAV*, volume 5123 of *Lecture Notes in Computer Science*, pages 552–555. Springer, 2008.
- 26 Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *STTT*, 1(1-2):134–152, 1997.
- 27 Jean-François Raskin and Pierre-Yves Schobbens. The logic of event clocks – decidability, complexity and expressiveness. *J. Autom. Lang. Comb.*, 4(3):247–282, 1999.
- 28 Victor Roussanaly, Ocan Sankur, and Nicolas Markey. Abstraction refinement algorithms for timed automata. In *CAV (1)*, volume 11561 of *Lecture Notes in Computer Science*, pages 22–40. Springer, 2019.
- 29 Jun Sun, Yang Liu, Jin Song Dong, and Jun Pang. PAT: towards flexible verification under fairness. In *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 709–714. Springer, 2009.
- 30 Tamás Tóth, Ákos Hajdu, András Vörös, Zoltán Micskei, and István Majzik. Theta: A framework for abstraction refinement-based model checking. In *FMCAD*, pages 176–179. IEEE, 2017.
- 31 Farn Wang. REDLIB for the formal verification of embedded systems. In *ISoLA*, pages 341–346. IEEE Computer Society, 2006.
- 32 Jianhua Zhao, Xuandong Li, and Guoliang Zheng. A quadratic-time DBM-based successor algorithm for checking timed automata. *Inf. Process. Lett.*, 96(3):101–105, 2005.

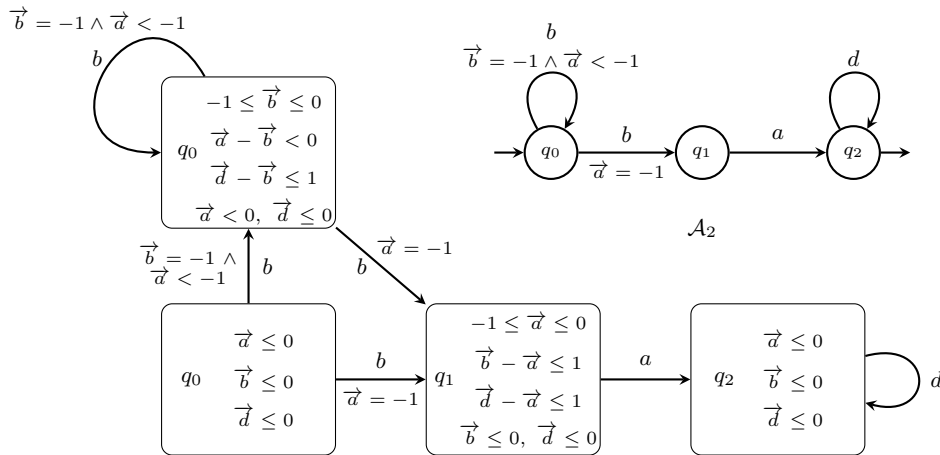
A Appendix for Section 3

In Figure 3, we give the event zone graph of the event-clock automaton \mathcal{A}_1 that recognizes the language $\{b^n a \mid n \geq 1\}$ such that there exists some b which occurs exactly one time unit before a .



■ **Figure 3** An event-clock automaton and its event zone graph. Missing lower bounds are of the form $-\infty \leq x - y$ and missing upper bounds are of the form $x - y \leq \infty$ (including $y = 0$).

Further, Geeraerts et al. [19, 20] showed that there exists no finite time abstract bisimulation relation for the event predicting automaton (EPA) \mathcal{A}_2 given in Figure 4. Figure 4 also depicts the event zone graph of \mathcal{A}_2 . Note that, since this is an event *predicting* automaton, there are no history clocks. It is easy to see that there are only finitely many distinct constraints involving the prophecy clocks.



■ **Figure 4** Event *predicting* automaton for which there exists no finite time abstract bisimulation and its event zone graph. Missing lower bounds are of the form $-\infty \leq x - y$ and missing upper bounds are of the form $x - y \leq \infty$ (including $y = 0$).


History-Deterministic Timed Automata

Thomas A. Henzinger 

IST Austria, Klosterneuburg, Austria

Karoliina Lehtinen 

CNRS, Aix-Marseille University, University of Toulon, LIS, France

Patrick Totzke 

University of Liverpool, UK

Abstract

We explore the notion of history-determinism in the context of timed automata (TA). History-deterministic automata are those in which nondeterminism can be resolved on the fly, based on the run constructed thus far. History-determinism is a robust property that admits different game-based characterisations, and history-deterministic specifications allow for game-based verification without an expensive determinization step.

We show yet another characterisation of history-determinism in terms of fair simulation, at the general level of labelled transition systems: a system is history-deterministic precisely if and only if it fairly simulates all language smaller systems.

For timed automata over infinite timed words it is known that universality is undecidable for Büchi TA. We show that for history-deterministic TA with arbitrary parity acceptance, timed universality, inclusion, and synthesis all remain decidable and are EXPTIME-complete.

For the subclass of TA with safety or reachability acceptance, we show that checking whether such an automaton is history-deterministic is decidable (in EXPTIME), and history-deterministic TA with safety acceptance are effectively determinizable without introducing new automata states.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Timed Automata, History-determinism, Good-for-games, fair simulation, synthesis

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.14

Funding *Thomas A. Henzinger*: This work was supported in part by the ERC-2020-AdG 101020093. *Patrick Totzke*: acknowledges support from the EPSRC, project no. EP/V025848/1.

1 Introduction

Automata offer paradigmatic formalisms both for specifying and for modelling discrete transition systems, *i.e.* for providing descriptive as well as executable definitions of formal languages. Given a finite or infinite word, an automaton specifies whether or not the word belongs to the defined language. Deterministic automata are executable, because the word can be processed left-to-right, with each transition of the automaton determined by the current input letter. Descriptive automata allow the powerful concept of nondeterminism, which yields more succinct or even more expressive specifications.

The notion of *history-determinism* lies between determinism and nondeterminism. History-deterministic automata are still executable, provided the execution engine is permitted to keep a record of all past inputs. Formally, a strategy r (*a.k.a.* “resolver”) is a function from finite prefix runs to transitions that suggests for each input word w a specific run $r^*(w)$ of the automaton over w , namely, the run that results from having the function r determine, after each input letter, the next transition based on the prefix of the word processed so far. An automaton is *history-deterministic* if there exists a resolver r so that for every input word w , the automaton has an accepting run over w iff the specific run $r^*(w)$ is accepting.



© Thomas A. Henzinger, Karoliina Lehtinen, and Patrick Totzke;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 14; pp. 14:1–14:21

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The concept of history-determinism was first identified in [21], where it was noted that for solving graph games, it is not necessary to determinize history-deterministic specifications of ω -regular winning conditions. For this reason, history-deterministic automata were called “good-for-games”. The term “history-determinism” was first used by [12]. The concept itself has since been referred to as both “history-determinism” and “good-for-gameness.” Since [9] recently showed that, in a general context of quantitative automata, the two notions do not always coincide (specifically: for certain quantitative winning conditions, history-determinism implies the “good-for-games” property of an automaton, but not vice versa), we follow their more nuanced terminology and use the term “history-determinism” to denote the existence of a resolver and “good-for-games” for automata that preserve the winner of games under composition, as required for solving games without determinization.

There is also a tight link between a variant of the Church synthesis problem, called *good-enough synthesis* [2], and deciding history-determinism. Church synthesis asks whether a system can guarantee that its interaction with an uncontrollable environment satisfies a specification language for all possible environment behaviours. This model assumes that the environment is hostile and will, if possible, sabotage the system’s efforts. This pessimistic view can be counter-productive. In the canonical example of a coffee machine, if the users (the environment) do not fill in the water container, the machine will fail to produce coffee. Church synthesis would declare the problem unrealisable: the machine may not produce coffee for all environment behaviours. In the good-enough synthesis problem, on the other hand, such failures are acceptable, and we can still return an implementation that produces coffee (satisfies the specification) whenever the environment behaves in a way that allows the desired behaviour (fills in the water container). Deciding the good-enough synthesis problem for a deterministic automaton is polynomially equivalent to deciding whether a nondeterministic automaton of the same type is history-deterministic [15, 9, 17]. The decidability and complexity of checking history-determinism is therefore particularly interesting.

In this paper, we study, for the first time, history-determinism in the context of *timed* automata. In a timed word, letters alternate with time delays, which are nonnegative real numbers. The resolver gets to look not only at all past input letters, but also at all past time delays, to suggest the next transition. We consider timed automata over infinite timed words with standard ω -regular acceptance conditions [3]. For the results of this paper, it does not matter whether or not the sum of all time delays provided by an infinite input word is required to diverge.

Our results can be classified into two parts. The first part of our results applies to all timed automata, and sometimes more generally, to all labelled transition systems. In this part we are concerned with solving the quintessential verification problem for timed systems, namely *timed language inclusion*, in the special case of history-deterministic (*i.e.* executable) specifications. Since universality is undecidable for general timed automata, so is the timed language-inclusion problem for nondeterministic specifications [3]. This is the reason why much previous work in timed verification has focused on identifying determinizable subclasses of timed automata, such as event-clock automata [4], and on studying deterministic extensions of the timed-automaton model, such as deterministic two-way timed automata [5]. Determinizable specifications can be complemented, thus supporting the *complementation-based* approach to language inclusion: in order to check if every word accepted by the implementation A is also accepted by the specification B , first determinize and complement B , and then check the intersection with A for emptiness. We show that the history-determinism of specifications suffices for deciding timed language inclusion, which demonstrates that determinizability is not required. More precisely, we prove that if A is a timed automaton and B is a history-deterministic timed automaton, it can be decided in

EXPTIME if every timed word accepted by A is also accepted by B (Corollary 18).

In contrast to the traditional complementation-based approach to language inclusion, the history-deterministic approach is *game-based*. Like the complementation-based approach, the game-based approach is best formulated in the generic setting of labelled transition systems with acceptance conditions, so-called *fair LTS*. The acceptance condition of a fair LTS declares a subset of the infinite runs of the LTS to be fair (a special case is *safety* acceptance, which declares all infinite runs to be fair). Given two fair LTS A and B , the language of A is included in the language of B if for every fair run of A there is a fair run of B over the same (infinite) word. A sufficient condition for the language inclusion between A and B is the existence of a fair simulation relation between the states of A and the states of B , or equivalently, the existence of a winning strategy for player p_B in the following 2-player *fair simulation game*: (i) every transition chosen by player p_A on the state-transition graph A can be matched by a transition chosen by player p_B on the state-transition graph B with the same label (letter or time delay), and (ii) if the infinite sequence of transitions chosen by p_A produce a fair run of A , then the matching transitions chosen by p_B produce a fair run of B [20]. Solving the fair simulation game is often simpler than checking language inclusion; it may be polynomial where language inclusion is not (*e.g.* in the case of finite safety or Büchi automata), or decidable where language inclusion is not (*e.g.* in the case of timed safety or Büchi automata [28]).

We show that for all fair LTS A and all history-deterministic fair LTS B , the condition that the language of A is included in the language of B is equivalent to the condition that A is fairly simulated by B . This observation reduces the language inclusion problem for history-deterministic specifications to the problem of solving a fair simulation game between implementation and specification. The solution of fair simulation games depends on the complexity of the acceptance conditions of A and B , but is often simpler than the complementation of B , and fair simulation games can be solvable even in the case of specifications that cannot be complemented. In the concluding Section 7, we conjecture the existence of such a timed language. The game-based approach to checking language inclusion, which requires history-determinism, is therefore more general, and often more efficient, than the traditional complementation-based approach to checking language inclusion, which usually requires full determinization. Indeed, history-determinism is exactly the condition that allows the game-based approach to language inclusion: for a given fair LTS B , if it is the case that B can fairly simulate all fair LTS A whose language is included in the language of B , then B must be history-deterministic (Theorem 4).

More generally, turn-based timed games for which the winning condition is defined by a history-deterministic timed automaton are no harder to solve than those with deterministic winning conditions: the winner of such a timed game can be determined on the product of the (timed) arena with the automaton specifying the winning condition. We conjecture that this is the case also for the concurrent timed games of [13] (cf. Section 7). Timed games have also been defined for the synthesis of timed systems from timed I/O specifications. Again, we show that the synthesis game of [14] can be solved not only for I/O specifications that are given by deterministic timed automata, but more generally, for those given by history-deterministic timed automata (Theorem 20).

The second part of our results investigates the problem of deciding history-determinism for timed automata and the determinizability of history-deterministic timed automata. In this part, we have only partial results, namely results for timed safety and reachability automata. Timed safety automata, in particular, constitute an important class of specifications, as many interesting timed and untimed properties can be specified by timed safety automata if time is

required to diverge [18, 19]. We prove that for timed safety automata and timed reachability automata, it can be decided in EXPTIME if a given timed automaton is history-deterministic (Theorem 16). Checking history-determinism remains open for more general classes of timed automata, such as timed Büchi and coBüchi automata. We also show that every history-deterministic timed safety automaton can be determinized, without increasing the number of automaton states, but with an exponential increase in the number of transitions or length of guards (Theorem 9). While the question of determinizability is undecidable for nondeterministic timed reachability automata [16], it is open for history-deterministic timed reachability automata and for history-deterministic timed automata with more general acceptance conditions. Finally, we show that if a timed safety or reachability automaton is good-for-games (in the sense explained earlier), then the automaton must be history-deterministic (Theorem 23). This implication is open for more general classes of timed automata.

Related Work. The notion of history-determinism was introduced independently, with slightly different definitions, by Henzinger and Piterman [21] for solving games without determinization, by Colcombet [12] for cost-functions, and by Kupferman, Safra, and Vardi [24] for recognising derived tree languages of word automata. Initially, history-determinism was mostly studied in the ω -regular setting, where these different definitions all coincide [8]. For some coBüchi-recognisable languages, history-deterministic automata can be exponentially more succinct than any equivalent deterministic automaton [23], and for Büchi and coBüchi automata, history-determinism is decidable in polynomial time [6, 23]. For transition-based history-deterministic automata, minimisation is PTIME [1], while for state-based ones, it is NP-complete [27]. Recently, the notion has been extended to richer automata models, such as pushdown automata [25, 17] and quantitative automata [9, 10], where deterministic and nondeterministic models have different expressivity, and therefore, allowing a little bit of nondeterminism can, in addition to succinctness, also provide more expressivity.

Paper Structure. After defining preliminary notions we proceed to introduce history-determinism, and show a new, fair-simulation-based characterisation in Section 3. In Section 4 we demonstrate that history-deterministic TA with safety acceptance are determinizable, and in Section 5 that one can decide whether a given safety or reachability TA is history-deterministic. Section 6 considers questions concerning timed games, timed synthesis, and timed language inclusion and shows that history-determinism coincides with good-for-gameness for reachability and safety TA.

2 Preliminaries

Numbers, Words. Let \mathbb{N} and $\mathbb{R}_{\geq 0}$ denote the nonnegative integers and reals, respectively. For $c \in \mathbb{R}_{\geq 0}$ we write $\lfloor c \rfloor$ for its integer and $\text{fract}(c) \stackrel{\text{def}}{=} c - \lfloor c \rfloor$ for its fractional part.

An alphabet Σ is a nonempty set of letters. Σ_ε denotes $\Sigma \cup \{\varepsilon\}$. Σ^* and Σ^ω denote the sets of finite and infinite words over Σ , respectively and $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$ denotes their union. The empty word is denoted by ε , the length of a finite word v is denoted by $|v|$, and the n -th letter of a finite or infinite word is denoted by $w[n]$ (starting with $n = 0$).

Labelled Transition Systems, Languages, Fair Simulation. A *labelled transition system* (LTS) is a graph $S = (V, \Sigma, E)$ with set V of states and edges $E \subseteq V \times \Sigma \times V$, labelled by alphabet Σ . It is *deterministic* if for all $(s, a) \in V \times \Sigma$ there is at most one s' with $s \xrightarrow{a} s'$,

and *complete* if for all $(s, a) \in V \times \Sigma$ there is at least one s' with $s \xrightarrow{a} s'$. We henceforth consider only complete LTSs. Together with an *acceptance condition* $Acc \subseteq E^\omega$ this can be used to define languages over Σ as usual: a word $w = l_0 l_1 \dots \in \Sigma^\omega$ is accepted from s_0 if there is a path (also *run*) $\rho = s_0 \xrightarrow{l_1} s_1 \xrightarrow{l_2} s_2 \dots$ that is accepting, i.e., in Acc . The *language* $L(s_0) \subseteq \Sigma^\omega$ of an initial state $s_0 \in V$ consists of all words for which there exists an accepting run from s_0 . We will write $s \subseteq_L s'$ to denote language inclusion, meaning $L(s) \subseteq L(s')$. The acceptance condition Acc can be given by a parity condition but does not have to be. We consider in this paper especially reachability (does the run visit a state in a given target set $T \subseteq V$?) and safety conditions (does the run always stay in a “safe” region $F \subseteq V$?). An LTS together with an accepting condition is referred to as *fair* LTS [20].

Fair simulations [20] are characterised by simulation games on (a pair of) fair LTSs in which Player 1 stepwise produces a path from s , and Player 2 stepwise produces an equally labelled path from s' . Player 2 wins if she produces an accepting run whenever Player 1 does. That is, s is fairly simulated by s' (write $s \preceq s'$) iff Player 2 has a strategy in the simulation game so that, whenever the run produced by Player 1 is accepting then so is the run produced by Player 2 in response. Fair simulation $s \preceq s'$ implies language inclusion $L(s) \subseteq L(s')$ but not vice versa.

Timed Alphabets, Words, and LTSs. For any alphabet Σ let Σ_T denote the timed alphabet $\{(a, t) \mid a \in \Sigma, t \in \mathbb{R}_{\geq 0}\}$. A timed word is a finite or infinite word $w \in (\Sigma_T)^\infty$ consisting of letters in Σ paired with distinct non-negative non-decreasing real-valued timestamps. We will also write $d_0 a_0 d_1 a_1 \dots$ to denote a timed word $(a_i, t_i) \in \Sigma_T^\infty$ where $t_0 = d_0$ and $t_{i+1} = t_i + d_{i+1}$. Conversely, the duration and the timed word of any sequence in $(\Sigma \cup \mathbb{R})^\infty$ is given inductively as follows. For any $d \in \mathbb{R}_{\geq 0}$, $\tau \in \Sigma$, $\alpha \in (\Sigma \cup \mathbb{R})^*$, and $\beta \in (\Sigma \cup \mathbb{R})^\infty$ let $\text{duration}(\tau) \stackrel{\text{def}}{=} 0$; $\text{duration}(d) \stackrel{\text{def}}{=} d$; $\text{duration}(\alpha\beta) = \text{duration}(\alpha) + \text{duration}(\beta)$; $\text{tword}(\varepsilon) = \text{tword}(d) \stackrel{\text{def}}{=} \varepsilon$; $\text{tword}(a d) \stackrel{\text{def}}{=} \text{tword}(a)$; and $\text{tword}(\alpha\tau) \stackrel{\text{def}}{=} \text{tword}(\alpha)(\tau, \text{duration}(\alpha))$. An infinite timed word of finite duration is called a *zeno* word. Our results hold whether time must diverge (i.e., zeno words are not considered) or not; we note whenever time divergence affects proofs.

A *timed* LTS is one with edge labels in $\Sigma \uplus \mathbb{R}_{\geq 0}$, so that edges labelled by $\mathbb{R}_{\geq 0}$ (modelling the passing of time) satisfy the following conditions for all $\alpha, \beta, \gamma \in V$ and $d, d' \in \mathbb{R}_{\geq 0}$.

1. (Zero-delay): $\alpha \xrightarrow{0} \alpha$,
2. (Determinism): If $\alpha \xrightarrow{d} \beta \wedge \alpha \xrightarrow{d} \gamma$ then $\beta = \gamma$,
3. (Additivity): $\alpha \xrightarrow{d} \beta \xrightarrow{d'} \gamma$ then $\alpha \xrightarrow{d+d'} \gamma$.

The timed language $L(s) \subseteq \Sigma_T^\omega$ of a state s consists of all the timed words read along accepting runs $L(s) \stackrel{\text{def}}{=} \text{tword}(L(s))$. We write $L(S)$ for the timed language of the initial state of the LTS S .

Timed Automata. Timed automata are finite-state automata equipped with finitely many real-valued variables called *clocks*, whose transitions are guarded by constraints on clocks. Constraints on clocks $C = \{x, y, \dots\}$ are (in)equalities $x \triangleleft n$ where $x \in C$, $n \in \mathbb{N}$ and $\triangleleft \in \{\leq, <\}$. Let $\mathcal{B}(C)$ denote the set of Boolean combinations of clock constraints, called *guards*. A clock *valuation* $\nu \in \mathbb{R}^C$ assigns a value $\nu(x)$ to each clock $x \in C$. We write $\nu \models g$ if ν satisfies the guard g . A timed automaton (TA) $\mathcal{T} = (Q, \iota, C, \Delta, \Sigma, Acc)$ is given by

- Q a finite set of states including an initial state ι ;
- Σ an input alphabet;
- C a finite set of clocks;

- $\Delta \subseteq Q \times \mathcal{B}(C) \times \Sigma \times 2^C \times Q$ a set of transitions; each transition is associated with a guard, a letter, and a set of clocks to reset. A transition that reads letter $a \in \Sigma$ will be called an a -transition. We assume that for all $(s, \nu, a) \in Q \times \mathbb{R}_{\geq 0}^C \times \Sigma$ there is at least one transition $(s, g, a, r, s') \in \Delta$ so that ν satisfies g .
- $Acc \subseteq \Delta^\omega$ an acceptance condition.

Timed automata induce timed LTSs, and can thus be used to define timed languages, as follows. A *configuration* is a pair consisting of a control state and a clock valuation. These can evolve in two ways, as follows. For all configurations $(s, \nu) \in Q \times \mathbb{R}_{\geq 0}^C$,

- there is a *delay* step $(s, \nu) \xrightarrow{d} (s, \nu + d)$ for every $d \geq 0$, which increments all clocks by d .
- there is a *discrete* step $(s, \nu) \xrightarrow{\tau} (s', \nu')$ if $\tau = (s, g, a, r, s') \in \Delta$ is a transition so that ν satisfies g and $\nu' = \nu[r \rightarrow 0]$, that is, it maps r to 0 and agrees with ν on all other values.

Naturally, each delay d yields a unique successor configuration and $\nu \xrightarrow{d} \nu' \iff \nu \xrightarrow{d+d'} \nu'$ for any two $d, d' \geq 0$ and valuations ν, ν' . So this indeed induces a timed LTS.

Discrete steps, however, are a source of nondeterminism: a configuration may have several a -successors induced by different transitions whose guards are satisfied. \mathcal{T} is *deterministic* if its induced LTS is deterministic, which is the case iff for every state s , all transitions from s have mutually exclusive guards.

A path $\rho = (s_0, \nu_0) \xrightarrow{l_1} (s_1, \nu_1) \xrightarrow{l_2} (s_2, \nu_2) \dots$ is called *reduced* if it does not contain consecutive delay steps. It is a *run on* timed word $w \in (\Sigma_T)^\infty$ if $\text{tword}(l_1 l_2 \dots) = w$. The acceptance condition is lifted to the LTS as expected. Namely, a run is *accepting* if $\rho \in Acc$. This way, the *language* $L(s, \nu) \subseteq \Sigma_T^\omega$ of a configuration (s, ν) consists of all timed words for which there exists an accepting run from (s, ν) . The language of \mathcal{T} is $L(\mathcal{T}) \stackrel{\text{def}}{=} L((\iota, 0))$, the languages if the initial configuration with state ι and all clocks set to zero.

3 History-determinism

Informally, an automaton or LTS is history-deterministic if the non-determinism can be resolved on-the-fly, based only on the history of the word and run so far. We give two equivalent definitions, each being more convenient than the other for some technical developments.

► **Definition 1** (History-determinism). *A fair LTS $S = (V, \Sigma, E)$ is history-deterministic (from initial state $s_0 \in V$) if there is a resolver $r : E^* \times \Sigma \rightarrow E$ that maps every finite run and letter $a \in \Sigma$ to an a -labelled transition such that, for all words $w = a_0 a_1 \dots \in L(s_0)$ the run ρ defined inductively for $i > 0$ by $\rho_{i+1} \stackrel{\text{def}}{=} \rho_i r(\rho_i, a_{i+1})$, is an accepting run on w from s_0 .*

Equivalently (from [8] for ω -regular automata), a resolver corresponds exactly to a winning strategy for Player 2 in the following *letter game*.

► **Definition 2** (Letter game). *The letter game on a fair LTS $S = (V, \Sigma, E)$ with initial state $s_0 \in V$ is played between Players 1 and 2. At turn i :*

- Player 1 chooses a letter $a_i \in \Sigma$.
- Player 2 chooses an a_i labelled edge $\tau_i \in E$.

A play is a pair (w, ρ) where $w = a_0 a_1 \dots$ is an infinite word and $\rho = \tau_0 \tau_1 \dots$ is a run on w . A play is winning for Player 2 if either $w \notin L(s_0)$ or ρ is an accepting run on w from s_0 .

In these and other games we consider, strategies for both players are defined as usual, associating finite histories (runs) to valid player choices. Now winning strategies for Player 2 in the letter game exactly correspond to resolvers for S and vice-versa.

► **Proposition 3.** *Player 2 wins the letter game on a fair LTS S if and only if S is history-deterministic.*

While history-determinism is known to relate to fair simulation, in the sense that history-deterministic automata simulate deterministic ones for the same language [21], their relation has so far not been studied in more details. Below we show that history-determinacy can equivalently be characterised in terms of fair simulation.

► **Theorem 4.** *For every fair LTS S and initial state q the following are equivalent:*

1. S is history-deterministic.
2. For all complete fair LTS S' with initial state q' , $q' \subseteq_L q$ if and only if $q' \preceq q$.

Proof.

(1) \implies (2). Fair simulation $q \preceq q'$ trivially implies $q \subseteq_L q'$ by definition.

For the other implication, assume that $q \subseteq_L q'$. By assumption (1) there exists a resolver, i.e. a winning strategy in the letter game. Player 2 can win the fair simulation game by ignoring her opponent's configuration and moving according to this resolver. By the completeness assumption on S' , Player 1 can never propose a letter for which there is no successor in S' . So each player produces an infinite run on the same word w and the run produced by Player 2 is the same as that produced by the resolver in S' . If $w \in L(q)$ then it is in $L(q')$ and Player 2's run accepts. If $w \notin L(q)$ then Player 2 wins due to the fairness condition. In both cases she wins the fair simulation game and therefore $q \preceq q'$.

(2) \implies (1). If condition (2) holds for all complete fair LTSs then q can fairly simulate the one consisting of a single state with self-loops for all transitions of S whose acceptance condition contains exactly all accepting runs from q . Then the strategy for Player 2 in the fair simulation game can be used as a strategy in the letter game. ◀

4 Expressivity

In this section we show that history-deterministic timed automata with safety acceptance are determinizable. To do so, we show (in Lemma 8) that these automata have simple resolvers, which only depend on the equivalence class of the current clock configuration with respect to the region abstraction. That is to say, the resolver only needs to know the integer part of clock values (up to the maximal value that appears in clock constraints) and the ordering of their fractional parts. We can then use such a simple resolver to determinize the automaton by adding guards that restrict transitions so that the automaton can only take one transition per region, as dictated by the resolver.

The following is the standard definition of regions (cf. [3], def. 4.3).

► **Definition 5** (Region abstraction). *Let $\mathcal{T} = (Q, \iota, C, \Delta, \Sigma, Acc)$ be a timed automaton and for any clock $x \in C$ let c_x denote the largest constant in any clock constraint involving x . Two valuations $\nu, \nu' \in \mathbb{R}_{\geq 0}^C$ are (region) equivalent (write $\nu \sim \nu'$) if all of the following hold.*

1. For all $x \in C$ either $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ or both $\nu(x)$ and $\nu'(x)$ are greater than c_x .
2. For all $x, y \in C$ with $\nu(x) \leq c_x$ and $\nu(y) \leq c_y$, $\text{fract}(\nu(x)) \leq \text{fract}(\nu(y))$ iff $\text{fract}(\nu'(x)) \leq \text{fract}(\nu'(y))$.
3. For all $x \in C$ with $\nu(x) \leq c_x$, $\text{fract}(\nu(x)) = 0$ iff $\text{fract}(\nu'(x)) = 0$.

Two configurations (q, ν) and (q', ν') are (region) equivalent, write $(q, \nu) \sim (q', \nu')$, if $q = q'$ and $\nu \sim \nu'$.

► **Definition 6** (Run-trees). *A run-tree on a timed word $u = (a_0, t_0)(a_1, t_1) \dots$ from TA configuration (s_0, ν_0) is a tree where nodes are labelled by configurations, and edges by transitions such that*

1. The labels along every branch form a run on u from (s_0, ν_0)
2. It is complete wrt. discrete steps: suppose the path leading towards some node is labelled by a run ρ which reads $\text{tword}(\rho) = (a_0, t_0) \dots (a_i, t_i)$, ends in a configuration (s, ν) , and has $\text{duration}(\rho) = t_{i+1}$. Then for every transition $\tau = (s, g, a_{i+1}, r, s') \in \Delta$ with $\nu \models g$ and so that $(s, \nu) \xrightarrow{\tau} (s', \nu')$, there is a τ -labelled edge to a new node labelled by (s', ν') . A run-tree is reduced if all its branches are. That is, there are no consecutive delay steps.

Notice that for every initial configuration and timed word, there is a unique reduced run-tree, all of whose branches are runs on the word (since we have no deadlocks), and vice versa, all reduced runs on the word appear as branches on the run-tree.

We extend the region equivalence from configurations to run-trees in the natural fashion: two run-trees are equivalent if they are isomorphic and all corresponding configurations are equivalent. That is, they can differ only in fractional clock values and the duration of delays.

The following is our key technical lemma.

► **Lemma 7.** *Consider two region equivalent configurations $(s, \nu) \sim (s', \nu')$.*

For every timed word u there is a timed word u' so that the reduced run-tree on u from (s, ν) is equivalent to the reduced run-tree on u' from (s', ν') .

Proof sketch. It suffices to show that for some (not necessarily reduced) run-tree on u from (s, ν) there exists some equivalent run-tree from (s', ν') as this implies the claim by collapsing all consecutive delay steps and thus producing the reduced tree on both sides.

We proceed by stepwise uncovering a suitable run-tree from (s, ν) for ever longer prefixes of u and constructing a corresponding equivalent run-tree from (s', ν') . The intermediate finite trees we build have the property that all branches have the same duration. In each round we extend all current leafs, in both trees, either by

1. all possible non-deterministic successors (for the letter prescribed by the word u), in case the duration of the branch is already equal to the next time-stamp in u , or
2. one successor configuration due to a delay, which must be *the same on all leafs*.

For the second case, the delays used to extend the two trees need not be the same because we only want to preserve region equivalence. Also, the delay chosen for the tree rooted in (s, ν) need not follow the timestamps in u but can be shorter, meaning the run-tree may not be reduced.. The difficulty lies in systematically choosing the delays to ensure that the two trees remain equivalent and secondly, that in the limit this procedure generates a run-tree on the whole word u from (s, ν) . Together this implies the existence of a corresponding word u' and a run-tree from (s', ν') .

To this end we propose a stronger invariant, namely that the relative orderings of the fractional values *in all leafs are the same on both sides*. The delays will be chosen in such a way as to always increase the maximal fractional clock value among all leafs to the next higher integer. Due to space constraints full details are deferred to Appendix A. ◀

We are now ready to show that history-deterministic TA with safety acceptance have simple resolvers based on the region abstraction.

► **Lemma 8.** *Every history-deterministic TA with safety acceptance has a resolver r that bases its decision only on the current letter and region. That is, for any letter $a \in \Sigma$ and any two finite runs $(\iota, 0) \xrightarrow{\rho} (s, \nu)$ and $(\iota, 0) \xrightarrow{\rho'} (s', \nu')$ consistent with r and so that $(s, \nu) \sim (s', \nu')$, it holds that $r(\rho, a) = r(\rho', a)$.*

Proof. Let r be a resolver for a history-deterministic safety TA \mathcal{T} .

We now build a resolver that only depends on the region of the current configuration. To do so, we choose a representative configuration within each region, which will determine the choice of the resolver for the whole region: For every region $R \in [Q \times \mathbb{R}_{\geq 0}^C]_{\sim}$, consider the configurations that are reached by at least one r -consistent run, and mark one of them m_R , if at least one exists, along with one r -consistent run ρ_R leading to the configuration m_R .

Let r' be the aspiring resolver that, when reading a letter a , considers the region R of the current configuration, and follows what r does when reading a after the marked r -consistent run ρ_R . We set $r'(\rho, a) \stackrel{\text{def}}{=} r(\rho_R, a)$ where R is the final region of the prefix-run ρ . Note that r' is well defined since it always follows transitions consistent with some r -consistent run and can therefore only visit marked regions.

We claim that r' is indeed a resolver. Towards a contradiction, assume that it is not a resolver, that is, there is some word $w \in L(\mathcal{T})$ for which r' builds a rejecting run. As \mathcal{T} is a safety automaton, we can consider the last configuration (s, ν) along this run from which the remaining suffix au of w can be accepted¹.

Suppose that ρ is the prefix of the run built by r' on w , which ends in (s, ν) and let $\tau = r'(\rho, a)$ be the a -transition chosen by r' . We know that τ leads from (s, ν) to some configuration (s', ν') from where u is not accepted. By definition of r' , there must be a marked configuration $m_R \sim (s, \nu)$ reached by some run ρ_R from which r chooses the same a -transition τ . By Lemma 7 there must be a word au' so that the run-tree on au from (s, ν) is equivalent to that on au' from m_R . This means that $au' \in L(m_R)$ and, as r is a resolver, there must be an accepting run that begins with a step $(m_R) \xrightarrow{\tau} (m'_R)$. We derive that u also has an accepting run from (q, ν) that begins with τ , contradicting the assumption that (q, ν) is the last position on the run r' built on w so that its suffix can be accepted. Therefore, r' is indeed a resolver. ◀

We can now use the region-based solver to determinize history-deterministic safety TA.

► **Theorem 9.** *Every history-deterministic safety TA is equivalent to a deterministic TA.*

Proof. Consider a history-deterministic TA $\mathcal{T} = (Q, \iota, C, \Delta, \Sigma, Acc)$, with a region-based resolver (as in Lemma 8) r , and let R be the region graph of \mathcal{T} . Define $\mathcal{T}' = (Q, \iota, C, \Delta', \Sigma, Acc)$ where $(q, g \wedge z, a, X, q') \in \Delta'$ for z a guard defining a region of R , that is, a guard that is satisfied exactly by valuations in R , if $(q, g, a, X, q') \in \Delta$ is the transition chosen by r in the region defined by the guard z . In other words, \mathcal{T}' is \mathcal{T} with duplicated transitions guarded so that a transition can only be taken from a region from which r chooses that transition. Observe that \mathcal{T}' is deterministic: the guards describing regions are mutually exclusive, therefore the guards of any two transitions from the same state over the same letter have mutually exclusive guards.

As runs of \mathcal{T}' corresponds to a run of \mathcal{T} with added guards, $L(\mathcal{T}') \subseteq L(\mathcal{T})$. Conversely, if $w \in L(\mathcal{T})$, then its accepting run consistent with r is also an accepting run in \mathcal{T}' , since each transition along this run, being chosen by r , is taken at a configuration that satisfies the additional guards in \mathcal{T}' . We can therefore conclude that $L(\mathcal{T}) = L(\mathcal{T}')$. ◀

¹ The fact that a rejecting run produced by a non-resolver must ultimately reach a configuration that cannot accept the remaining word also holds for TAs over finite words. However, this is *not* the case for reachability acceptance, which is why we only state the claim for safety here. Still, we conjecture that history-deterministic TA with reachability acceptance admit region-based resolvers.

While this determinization procedure preserves the state-space of the automaton, it multiplies the number of transitions (or the size of guards) by the size of the region abstraction. Then, while history-deterministic safety TA are no more expressive than deterministic ones, they could potentially be exponentially more succinct, when counting transitions and guards.

5 Deciding History-determinism

Recall the letter game characterisation of history-determinism: Player 1 plays timed letters and Player 2 responds with transitions. Player 2 wins if either the word is not in the language of the automaton, or her run is accepting. As TA are not closed under complement, it isn't clear how to solve this game. Bagnol and Kuperberg [6] introduced *token games*, which are easier to solve, but which coincide with the letter game for some types of automata, in particular for Büchi [6], coBüchi [7] and some quantitative automata [10].

In the k -token game, in addition to providing letters, Player 1 also builds k runs, of which at least one should be accepting. The fewer runs Player 1 is allowed to use, the more information he gives Player 2 about the word he will play. We show that the 1 and 2-token games characterize history-determinism for fair LTSs with safety and reachability acceptance.

► **Definition 10** (k -token game [6]). *Given a fair LTS $S = (V, \Sigma, E)$ with initial state $s_0 \in V$ and an integer $k > 0$, the game $G_k(S)$ proceeds in rounds. At each round i :*

- *Player 1 plays a letter $a_i \in \Sigma$*
- *Player 2 plays a transition τ_i in E*
- *Player 1 plays transitions $\tau_{1,i}, \tau_{2,i} \dots \tau_{k,i}$ in S*

This way, Player 1 chooses an infinite word $w = a_0 a_1 \dots$ and exactly k runs $\rho_i = \tau_{i,0} \tau_{i,1} \tau_{i,2} \dots$ for $1 \leq i \leq k$, and Player 2 chooses a run $\rho = \tau_0 \tau_1 \dots$. The play is winning for Player 1 if some ρ_j is an accepting run over $t_0 a_0 \dots$ from s_0 but ρ is not. Else it is winning for Player 2.

We write $G_k(\mathcal{T})$ to mean the k -token game on the LTS induced by \mathcal{T} .

► **Remark 11.** $G_k(S)$ and the letter game are determined for any k and fair LTS S for any Borel-definable acceptance condition [26]. In particular, the letter game is determined for both safety and reachability TA. Indeed, the winning condition for Player 2 is a disjunction of the complement of $L(\mathcal{B})$ and of the acceptance condition of \mathcal{B} . Then, as long as $L(\mathcal{B})$ is Borel, by the closure of Borel sets under complementation and disjunction, the letter-game is Borel, and therefore determined, following Martin's Theorem [26]. If time is not required to diverge, then reachability timed languages and safety timed languages are clearly Borel. Since words in which time diverges are also Borel (they can be seen as the countable intersection of words where time reaches each unit time), this remains the case when we require divergence.

The next lemma was first stated for finite [6], then for quantitative automata [10]. The same proof works for all (generally infinite) fair LTSs, and is given again in Appendix B.

► **Lemma 12.** *Given an fair LTS S , if Player 2 wins $G_2(S)$ then she wins $G_k(S)$ for all k .*

$G_1(S)$ was shown to characterise history-determinism for a number of quantitative automata in [10]. In Appendix B we show, using similar proof techniques, that this is also the case for all safety LTSs. The key observation is that for Player 2 to win the letter game, it suffices that she avoids mistakes. We then show that a winning strategy for her in $G_1(S)$ can be used to build such a strategy.

► **Lemma 13.** *Given a fair LTS S with a safety acceptance condition, Player 2 wins $G_1(S)$ if and only if S is history-deterministic.*

This argument does not work for reachability TA: it is no longer enough for Player 2 to avoid bad moves to win; she needs to also guarantee that she will actually reach a final state. Here, we characterise history-determinism with the 2-token game. However, our proof requires finite branching in Player 2's choices, so we can not state it for LTSs in general.

► **Lemma 14.** *Given a finitely branching fair LTS S with a reachability acceptance condition, Player 2 wins $G_2(S)$ if and only if S is history-deterministic.*

Proof. If Player 2 wins in the letter game, she wins in $G_2(S)$ by ignoring Player 1's tokens.

Else, since the letter game is determined (Remark 11), Player 1 wins in the letter game on S with a strategy σ . All plays that agree with σ must eventually play a good prefix, that is, a prefix of a timed word of which either all continuations are in $L(S)$ if time is not required to diverge, or all non-zeno continuations are in $L(S)$ if time is required to diverge. At each turn Player 2 has only a finite number of enabled transitions to choose from, because S is finitely branching. Therefore the strategy-tree for σ is finitely branching and by König's lemma, there is a bound k such that any play that agrees with σ has played a good prefix after k steps.

We now argue that Player 1 wins in $G_{k'}(S)$ for a large enough k' . Let k' be larger than the number of distinct run prefixes of length k on any word of length k played by σ (that is, at most b^k where b is the branching degree of S). Then, in $G_{k'}(S)$, Player 1 wins by using the following strategy: he plays the letters according to σ and Player 2's moves and moves his k' tokens along all possible run prefixes for the first k moves, and then chooses transitions arbitrarily. Since after k steps σ guarantees that he has played a good prefix, at least one of his runs built in this manner is accepting.

This strategy is winning: indeed, if Player 2 could beat it with some strategy σ' , then she could use σ' in the letter game to beat σ , a contradiction. From Lemma 12, and the determinacy of $G_k(S)$, Player 1 therefore wins $G_2(S)$ whenever he wins the letter game. ◀

We now consider the problem of deciding whether a given safety or reachability TA is history-deterministic. We use the observation that the k -token games played on LTSs induced by TA can be expressed as a timed parity game from [11] played on the $(k + 1)$ -fold product.

► **Lemma 15.** *For all k (given in unary) and timed safety or reachability automata \mathcal{T} , the game $G_k(\mathcal{T})$ is solvable in EXPTIME.*

Proof. $G_k(\mathcal{T})$ is a timed game on an arena consisting of the configuration space of the product of $k + 1$ copies of \mathcal{T} . The winning condition consists of a boolean combination of safety or reachability conditions. Such games can be solved as timed parity games as defined in [11] in time exponential in the number of clocks c and in k [11, Theorem 3]. Note that [11] uses concurrent timed parity games, of which turn-based ones are a special case. ◀

► **Theorem 16.** *Given a safety or reachability TA, deciding whether it is history-deterministic is decidable in EXPTIME.*

Proof. From Lemma 13 and Lemma 14, deciding the history-determinism of a safety or reachability TA \mathcal{T} reduces to solving $G_1(\mathcal{T})$ or $G_2(\mathcal{T})$ respectively, both of which can be done in EXPTIME, from Lemma 15. ◀

As explained in the introduction, this also solves the good-enough synthesis problem of deterministic safety and reachability TA.

6 Synthesis, Games and Composition

In this section we consider several games played on (LTSs of) timed automata and how they can be used to decide classical verification problems. We focus on turn-based games, although our techniques can be generalised to concurrent ones. We first look at language inclusion, then synthesis, and finally we consider good-for-games timed automata, that is, automata that preserve the winner when composed with a game and show that good-for-gameness and history-determinism coincide for both reachability and safety timed automata.

6.1 Language Inclusion and Fair Simulation Games

The connection between history-determinism and fair simulation, established in Theorem 4, allows to transfer decidability results to history-deterministic TA. Let's first recall that simulation checking is decidable for timed automata using a region construction [28]. This paper precedes the notion of fair simulation (restricting Player 1 to fair runs) and is thus only applicable for safety conditions. However, the result holds for more general parity acceptance (for which each state is assigned an integer priority and where a run is accepted if the highest priority it sees infinitely often is even).

► **Theorem 17.** *Fair simulation is decidable and EXPTIME-complete for parity timed automata.*

Proof. It suffices to observe that the simulation game can be presented as a timed parity game, as studied in [11], played on the product of two copies of the automaton. These can be solved in EXPTIME. A matching lower bound holds even for safety or reachability acceptance (see Lemma 24 in Appendix C for details). ◀

► **Corollary 18.** *Timed language inclusion is decidable and EXPTIME-complete for history-deterministic TA. More precisely, given a TA S with initial state q and a history-deterministic TA S' with initial state q' , checking if $q \subseteq_L q'$ holds is EXPTIME-complete.*

Proof. As \mathcal{B} is history-deterministic and by Theorem 4, we have $q \subseteq_L q'$ if, and only if, $q \preceq q'$. The result follows from Theorem 17. ◀

6.2 Synthesis Games

We show that as is the case in the regular [21], pushdown [25], cost function [12], and quantitative [9] settings, synthesis games with winning conditions given by history-deterministic TA are no harder to solve than those with for winning condition given by deterministic TA.

► **Definition 19** (Timed synthesis game). *Given a timed language $L \subseteq (\Sigma_I \times \Sigma_O)_T^\omega$, the synthesis game for L proceeds as follows. At turn i :*

- *Player I plays a delay d_i and a letter $a_i \in \Sigma_I$*
- *Player II plays a letter $b_i \in \Sigma_O$.*

Player II wins if $d_0 \binom{a_0}{b_0} d_1 \binom{a_1}{b_1} \dots \in L$ or if time does not progress. If Player II has a winning strategy in the synthesis game, we say that L is realisable.

► **Theorem 20.** *Given a history-deterministic timed parity automaton \mathcal{T} , the synthesis game for $L(\mathcal{T})$ is decidable and EXPTIME-complete.*

The proof (in Appendix C) follows a similar reduction to one in [25], in which the nondeterminism of the automaton is moved into Player 2's output alphabet, forcing her to simultaneously build a word in the winning condition and an accepting run witnessing this. Since accepting runs are recognised by deterministic automata, this reduces the problem to the synthesis problem for deterministic timed automata. The lower bound follows from the EXPTIME-completeness of synthesis for deterministic TA [14].

The EXPTIME decidability of universality for history-deterministic TA follows both from the decidability of language inclusion in the previous section and from the decidability of synthesis: the universality of \mathcal{T} reduces to deciding the winner of the synthesis game over $\{\binom{w}{w} \mid w \in L(\mathcal{T})\}$, recognised by a history-deterministic TA if \mathcal{T} is history-deterministic.

6.3 Composition with Games

Implicitly, at the heart of these reductions is the notion of composition: the composition of the game to solve with a history-deterministic automaton for the winning condition yields an equivalent game with a simpler winning condition. We say that an automaton is good-for-games if this composition operation preserves the winner of the game for all games. While history-determinism always implies good-for-gameness, the converse is not necessarily true. While the classes of history-deterministic and good-for-games automata coincide for ω -regular automata [8], this is not the case for quantitative automata [9], which can be good-for-games without being history-deterministic. We argue that for reachability and safety timed automata, good-for-gameness and history-determinism coincide.

► **Definition 21** (Timed Games). *A timed game (roughly following [14]), consists of an arena $\mathcal{G} = (Q, \iota, C, \Delta, \Sigma, L)$ and is similar to a TA except that Q , which need not be finite, is partitioned into $Q = Q_1 \uplus Q_2$, that is, positions Q_1 belonging to Player 1 and positions Q_2 belonging to Player 2, and L is a timed language, not an acceptance condition. Furthermore, an a -transition produces the letter a , rather than reads it. Configurations are defined as for TA and we assume every configuration to have at least one successor-configuration.*

A timed game proceeds in the configuration space of \mathcal{G} with Player 1 at each turn i advancing time with a delay $d_i \in \mathbb{R}$. Then, from the resulting configuration c_i , the owner of the state of c_i chooses a transition in Δ enabled in c_i , leading to a transition c_{i+1} producing a letter a_i . An infinite play is winning for Player 2 if the word $d_0 a_0 d_1 a_1 \dots$ produced is in L .

► **Definition 22** (Composition). *Intuitively, the composition of a game \mathcal{G} and an automaton \mathcal{T} consists of a game in which the two players play on \mathcal{G} while Player 2 must also build, letter by letter, a run of \mathcal{T} on the outcome of the game in \mathcal{G} . More formally, given a TA \mathcal{T} and a game \mathcal{G} with winning condition $L(\mathcal{T})$, the composition $\mathcal{T} \circ \mathcal{G}$ consists of a game played on the product of the configuration spaces of \mathcal{G} and \mathcal{T} , starting from the initial state of both, in which, at each turn i , from a configuration (c_i, c'_i) , Player 1 plays a time delay $d_i \in \mathbb{R}$, the owner of the current \mathcal{G} -state chooses a move in the configuration space of \mathcal{G} to a successor-configuration c_{i+1} , producing a letter a_i , and then Player 2 chooses a transition over (d_i, a_i) enabled at the current \mathcal{T} -configuration c'_i , leading to a successor-configuration c'_{i+1} . The game then proceeds from (c_{i+1}, c'_{i+1}) .*

Player 2 wins infinite plays if the run built in \mathcal{T} is accepting, and loses if it is rejecting or if she cannot move in the \mathcal{G} -component.

Observe that if Player 1 wins in \mathcal{G} , then he also wins in $\mathcal{T} \circ \mathcal{G}$ with a strategy that produces a word not in $L(\mathcal{T})$ in \mathcal{G} , as then Player 2 can not produce an accepting run in \mathcal{T} .

[9, Lemma 7] shows that for (quantitative) automata for which the letter-game is determined, (threshold) history-determinism coincides with good-for-gameness. The lemma is stated for quantitative automata, where thresholds are relevant; in the Boolean setting,

it simply states that the determinacy of the letter game implies the equivalence of history-determinism and good-for-gameness. In our timed setting, a similar argument, combined with the determinacy of the letter game for safety and reachability TA, gives us the following.

► **Theorem 23.** *Let \mathcal{T} be a safety or reachability TA. The following are equivalent:*

1. \mathcal{T} is history-deterministic.
2. For all timed games \mathcal{G} with winning condition $L(\mathcal{T})$, whenever Player 2 wins \mathcal{G} , she also wins $\mathcal{T} \circ \mathcal{G}$.

Proof.

(1) \implies (2). If \mathcal{T} is history-deterministic, the resolver can be used as a strategy in the \mathcal{T} component of $\mathcal{T} \circ \mathcal{G}$. When combined with a winning strategy in \mathcal{G} that guarantees that the \mathcal{G} -component produces a word in $L(\mathcal{T})$, the resolver guarantees that the \mathcal{T} -component produces an accepting run, thus giving the victory to Player 2.

(2) \implies (1). Towards a contradiction, assume \mathcal{T} is not history-deterministic, that is, by determinacy of the letter game from Remark 11, that Player 1 has a winning strategy σ in the letter game. Now consider the game \mathcal{G}_σ , without clocks or guards, in which positions, all belonging to Player 1, consist of the prefixes of timed words played by σ , with moves $w \xrightarrow{(t,a)} w(t,a)$. As σ is winning for Player 1, all maximal paths in \mathcal{G}_σ are labelled by a timed word in $L(\mathcal{T})$, so \mathcal{G}_σ is winning for Player 2.

We now argue that Player 1 wins $\mathcal{T} \circ \mathcal{G}_\sigma$ by interpreting Player 2's moves in the \mathcal{T} component as her moves in the letter game, and choosing moves in \mathcal{G} mimicking the letter dictated by σ . Then, if Player 2 could win against this strategy in $\mathcal{T} \circ \mathcal{G}_\sigma$, she could also win against σ in the letter game by interpreting Player 1's choices of letters as moves in \mathcal{G} , and responding with the same transition as she plays in the \mathcal{T} component of $\mathcal{T} \circ \mathcal{G}_\sigma$. Such a strategy is a valid strategy in the letter game on \mathcal{T} , and while it might not be winning in general, it is winning against σ , contradicting that σ is a winning strategy for Player 1. ◀

This proof fails for acceptance conditions beyond safety and reachability, as it isn't clear whether timed Büchi and coBüchi automata define Borel sets. If this was the case then history-deterministic timed automata would be exactly those that preserve winners in composition with games, as is the case in the ω -regular setting.

7 Conclusion

We introduced history-determinism for timed automata and showed that it suffices for solving important problems that previously required full determinism, in particular, timed language inclusion, universality and synthesis. We showed that for the important classes of timed safety and timed reachability automata, history-determinism can be checked (and therefore good-enough synthesis of deterministic reachability and safety automata can be solved) and every history-deterministic timed safety automaton can be determinized.

We conjecture that determinizability does not hold for history-deterministic timed coBüchi automata. Consider the timed coBüchi language “there is a real time t such that for every nonnegative integer i , there is a letter a at time $t + i$.” This timed language is recognised by a history-deterministic coBüchi automaton in which a nondeterministic transition guesses a “witness time” t after which a occurs at every unit interval, and which allows for an unbounded number of failed guesses (using the coBüchi condition). To see that this automaton is history-deterministic, let the resolver repeatedly and deterministically pick the time with the most previous occurrences of a at unit-interval distances. If a timed input word is in the language, then this resolver will eventually choose a correct witness time and produce an accepting run.

We conjecture that the complement of this language cannot be defined by a (nondeterministic) timed automaton. Informally, a timed automaton would require an unbounded number of clocks to check that “for all occurrences of a there is a nonnegative integer distance i such that a is not followed by another a after i time units.” If so, this timed language would separate the classes of deterministic and history-deterministic timed languages.

Let us conclude with another conjecture. We showed that history-deterministic timed automata are “good” for solving turn-based timed games, where in each turn of the game, one of the two players chooses a time delay or an action. A more general, concurrent setting for timed games is presented in [13]. In the concurrent version both players simultaneously choose permissible pairs of time delays and actions, and the player who has picked the shorter time delay gets to move. While concurrent games may not be determined, we conjecture that these concurrent timed games can again be solved by composing the (timed) arena with the (timed) winning condition, as long as the winning condition is history-deterministic.

References

- 1 Bader Abu Radi and Orna Kupferman. Minimizing gfg transition-based automata. In *International Colloquium on Automata, Languages and Programming (ICALP)*. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2019.
- 2 Shaull Almagor and Orna Kupferman. Good-enough synthesis. In *Computer Aided Verification (CAV)*, volume 12225 of *Lecture Notes in Computer Science*, pages 541–563. Springer, 2020.
- 3 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.
- 4 Rajeev Alur, Limor Fix, and Thomas A. Henzinger. Event-clock automata: A determinizable class of timed automata. *Theoretical Computer Science*, 211(1-2):253–273, 1999. doi:10.1016/S0304-3975(97)00173-4.
- 5 Rajeev Alur and Thomas A. Henzinger. Back to the future: Towards a theory of timed regular languages. In *33rd Annual Symposium on Foundations of Computer Science*, pages 177–186. IEEE Computer Society, 1992. doi:10.1109/SFCS.1992.267774.
- 6 Marc Bagnol and Denis Kuperberg. Büchi Good-for-Games Automata Are Efficiently Recognizable. In Sumit Ganguly and Paritosh Pandya, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 122 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSTTCS.2018.16.
- 7 Udi Boker, Denis Kuperberg, Karoliina Lehtinen, and Michal Skrzypczak. On succinctness and recognisability of alternating good-for-games automata. *CoRR*, abs/2002.07278, 2020. arXiv:2002.07278.
- 8 Udi Boker and Karoliina Lehtinen. Good for games automata: From nondeterminism to alternation. In *International Conference on Concurrency Theory (CONCUR)*, volume 140 of *LIPIcs*, pages 19:1–19:16, 2019.
- 9 Udi Boker and Karoliina Lehtinen. History Determinism vs. Good for Gameness in Quantitative Automata. In Mikołaj Bojańczyk and Chandra Chekuri, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 213 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 38:1–38:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2021.38.
- 10 Udi Boker and Karoliina Lehtinen. Token games and history-deterministic quantitative automata. In Patricia Bouyer and Lutz Schröder, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 120–139, Cham, 2022. Springer International Publishing. doi:10.1007/978-3-030-99253-8_7.

- 11 Krishnendu Chatterjee, Thomas A. Henzinger, and Vinayak S. Prabhu. Timed parity games: Complexity and robustness. In Franck Cassez and Claude Jard, editors, *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, pages 124–140, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- 12 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 139–150, 2009.
- 13 Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In *International Conference on Concurrency Theory (CONCUR)*, volume 2761 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2003. doi:10.1007/978-3-540-45187-7_9.
- 14 Deepak D’souza and P. Madhusudan. Timed control synthesis for external specifications. In *International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 571–582. Springer Berlin Heidelberg, 2002. doi:10.1007/3-540-45841-7_47.
- 15 Emmanuel Filiot, Christof Löding, and Sarah Winter. Synthesis from weighted specifications with partial domains over finite words. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2020.
- 16 Olivier Finkel. Undecidable problems about timed automata. In *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, volume 4202 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2006. doi:10.1007/11867340_14.
- 17 Shibashis Guha, Ismaël Jecker, Karoliina Lehtinen, and Martin Zimmermann. A Bit of Nondeterminism Makes Pushdown Automata Expressive and Succinct. In Filippo Bonchi and Simon J. Puglisi, editors, *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 53:1–53:20, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2021.53.
- 18 T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 394–406, 1992. doi:10.1109/LICS.1992.185551.
- 19 Thomas A. Henzinger, Peter W. Kopke, and Howard Wong-Toi. The expressive power of clocks. In *International Colloquium on Automata, Languages and Programming (ICALP)*, volume 944 of *Lecture Notes in Computer Science*, pages 417–428. Springer, 1995. doi:10.1007/3-540-60084-1_93.
- 20 Thomas A. Henzinger, Orna Kupferman, and Sriram K. Rajamani. Fair simulation. In *International Conference on Concurrency Theory (CONCUR)*, pages 273–287. Springer Berlin Heidelberg, 1997.
- 21 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In Zoltán Ésik, editor, *Computer Science Logic (CSL)*, pages 395–410, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- 22 Marcin Jurdzinski, Francois Laroussinie, and Jeremy Sproston. Model Checking Probabilistic Timed Automata with One or Two Clocks. *Logical Methods in Computer Science*, Volume 4, Issue 3, September 2008. doi:10.2168/LMCS-4(3:12)2008.
- 23 Denis Kuperberg and Michał Skrzypczak. On determinisation of good-for-games automata. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 299–310, 2015.
- 24 Orna Kupferman, Shmuel Safra, and Moshe Y Vardi. Relating word and tree automata. *Ann. Pure Appl. Logic*, 138(1-3):126–146, 2006. Conference version in 1996.
- 25 Karoliina Lehtinen and Martin Zimmermann. Good-for-games ω -pushdown automata. *Logical Methods in Computer Science*, 18, 2022.
- 26 Donald A Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

- 27 Sven Schewe. Minimising good-for-games automata is np-complete. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2020.
- 28 Serdar Tasiran, Rajeev Alur, Robert P. Kurshan, and Robert K. Brayton. Verifying abstractions of timed systems. In *International Conference on Concurrency Theory (CONCUR)*, volume 1119 of *Lecture Notes in Computer Science*, pages 546–562. Springer, 1996. doi:10.1007/3-540-61604-7_75.

A Expressivity

► **Lemma 7.** *Consider two region equivalent configurations $(s, \nu) \sim (s', \nu')$.*

For every timed word u there is a timed word u' so that the reduced run-tree on u from (s, ν) is equivalent to the reduced run-tree on u' from (s', ν') .

Proof. It suffices to show that for some (not necessarily reduced) run-tree on u from (s, ν) there exists some equivalent run-tree from (s', ν') as this implies the claim by collapsing all consecutive delay steps and thus producing the reduced tree on both sides.

We proceed by stepwise uncovering the run-tree from (s, ν) for ever longer prefixes of u and constructing a corresponding equivalent run-tree from (s', ν') . The intermediate finite trees we build have the property that all branches have the same duration. In each round we extend all current leafs, in both trees, either by

1. all possible non-deterministic successors (for the letter prescribed by the word u), in case the duration of the branch is already equal to the next time-stamp in u , or
2. one successor configuration due to a delay, which must be *the same on all leafs*.

For the second case, the delays used to extend the two trees need not be the same because we only want to preserve region equivalence. Also, the delay chosen for the tree rooted in (s, ν) need not follow the timestamps in u but can be shorter, meaning the run-tree may not be reduced.. The difficulty lies in systematically choosing the delays to ensure that the two trees remain equivalent and secondly, that in the limit this procedure generates a run-tree on the whole word u from (s, ν) . Together this implies the existence of a corresponding word u' and a run-tree from (s', ν') .

Invariant. To this end we propose a stronger invariant, namely that the relative orderings of the fractional values *in all leafs are the same on both sides*. To be precise, let's reinterpret a clock valuation as a function $\nu : C \times \mathbb{N} \rightarrow \{\perp\} \cup [0, 1)$, that assigns to every clock and possible integral value either a fractional value between 0 and 1, or \perp (indicating that the given clock does not have the given integral value). This way for every clock x there is exactly one $n \in \mathbb{N}$ with $\nu(x, n) \neq \perp$ and the image $\nu(C \times \mathbb{N})$ has at most $|C| + 1$ different elements. For any ordered set $F = \{\perp < f_1 < f_2 < \dots < f_l\} \supseteq \nu(C \times \mathbb{N})$ of fractional values, we can thus represent ν as a function $\hat{\nu} : C \times \mathbb{N} \rightarrow \{\perp, 1, \dots, l\}$ that, instead of exact fractional clock values only yields their index in F (and maps $\perp \mapsto \perp$).

Consider some run-tree with leafs $(q_1, \nu_1)(q_2, \nu_2) \dots (q_l, \nu_l)$ with combined fractional values $F = \bigcup_{i=1}^l \nu_i(C \times \mathbb{N})$, and an equivalent run-tree with leafs $(q'_1, \nu'_1)(q'_2, \nu'_2) \dots (q'_l, \nu'_l)$ with combined fractional values $F' = \bigcup_{i=1}^l \nu'_i(C \times \mathbb{N})$. The two trees are *aligned* if for all $1 \leq i \leq l$, $\hat{\nu}_i = \hat{\nu}'_i$. Notice that this still allows the two trees to differ on their exact fractional values but now they must agree on the relative order of all contained clocks on leafs, and in particular which ones are maximal and therefore the closest to the next larger integer. We will always select a delay of $1 - \max\{F\}$ and $1 - \max\{F'\}$, respectively, in step 2 above.

To show the claim we produce the required run-trees starting in $(s, \nu) \sim (s'\nu')$. These are in particular two aligned run-trees on the empty word.

Assume two aligned trees as above, where leaves have fractional values $F = \{\perp < f_1 < f_2 < \dots < f_m\}$ and $F' = \{f'_0 < f'_1 < \dots < f'_m\}$, respectively, and assume that the tree rooted in (s, ν) reads a strict prefix $(a_0, t_0), \dots, (a_i, t_i)$ of u .

Case 1: the duration of all branches in the first tree equals t_{i+1} , the timestamp of the next symbol in u . Then we extend each leaf in both trees by all possible a_{i+1} -successors. This will produce two aligned trees because each leaf configuration in one must be region equivalent to the corresponding configuration in the other, and therefore satisfies the same guards, enabling the same a_{i+1} -transitions leading to equivalent successors. Note also that all branches in each tree still have the same duration, as no delay step was taken.

Case 2: the duration of all branches in the first tree is strictly less than t_{i+1} . Then we extend all leaves in the tree from (s, ν) by a delay of duration $d = 1 - f_m$ and all leaves in the other tree by a delay of duration $d' = 1 - f'_m$. Naturally, this produces exactly one successor for each former leaf. The sets of new fractional values on leaves are $\bigcup_{i=1}^m (\mu + d)(C \times \mathbb{N}) = \{\perp < 0 < f_1 + d < \dots < f_{m-1} + d\}$ and for any former leaf (q, μ) extended by a delay $(q, \mu) \xrightarrow{d} (q, \mu + d)$, we have

$$\hat{\mu}(x, n - 1) = m \iff \widehat{(\mu + d)}(x, n) = 0 \quad (1)$$

and

$$\hat{\mu}(x, n) = i < m \iff \widehat{(\mu + d)}(x, n) = i + 1 \leq m \quad (2)$$

Analogous equivalences hold for the corresponding step $(q, \mu') \xrightarrow{d'} (q, \mu' + d')$ on the other tree. Notice that the two cases above are exhaustive as again, for all $x \in C$ there is exactly one $n \in \mathbb{N}$ with $\mu(x, n) \neq \perp$. We aim to show that $\widehat{(\mu + d)} = \widehat{(\mu' + d')}$. Consider any $x \in C$ and $n \in \mathbb{N}$. We have that

$$\begin{aligned} \widehat{(\mu + d)}(x, n) = m &\stackrel{(1)}{\iff} \hat{\mu}(x, n + 1) = 0 \\ &\stackrel{(IH)}{\iff} \hat{\mu}'(x, n + 1) = 0 \\ &\stackrel{(1)}{\iff} \widehat{(\mu' + d')}(x, n) = m \end{aligned}$$

and

$$\begin{aligned} \widehat{(\mu + d)}(x, n) = i < m &\stackrel{(2)}{\iff} \hat{\mu}(x, n) = i + 1 \\ &\stackrel{(IH)}{\iff} \hat{\mu}'(x, n) = i + 1 \\ &\stackrel{(2)}{\iff} \widehat{(\mu' + d')}(x, n) = i < m \end{aligned}$$

It follows that $\widehat{(\mu + d)} = \widehat{(\mu' + d')}$ which means that the two trees are again aligned, as required.

To see why this procedure produces a run-tree on u (and an equivalent run-tree on some word u'), observe that there can be at most $|F| + 1$ many consecutive delay extensions according to step 2) before all integral clock values are strictly increased. \blacktriangleleft

B Deciding History-determinism

► **Lemma 12.** *Given an fair LTS S , if Player 2 wins $G_2(S)$ then she wins $G_k(S)$ for all k .*

This is the generalisation of [6, Thm 14] (on ω -regular automata) to fair LTSs. The proof is similar to [6], without requiring positional strategies, and identical to that of [10, Theorem 4] (on quantitative automata), without the quantitative aspects. If Player 2 wins $G_2(S)$ then she obviously wins $G_1(S)$, using her G_2 strategy with respect to two copies of Player 1's single token in G_1 . We therefore consider below $k > 2$.

Let σ_2 be a winning strategy for Player 2 in $G_2(S)$. We inductively show that Player 2 has a winning strategy σ_i in $G_i(S)$ for each finite i . To do so, we assume a winning strategy σ_{i-1} in $G_{i-1}(S)$. The strategy σ_i maintains some additional (not necessarily finite) memory that maintains the position of one virtual token in S , a position in the (not necessarily finite) memory structure of σ_{i-1} , and a position in the (not necessarily finite) memory structure of σ_2 . The virtual token is initially at the initial state of S . Then, the strategy σ_i then plays as follows: at each turn, after Player 1 has moved his i tokens and played a letter (or, at the first turn, just played a letter), it first updates the σ_{i-1} memory structure, by ignoring the last of Player 1's tokens, and, treating the position of the virtual token as Player 2's token in $G_{i-1}(S)$, it updates the position of the virtual token according to the strategy σ_{i-1} ; it then updates the σ_2 memory structure by treating Player 1's last token and the virtual token as Player 1's 2 tokens in $G_2(S)$, and finally outputs the transition to be played according to σ_2 .

We now argue that this strategy is indeed winning in $G_i(S)$. Since σ_{i-1} is a winning strategy in $G_{i-1}(S)$, the virtual token traces an accepting run if any of the runs built by the first $i - 1$ tokens of Player 1 is accepting. Since σ_2 is also winning, the run built by Player 2's token is accepting if either the run built by the virtual token or by Player 1's last token is accepting. Hence, Player 2's is accepting whenever one of Player 1's runs is accepting, making this a winning strategy in $G_i(S)$.

► **Lemma 13.** *Given a fair LTS S with a safety acceptance condition, Player 2 wins $G_1(S)$ if and only if S is history-deterministic.*

Proof. If S is history-deterministic then Player 2 wins $G_1(S)$ by using the resolver to choose her transitions. This guarantees that for all words in $L(S)$ played by Player 1, her run is accepting, which makes her victorious regardless of Player 1's run.

For the converse, if Player 2 wins $G_1(S)$, consider the following family of *copycat strategies* for Player 1: at first, Player 1 plays σ and chooses the same transitions as Player 2; if, eventually, Player 2 chooses a transition τ from a configuration c that is not language-maximal, that is, moves to a configuration c' that does not accept some word w that is accepted by some other configuration c'' reachable by some other transition τ' from c , we call such a move non-cautious, and Player 1 stops copying Player 2 and instead chooses τ' . From there, Player 1 wins by playing w and an accepting run on w from c'' . Since Player 2 wins $G_1(S)$, her winning strategy σ does not play any non-cautious moves against copycat strategies.

Then, she can use σ in the letter-game, by playing as σ would play in $G_1(S)$ if Player 1 copies her transitions. This guarantees that she never makes a non-cautious move, and, in particular, never moves out of the safe region of the automaton unless the prefix played by Player 1 has no continuations in $L(S)$. This is a winning strategy in the letter-game, so S is history-deterministic. ◀

C Synthesis, Games and Composition

► **Theorem 20.** *Given a history-deterministic timed parity automaton \mathcal{T} , the synthesis game for $L(\mathcal{T})$ is decidable and EXPTIME-complete.*

Proof. For the upper bound, we reduce the problem to solving synthesis games for deterministic timed parity automata, which is in EXPTIME [14].

Let $\mathcal{T} = (S, \iota, C, \Delta, \Sigma, Acc)$ be a timed automaton. Let \mathcal{T}' be the deterministic timed automaton $(S, \iota, C, \Delta', \Sigma \times \Delta, Acc)$ where:

$$\Delta' = \{(s, g, (\sigma, (s, g, \sigma, c, s')), c, s') \mid (s, g, \sigma, c, s') \in \Delta\}$$

In other words, \mathcal{T}' is a deterministic automaton with the state space of \mathcal{T} , over the alphabet $\Sigma \times \Delta$, where the transition in the input letter dictates the transition in the automaton. The language of \mathcal{T}' is the set of words (w, ρ) such that there is an accepting run of \mathcal{T} over w along the transitions of ρ .

We now claim that given a history-deterministic automaton \mathcal{T} with resolver r , Player II wins the synthesis game on \mathcal{T} if and only if she wins it on \mathcal{T}' . First assume that Player II wins the synthesis game for \mathcal{T} with a strategy s . Then, to win the synthesis game for \mathcal{T}' , at each turn i , after Player I plays d_i and a_i , she needs to make two choices: she must choose both a response letter b_i and a transition in \mathcal{T} over (a_i, b_i) . Given Player I's move and the (first component of the) word built so far, she can use the strategy s to choose the response letter b_i ; this guarantees that the first component of the play is a word accepted by \mathcal{T} . To choose the transition of \mathcal{T} , she can use the resolver r : given the run ρ built from the delays (including d_i) and transitions played so far, she plays $r(\rho, (a_i, b_i))$. Since r is a resolver, this strategy guarantees that the resulting run is accepting, and hence that she wins the synthesis game on \mathcal{T}' .

On the other hand, if Player I wins the synthesis game on \mathcal{T} , he has a strategy s which guarantees a play $w \in (\Sigma_i \times \Sigma_o)^T$ that is not in the language of \mathcal{T} . He can use the same strategy in the synthesis game of \mathcal{T}' to guarantee a play (w, ρ) such that w is not in the language of \mathcal{T} , and by extension (w, ρ) is not in the language of \mathcal{T}' , as there are no accepting runs over w in \mathcal{T} .

The lower bound follows from the EXPTIME-completeness of synthesis for deterministic TA [14]. \blacktriangleleft

Below we demonstrate that fair simulation checking for TA is EXPTIME-hard even for very simple acceptance conditions.

► **Lemma 24.** *Checking fair simulation between TA is EXPTIME-hard already for reachability or safety acceptance, or over finite words.*

Proof. This can be shown by reduction from *countdown games* [22], which are two-player games (Q, T, k) given by a finite set Q of control states, a finite set $T \subseteq (Q \times \mathbb{N}_{>0} \times Q)$ of transitions, labelled by positive integers, and a target number $k \in \mathbb{N}$. All numbers are given in binary encoding. The game is played in rounds, each of which starts in a pair (p, n) where $p \in Q$ and $n \leq k$, as follows. First Player 1 picks a number $l \leq k - n$, so that at least one $(p, l, p') \in T$ exists; Then Player 2 picks one such transition and the next round starts in $(p', n + l)$. Player 1 wins iff she can reach a configuration (q, k) for some state q .

Determining the winner in a countdown game is EXPTIME-complete [22] and can easily be encoded as a simulation game between two TAs \mathcal{A} and \mathcal{B} as follows. Let \mathcal{A} be the TA with no clocks and unrestricted (guards are *True*) self-loops for the two letters a and e ; The idea is that Player 1 proposes l by waiting that long and then makes a discrete a -labelled move. Then Player 2, currently in some state p can update his configuration to mimic that of the countdown game, and punish (by going to a winning sink) if Player 1 cheated or the game should end. To implement this, \mathcal{B} has two clocks: one to store n – the total time that passed – and one to store the current l , which is reset in each round.

Suppose Player 1 waits for l units of time and then proposes a . Player 2, currently in some state p will have

- a and e -labelled transitions to a winning state with a guard that verifies that there is no transition (p, l, p') .
- a -labelled transitions to a state p' , with a guard that verifies that a some $(p, l, p') \in T$ exists, and which resets clock x_2 .
- a , and e -labelled transitions to a winning state guarded by $x_1 > k$. This enables Player 2 to win if the global time has exceeded the target k .

The only way that Player 1 can win is by following a winning strategy in the countdown game and by playing the letter e once \mathcal{B} is in a configuration (q, k) . Player 2 will not be able to respond. ◀

Decidability of One-Clock Weighted Timed Games with Arbitrary Weights

Benjamin Monmege ✉ 

Aix Marseille Univ, CNRS, LIS, Marseille, France

Julie Parreaux ✉

Aix Marseille Univ, CNRS, LIS, Marseille, France

Pierre-Alain Reynier ✉

Aix Marseille Univ, CNRS, LIS, Marseille, France

Abstract

Weighted Timed Games (WTG for short) are the most widely used model to describe controller synthesis problems involving real-time issues. Unfortunately, they are notoriously difficult, and undecidable in general. As a consequence, one-clock WTG has attracted a lot of attention, especially because they are known to be decidable when only non-negative weights are allowed. However, when arbitrary weights are considered, despite several recent works, their decidability status was still unknown. In this paper, we solve this problem positively and show that the value function can be computed in exponential time (if weights are encoded in unary).

2012 ACM Subject Classification Software and its engineering → Formal software verification; Theory of computation → Algorithmic game theory

Keywords and phrases Weighted timed games, Algorithmic game theory, Timed automata

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.15

Related Version *Full Version*: <https://arxiv.org/abs/2207.01608> [21]

Funding This work was partially funded by ANR project Ticktac (ANR-18-CE40-0015) and ANR project DeLTA (ANR-16-CE40-0007).

1 Introduction

The task of designing programs is becoming more and more involved. Developing formal methods to ensure their correctness is thus an important challenge. Programs sensitive to real-time allow one to measure time elapsing in order to take decisions. The design of such programs is a notoriously difficult problem because timing issues may be intricate, and a posteriori debugging such issues is hard. The model of timed automata [2] has been widely adopted as a natural and convenient setting to describe real-time systems. This model extends finite-state automata with finitely many real-valued variables, called clocks, and transitions can check clocks against lower/upper bounds and reset some clocks.

Model-checking aims at verifying whether a real-time system modelled as a timed automaton satisfies some desirable property. Instead of verifying a system, one can try to synthesise one automatically. A successful approach, widely studied during the last decade, is one of the two-player games. In this context, a player represents the *controller*, and an antagonistic player represents the *environment*. Being able to identify a winning strategy of the controller, i.e. a recipe on how to react to uncontrollable actions of the environment, consists in the synthesis of a system that is guaranteed to be correct by construction.

In the realm of real-time systems, timed automata have been extended to timed games [3] by partitioning locations between the two players. In a turn-based fashion, the player that must play proposes a delay and a transition. The controller aims at satisfying some ω -regular



© Benjamin Monmege, Julie Parreaux, and Pierre-Alain Reynier;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 15; pp. 15:1–15:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

objective however the environment player behaves. Deciding the winner in such turn-based timed games has been shown to be EXPTIME-complete [18], and a symbolic algorithm allowing tool development has been proposed [4].

In numerous application domains, in addition to real-time, other quantitative aspects have to be taken into account. For instance, one could aim at minimising the energy used by the system. To address this quantitative generalisation, weighted (aka priced) timed games (WTG for short) have been introduced [8, 5]. Locations and transitions are equipped with integer weights, allowing one to define the accumulated weight associated with a play. In this context, one focuses on a simple, yet natural, reachability objective: given some target location, the controller, that we now call *Min*, aims at ensuring that it will be reached while minimising the accumulated weight. The environment, that we now call *Max*, has the opposite objective: avoid the target location or, if not possible, maximise the accumulated weight. This allows one to define the value of the game as the minimal weight *Min* can guarantee. The associated decision problem asks whether this value is less than or equal to some given threshold.

In the earliest studies of this problem, [1, 8] proposed semi-decision procedures to approximate this value for WTG with non-negative weights. In addition, [8] identifies the subclass of strictly non-Zeno cost WTG for which their algorithm terminates. This approximation is motivated by the undecidability of the problem, first shown in [11]. This restriction has recently been lifted to WTG with arbitrary weights in [15].

An orthogonal research direction to recover decidability is to reduce the number of clocks and more precisely to focus on *one-clock WTG*. Though restricted, a single clock is often sufficient for modelling purposes. When only non-negative weights are considered, decidability has been proven in [10] and later improved in [22, 17] to obtain exponential time algorithms. Despite several recent works, the decidability status of one-clock WTG with arbitrary weights is still open. In the present paper, we show the decidability of the value problem for this class. More precisely, we prove that the value function can be computed in exponential time (if weights are encoded in unary and not in binary).

Before exposing our approach, let us briefly recap the existing results. Positive results obtained for one-clock WTG with non-negative weights are based on a reduction to so-called *simple WTG*, where the underlying timed automata contain no guard, no reset, and the clock value along with the execution exactly spans the $[0, 1]$ interval. In simple WTG, it is possible to compute (in exponential time) the whole value function starting at time 1 and going back in time until 0 [10, 22]. Another technique, that we will not explore further in the present work, consists in using the paradigm of strategy iteration [17], leading to an exponential-time algorithm too. A PSPACE lower-bound is also known for related decision problems [16].

More recent works extend the positive results of simple WTG to arbitrary weights [12, 13], yielding decidability of *reset-acyclic* one-clock WTG with arbitrary weights, with a *pseudo-polynomial time* complexity (that is polynomial if weights are encoded in unary). It is also explained how to extend the result to all WTG where no cyclic play containing a reset may have a negative weight arbitrarily close to 0. Moreover, it is shown that *Min* needs memory to play (almost-)optimally, in a very structured way: *Min* uses *switching strategies*, that are composed of two memoryless strategies, the second one being triggered after a given (pseudo-polynomial) number κ of steps.

The crucial ingredient to obtain decidability for non-negative weights or reset-acyclic weighted timed games is to limit the number of reset transitions taken along a play. This is no longer possible in presence of cycles of negative weights containing a reset. There, *Min*

may need to iterate cycles for a number κ of times that depends on the desired precision ε on the value (to play ε -optimally, Min needs to cycle $O(1/\varepsilon)$ times, see Example 3). To rule out these annoying behaviours, we rely on three main ingredients:

- As there is a single clock, a cyclic path ending with a reset corresponds to a cycle of configurations. We define the *value* of such a cycle, that allows us to identify which player may benefit from iterating it.
- Using the classical region graph construction, we prove stronger properties on the value function (it is continuous on the closure of region intervals). This allows us to prove that Max has an optimal memoryless strategy that avoids cycles whose value is negative.
- We introduce a partial unfolding of the game, so as to obtain an acyclic WTG, for which decidability is known. To do so, we rely on the existence of (almost-)optimal switching strategies for Min, allowing us to limit the depth of exploration. Also we keep track of cycles encountered and handle them according to their value. Using the previous result on the existence of a “smart” optimal strategy for Max, we show that this unfolding has the same value as the original WTG.

The paper is organized as follows: weighted timed games are presented in Section 2. We then focus on cycles in Section 3. Our unfolding is presented in Section 4, with a sketch of the main proof. Some of the technical proofs can be found in Appendix, and a long version is available with all the proofs [21].

2 Weighted timed games

2.1 Definitions

We will consider weighted timed games with a single clock, denoted by x . The valuation of this clock is a non-negative real number ν . On such a clock, transitions of the timed games will be able to check some interval constraints on the clock, i.e. intervals I of real values with closed or open bounds that are natural numbers (or $+\infty$). For every interval $I = (a, b)$ we denote by $\bar{I} = [a, b]$ its closure.

- **Definition 1.** A weighted timed game is a tuple $\langle Q_{\text{Min}}, Q_{\text{Max}}, Q_t, Q_u, \Delta, \text{wt}, \text{wt}_t \rangle$ with
- $Q = Q_{\text{Min}} \uplus Q_{\text{Max}} \uplus Q_t$ a finite set of locations split between players Min and Max (in drawings, locations belonging to Min are depicted by circles and the ones belonging to Max by squares) and a set of target locations;
 - $Q_u \subseteq Q_{\text{Min}} \uplus Q_{\text{Max}}$ a set of urgent locations where time cannot be delayed;
 - Δ a finite set of transitions each of the form $q \xrightarrow{I, R, w} q'$, with q and q' two locations (with $q \notin Q_t$), I an interval, $w \in \mathbb{Z}$ being the weight of the transition, and R being either $\{x\}$ when the clock must be reset (depicted by $x := 0$), or \emptyset when it does not;
 - $\text{wt}: Q \rightarrow \mathbb{Z}$ a weight function associating an integer weight with each location: for uniformisation of the notations, we extend this weight function to also associate with each transition the weight it contains, i.e. $\text{wt}(q \xrightarrow{I, R, w} q') = w$;
 - and $\text{wt}_t: Q_t \times \mathbb{R}_{\geq 0} \rightarrow \bar{\mathbb{R}}$ a function mapping each target configuration to a final weight, where $\bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$.

The addition of final weights in weighted timed games (WTG) is not standard, but we use it in the process of solving those games: in any case, it is possible to simply map a given target location to the weight 0, allowing us to recover the standard definitions of the literature. The presence of urgent locations is also unusual: in a timed automaton with several clocks, urgency can be modelled with an additional clock u that is reset just before

15:4 Decidability of One-Clock Weighted Timed Games with Arbitrary Weights

entering the urgent location and with constraints $u \in [0, 0]$ on outgoing transitions. However, when limiting the number of clocks to one, we regain modelling capabilities by allowing for such urgent locations. The weight of an urgent location is never used and will thus not be given in drawings: instead, urgent locations will be displayed with a u inside.

The semantics of a WTG \mathcal{G} is defined in terms of an infinite transition system $\llbracket \mathcal{G} \rrbracket$ whose vertices are configurations $(q, \nu) \in Q \times \mathbb{R}_{\geq 0}$. Configurations are split into players according to the location q , and a configuration (q, ν) is a target if $q \in Q_t$. Edges linking vertices will be labelled by elements of $\mathbb{R}_{\geq 0} \times \Delta$, to encode the delay that a player wants to spend in the current location, before firing a certain transition. For every delay $t \in \mathbb{R}_{\geq 0}$, transition $\delta = q \xrightarrow{I, R, w} q' \in \Delta$ and valuation ν , we add a labelled edge $(q, \nu) \xrightarrow{t, \delta} (q', \nu')$ if

- $\nu + t \in I$;
- $\nu' = 0$ if $R = \{x\}$, and $\nu' = \nu + t$ otherwise;
- and $t = 0$ if $q \in Q_u$.

This edge is given a weight $t \times \text{wt}(q) + \text{wt}(\delta)$ taking into account discrete and continuous weights.

As usual in related work [1, 8, 9], we will assume that the valuation of the clock x is *bounded* by the greatest constant M to appear in guards, and we, therefore, restrict ourselves to configurations of the form $(q, \nu) \in Q \times [0, M]$. We also suppose the absence of deadlocks except on target locations, i.e. for each location $q \in Q \setminus Q_t$ and valuation $\nu \in [0, M]$, there exist $t \in \mathbb{R}_{\geq 0}$ and $\delta = q \xrightarrow{I, R, w} q' \in \Delta$ such that $(q, \nu) \xrightarrow{t, \delta} (q', \nu')$, and no transitions start from Q_t . This second restriction is without loss of generality by applying classical techniques [6, Lemma 5].

We also assume that the final weight functions satisfy a sufficient property ensuring that they can be encoded in finite space. First, we call *regions*¹ of \mathcal{G} the set

$$\text{Reg}_{\mathcal{G}} = \{(M_i, M_{i+1}) \mid 0 \leq i \leq k-1\} \cup \{\{M_i\} \mid 0 \leq i \leq k\}$$

where $M_0 = 0 < M_1 < \dots < M_k$ are all the endpoints of the intervals appearing in the guards of \mathcal{G} (to which we add 0 if needed). Then, we require final weight functions to be piecewise affine with a finite number of pieces and continuous on each region. More precisely, we assume that cutpoints and coefficients are rational and given in binary.

We let W_{loc} , W_{tr} and W_{fin} be the maximum absolute value of weights of locations, transitions and final functions, i.e.

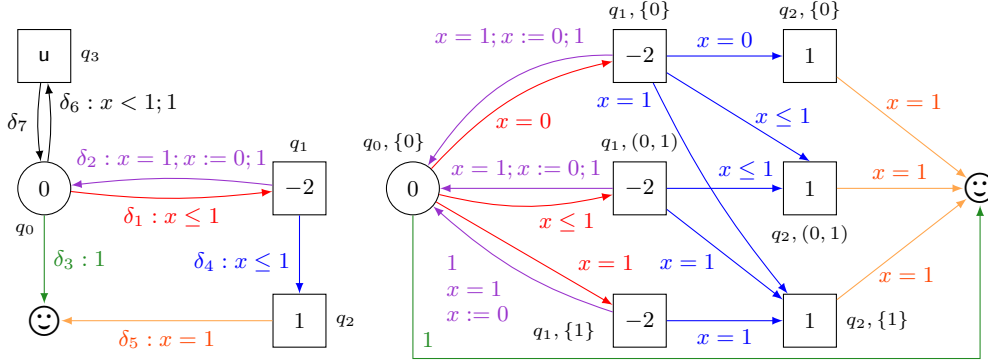
$$W_{\text{loc}} = \max_{q \in Q} |\text{wt}(q)| \quad W_{\text{tr}} = \max_{\delta \in \Delta} |\text{wt}(\delta)| \quad W_{\text{fin}} = \sup_{q \in Q_t \text{ s.t. } \text{wt}_t(q, \cdot) \notin \{+\infty, -\infty\}} \sup_{\nu \in I} |\text{wt}_t(q, \nu)|$$

We also let W be the maximum of W_{loc} , W_{tr} , and W_{fin} .

We call *path* a finite or infinite sequence of consecutive transitions $\delta_0, \delta_1, \dots$ of Δ , that we sometimes denote by $q_0 \xrightarrow{\delta_0} q_1 \xrightarrow{\delta_1} q_2 \dots$. We let FPaths be the set of all finite paths. We let $|\pi|$ be the number of transitions in the finite path π , that we call its *length*. For a given transition δ , we let $|\pi|_{\delta}$ denote the number of occurrences of δ in π .

We call *play* a finite or infinite sequence of edges in the semantics of the game that we denote by $(q_0, \nu_0) \xrightarrow{t_0, \delta_0} (q_1, \nu_1) \xrightarrow{t_1, \delta_1} (q_2, \nu_2) \dots$. A play is said to *follow* a path if both use the same sequence of transitions. We let $|\rho|$ be the *length* of play, defined as the length of the path it follows. We let $|\rho|_{\delta}$ be the number of occurrences of the transition δ in the finite play ρ . More generally, for all sets of transitions A , we let $|\rho|_A$ be the number of

¹ This is inspired by a construction by Laroussinie, Markey, and Schnoebelen [19], which allows one to reduce the number of regions with respect to the more usual one of [2] in the case of a single clock.



■ **Figure 1** On the left, a WTG with a cyclic path of weight $[-1, 1]$ containing a reset. Missing weights are 0. The target location is \oplus , whose final weight function is zero. Location q_3 is urgent. On the right, its closure restricted to locations q_0, q_1, q_2 and \oplus .

occurrences of all transitions from A in the finite play ρ , i.e. $|\rho|_A = \sum_{\delta \in A} |\rho|_\delta$. We let FPlays be the set of finite plays. For a finite path π or a finite play ρ , we let $\text{last}(\pi)$ and $\text{last}(\rho)$ be the last location or configuration. We let $\text{FPaths}_{\text{Max}}$ (respectively, $\text{FPaths}_{\text{Min}}$) and $\text{FPlays}_{\text{Max}}$ (respectively, $\text{FPlays}_{\text{Min}}$) be the subset of finite paths or plays whose last element belong to player Max (respectively, Min).

A finite play ρ can be associated with a weight that consists in accumulating the weight of the edges it traverses: if $\rho = (q_0, \nu_0) \xrightarrow{t_0, \delta_0} (q_1, \nu_1) \cdots (q_k, \nu_k)$, we let

$$\text{wt}_\Sigma(\rho) = \sum_{i=0}^{k-1} (\text{wt}(\ell_i) \times t_i + \text{wt}(\delta_i)).$$

A maximal play ρ (either infinite or trapped in a deadlock that is necessarily a target configuration) is associated with a payoff $\text{P}(\rho)$ as follows: the payoff of an infinite play (meaning that it never visits a target location) is $+\infty$, while the payoff of a finite play, thus ending in a target configuration (q, ν) , is $\text{wt}_\Sigma(\rho) + \text{wt}_t(q, \nu)$. The weight of a finite path π consists of the set of the cumulated weight of all the finite plays that follow π : $\text{wt}_\Sigma(\pi) = \{\text{wt}_\Sigma(\rho) \mid \rho \text{ following } \pi\}$. By [5], the weight of a path is known to be an interval of values. Moreover, when all the guards along the path are closed intervals, the weight of the path is also a closed interval.

A *cyclic path* is a finite path that starts and ends in the same location. A *cyclic play* is a finite play that starts and ends in the same configuration: it necessarily follows a cyclic path, but the reverse might not be true since some non-cyclic plays can follow a cyclic path (if they do not end in the same clock valuation as the one in which they start).

► **Example 2.** The cyclic path $\pi = q_0 \xrightarrow{\delta_1} q_1 \xrightarrow{\delta_2} q_0$ depicted on the left in Figure 1 has a weight between -1 (with the play $(q_0, 0) \xrightarrow{0, \delta_1} (q_1, 0) \xrightarrow{1, \delta_2} (q_0, 0)$) and 1 (with the play $(q_0, 0) \xrightarrow{1, \delta_1} (q_1, 1) \xrightarrow{0, \delta_2} (q_0, 0)$), so $\text{wt}_\Sigma(\pi) = [-1, 1]$. Another cyclic path is $\pi' = q_0 \xrightarrow{\delta_6} q_3 \xrightarrow{\delta_7} q_0$ which goes via an urgent location. All plays that follow this path are of the form $(q_0, \nu) \xrightarrow{t, \delta_6} (q_3, \nu + t) \xrightarrow{0, \delta_7} (q_0, \nu + t)$ with ν and $\nu + t$ less than 1, that all have a weight 1. Thus $\text{wt}_\Sigma(\pi') = \{1\}$.

A *strategy* aims at giving the recipe of each player. A strategy of Min is a function $\sigma: \text{FPlays}_{\text{Min}} \rightarrow \mathbb{R}_{\geq 0} \times \Delta$ mapping each finite play ρ whose last configuration belongs to Min to a pair (t, δ) of delay and transition, such that the play ρ can be extended by an edge

labelled with (t, δ) . A similar definition holds for strategies τ of Max. We let $\text{Strat}_{\text{Min}, \mathcal{G}}$ (respectively, $\text{Strat}_{\text{Max}, \mathcal{G}}$) be the set of strategies of Min (respectively, Max) in the game \mathcal{G} , or simply $\text{Strat}_{\text{Min}}$ and $\text{Strat}_{\text{Max}}$ if the game is clear from the context: we will always use letters σ and τ to differentiate from strategies of Min and Max.

A strategy is said to be *memoryless* if it only depends on the last configuration of the plays. More formally, Max's strategy τ is memoryless if for all plays ρ and ρ' such that $\text{last}(\rho) = \text{last}(\rho')$, we have $\tau(\rho) = \tau(\rho')$.

A play ρ is said to be *conforming* to a strategy σ (respectively, τ) if the choice made in ρ at each location of Min (respectively, Max) is the one prescribed by σ (respectively, τ). Moreover, a finite path π is said to be conforming to a strategy σ (respectively, τ) if there exists a finite play following π that is conforming to σ (respectively, τ).

After both players have chosen their strategies σ and τ , each initial configuration (q, ν) gives rise to a unique maximal play that we denote by $\text{Play}((q, \nu), \sigma, \tau)$. The *value* of the configuration (q, ν) is then obtained by letting players choose their strategies as they want, first Min and then Max, or vice versa since WTG is known to be determined [12]:

$$\text{Val}_{\mathcal{G}}(q, \nu) = \sup_{\tau} \inf_{\sigma} \text{P}(\text{Play}((q, \nu), \sigma, \tau)) = \inf_{\sigma} \sup_{\tau} \text{P}(\text{Play}((q, \nu), \sigma, \tau)).$$

The value of a strategy σ of Min (symmetric definitions can be given for strategies τ of Max) is defined as $\text{Val}_{\mathcal{G}}^{\sigma}(q, \nu) = \sup_{\tau} \text{P}(\text{Play}((q, \nu), \sigma, \tau))$. Then, a strategy σ^* of Min is *optimal* if, for all initial configurations (q, ν) , $\text{Val}_{\mathcal{G}}^{\sigma^*}(q, \nu) \leq \text{Val}_{\mathcal{G}}(q, \nu)$. Because of the infinite nature of the timed games, optimal strategies may not exist: for example, a player may want to let time elapse as much as possible, but with a delay $t < 1$ because of a strict guard, preventing them to obtain the optimal value. We will see in Example 12 that this situation can even happen when all guards contain only *closed* comparisons. We naturally extend the definition to *almost-optimal strategies*, taking into account small possible errors: we say that a strategy σ^* of Min is ε -*optimal* if, for all initial configurations (q, ν) , $\text{Val}_{\mathcal{G}}^{\sigma^*}(q, \nu) \leq \text{Val}_{\mathcal{G}}(q, \nu) + \varepsilon$.

► **Example 3.** We have seen that in q_0 (on the left in Figure 1), Min has no interest in following the cycle $q_0 \xrightarrow{\delta_6} q_3 \xrightarrow{\delta_7} q_0$ since it has weight $\{1\}$. Jumping directly to the target location via δ_3 leads to a weight of 1. But Min can do better: from valuation 0, by jumping to q_1 after a delay of $t \leq 1$, it leaves a choice to Max to either jump to q_2 and the target leading to a total weight of $1 - t$, or to loop back in q_0 thus closing a cyclic play of weight $-2(1 - t) + 1 = 2t - 1$. If t is chosen too close to 1, the value of the cycle is greater than 1, and Max will benefit from it by increasing the total weight. If t is chosen as smaller than $1/2$, the weight of the cycle is negative, and Max will prefer to go to the target to obtain a weight $1 - t$ close to 1, not very beneficial to Min. Thus, Min prefers to play just above $1/2$, let say at $1/2 + \varepsilon$. In this case, Max will choose to go to the target with a total weight of $1/2 + \varepsilon$. The value of the game, in configuration $(q_0, 0)$ is thus $\text{Val}_{\mathcal{G}}(q_0, 0) = 1/2$. Not only Min does not have an optimal strategy (but only ε -optimal ones, for every $\varepsilon > 0$), but needs memory to play ε -optimally, since Min cannot play *ad libitum* transition δ_2 with a delay $1/2 - \varepsilon$: in this case, Max would prefer staying in the cycle, thus avoiding the target. Thus, Min will play the transition δ_1 at least $1/4\varepsilon$ times so that the cumulated weight of all the cycles is below $-1/2$, in which case Min can safely use transition δ_1 still earning $1/2$ in total.

2.2 Closure

We first recall more in details the method used to solve WTG in [12], starting with the (slightly updated presentation of the) construction that consists in enhancing the locations with regions and closing all guards while preserving the value of the game.

► **Definition 4.** The closure of a WTG \mathcal{G} is the WTG $\overline{\mathcal{G}} = \langle L_{\text{Min}}, L_{\text{Max}}, L_t, L_u, \overline{\Delta}, \overline{\text{wt}}, \overline{\text{wt}_t} \rangle$ where:

- $L = L_{\text{Min}} \uplus L_{\text{Max}} \uplus L_t$ with $L_{\text{Min}} = Q_{\text{Min}} \times \text{Reg}_{\mathcal{G}}$, $L_{\text{Max}} = Q_{\text{Max}} \times \text{Reg}_{\mathcal{G}}$, $L_t = Q_t \times \text{Reg}_{\mathcal{G}}$, $L_u = Q_u \times \text{Reg}_{\mathcal{G}}$;
- for all $(q, I) \in L$, $(q, I) \xrightarrow{I_g \cap I'', R, w} (q', I') \in \overline{\Delta}$ if and only if there exist a transition $q \xrightarrow{I_g, R, w} q' \in \Delta$, and a region I'' such that $I_g \cap I'' \neq \emptyset$, the lower bound of I'' is at least the one of I (to model time elapsing), and I' is equal to I'' if $R = \emptyset$ and to $\{0\}$ otherwise: $\overline{I_g \cap I''}$ stands for the topological closure of the non-empty interval $I_g \cap I''$;
- for all (q, I) , we have $\overline{\text{wt}}(q, I) = \text{wt}(q)$;
- for all $(q, I) \in L_t$, for $\nu \in I$, $\overline{\text{wt}_t}((q, I), \nu) = \text{wt}_t(q, \nu)$ and extend $\nu \mapsto \overline{\text{wt}_t}((q, I), \nu)$ by continuity on \overline{I} , the closure of the interval I . We may also let $\overline{\text{wt}_t}((q, I), \nu) = +\infty$ for all $\nu \notin \overline{I}$, even though we will never use this in the following.

The following set of configurations is an invariant of the closure (i.e. starting from such configuration fulfilling the invariant, we can only reach configurations fulfilling the invariant):

- configurations $((q, \{M_k\}), M_k)$;
- and configurations $((q, (M_k, M_{k+1})), \nu)$ with $\nu \in [M_k, M_{k+1}]$ (and not only in (M_k, M_{k+1}) as one might expect).

► **Example 5.** Figure 1 depicts the closure (left) of the WTG (right) restricted to locations q_0, q_1, q_2 , and \ominus (we have seen that q_3 is anyway useless).

The closure of the guards allows players to mimic a move in \mathcal{G} “arbitrarily close” to M_{k+1} in (M_k, M_{k+1}) to be simulated by jumping on M_{k+1} still in the region (M_k, M_{k+1}) .

► **Lemma 6** ([12]). For all WTG \mathcal{G} , $(q, I) \in Q \times \text{Reg}_{\mathcal{G}}$ and $\nu \in I$, $\text{Val}_{\mathcal{G}}(q, \nu) = \text{Val}_{\overline{\mathcal{G}}}((q, I), \nu)$.

It is also shown in [12] that we can transform an ε -optimal strategy of $\overline{\mathcal{G}}$ into an ε' -optimal strategy of \mathcal{G} with $\varepsilon' < 2\varepsilon$ and vice-versa. Not only the closure construction adds the capability for a player to play “arbitrarily close” to the border of a region as a new move, but it also makes the value function more manageable for our purpose. Indeed, as shown in [12], the mapping $\nu \mapsto \text{Val}_{\mathcal{G}}(\ell, \nu)$ is continuous over all regions, but there might be discontinuities at the borders of the regions. The closure construction clears this issue by softening the borders of each region independently:

► **Lemma 7.** For all WTG \mathcal{G} and $(q, I) \in Q \times \text{Reg}_{\mathcal{G}}$, the mapping $\nu \mapsto \text{Val}_{\overline{\mathcal{G}}}((q, I), \nu)$ is continuous over \overline{I} .

In [12], it is also shown that the mapping $\nu \mapsto \text{Val}_{\mathcal{G}}(\ell, \nu)$ is piecewise affine on each region where it is not infinite, that the total number of pieces (and thus of *cutpoints*, in-between two such affine pieces) is exponential, and that all cutpoints and the value associated to such a cutpoint are rational numbers. In more recent developments in [13], authors improve the exponential complexity into *pseudo-polynomial* (i.e. polynomial in the number of locations and in the biggest weight W), which we will use in the sequel. Thus, they obtain:

► **Theorem 8** ([13]). If \mathcal{G} is an acyclic WTG (i.e. that does not contain cyclic path), then for all locations q , the piecewise affine mapping $\nu \mapsto \text{Val}_{\mathcal{G}}(q, \nu)$ is computable in time polynomial in $|Q|$ and W .

In [13], this result is slightly extended to take into account cyclic paths containing resets when their weight is either non-negative, or not arbitrarily close to 0.

► **Example 9.** Notice that the game on the left in Figure 1 does not fulfil this hypothesis: indeed the play $(q_0, 0) \xrightarrow{1/2-\varepsilon, \delta_1} (q_1, 1/2-\varepsilon) \xrightarrow{1/2+\varepsilon, \delta_2} (q_0, 0)$ is a cyclic play of weight -2ε negative and arbitrarily close to 0.

2.3 Contribution

In this work, we use a different technique to push the decidability frontier, and prove that the value function is computable for all WTG (in particular the one of Figure 1):

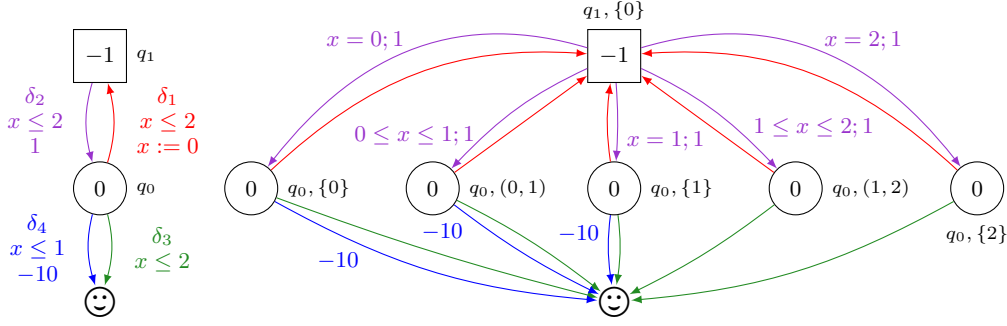
► **Theorem 10.** *For all WTG \mathcal{G} and all locations q_i , the mapping $\nu \mapsto \text{Val}_{\mathcal{G}}(q_i, \nu)$ is computable in time exponential in $|Q|$ and W_{tr} , and polynomial in W_{loc} and W_{fin} .*

► **Remark 11.** The complexities of Theorems 8 and 10 would be more traditionally considered as exponential and doubly-exponential if weights of the WTG were encoded in binary as usual. In this work, we thus count the complexities as if all weights were encoded in unary and thus consider W to be the bound of interest. For Theorem 8, the obtained bound is classically called *pseudo-polynomial* in the literature.

The rest of this article gives the proof of Theorem 10. We fix a WTG \mathcal{G} and an initial location q_i . We let $\bar{\mathcal{G}} = \langle L_{\text{Min}}, L_{\text{Max}}, L_t, L_u, \bar{\Delta}, \bar{\text{wt}}, \bar{\text{wt}}_t \rangle$ be its closure. We first use Lemma 6 which allows us to deduce the result by computing the value functions $\nu \mapsto \text{Val}_{\bar{\mathcal{G}}}((q_i, I), \nu)$, for all regions I . Regions I over which $\nu \mapsto \text{Val}_{\bar{\mathcal{G}}}((q_i, I), \nu)$ is constantly equal to $+\infty$ or $-\infty$ are computable in polynomial-time, as explained in [12]. We, therefore, remove them from $\bar{\mathcal{G}}$ from now on. We now fix an initial region I_i and let $\ell_i = (q_i, I_i)$.

As in the non-negative case [10], the objective is to limit the number of transitions with a reset taken into the plays while not modifying the value of the game. When all weights are non-negative, this is fairly easy to achieve since, intuitively speaking, **Min** has no interest in using any cycles containing such a transition (since it has non-negative weight and is thus non-beneficial for **Min**). The game can thus be transformed so that each transition with a reset is taken at most once. To obtain a smaller game, it is even possible to simply count the number of transitions with a reset taken so far in the play and stop the game (with a final weight $+\infty$) in case the counter goes above the number of such transitions in the game. The transformed game has a polynomial number of locations with respect to the original game, and is reset-acyclic, which allows one to solve it with a pseudo-polynomial time complexity (instead of the exponential-time complexity originally achieved in [10, 22]).

The situation is much more intricate in the presence of negative weights since negative cycles containing a transition with a reset can be beneficial for **Min**, as we have seen in Example 3. Notice that this is still true in the closure of the game, as can be checked on the right in Figure 1. Moreover, some cyclic paths may have an interval of possible weights with both positive and negative values, making it difficult to determine whether it is beneficial to **Min** or not. To overcome this situation, we will consider the point of view of **Max**, making a profit from the determinacy of the WTG. We will show that, in a closed game $\bar{\mathcal{G}}$, **Max** can play *optimally* with *memoryless* strategies while *avoiding negative cyclic plays*. This will simplify our further study since, by following this strategy, **Max** ensures that only non-negative cyclic plays will be encountered, which is not beneficial to **Min**. Therefore, as in [10], we will limit the firing of transitions with a reset to at most once. However, we are not able to do it without blowing up exponentially the number of locations of the games. Instead, along the unfolding of the game, we need to record enough information in order to know, in case a cyclic path ending with a reset is closed, whether this cyclic path has a potential negative weight (in which case **Max** will indeed not follow it) or non-negative weight (in which case it is not beneficial for **Min** to close the cycle). Determining in which case we are will be made possible by introducing the notion of value of a cyclic path in Section 3. Then, **Max** has even an optimal strategy to avoid closing cyclic paths with negative value (which is stronger than only avoiding creating negative cyclic plays). The unfolding, denoted \mathcal{U} , will be defined in Section 4. In order to prove that it is a game equivalent to $\bar{\mathcal{G}}$, we will prove that **Max** can do as well as in \mathcal{U} from $\bar{\mathcal{G}}$ and vice-versa.



■ **Figure 2** On the left, a WTG where Max needs memory to play ε -optimally. On the right, its closure where we merged several transitions by removing unnecessary guards.

3 Controlling negative cycles

One of the main arguments of our proof is that, in the closed game $\bar{\mathcal{G}}$, Max can play *optimally* with *memoryless* strategies while *avoiding negative cyclic plays*. As already noticed in [12], this is not always true in non-closed games: Max may need memory to play ε -optimally without the possibility to avoid some negative cyclic plays.

► **Example 12.** In the WTG \mathcal{G} depicted on the left in Figure 2, we can see that $\text{Val}(q_1, 0) = 0$, but Max does not have an optimal strategy, needs memory to play ε -optimally, and cannot avoid negative cyclic plays. Indeed, if at some point the strategy of Max chooses a delay less than or equal to 1, then Min can always choose δ_4 , and the value of this strategy is -10 . Thus, an optimal strategy for Max always chooses a delay greater than 1. However, Max must choose a delay closer and closer to 1. Otherwise, if there exists $\beta > 0$ such that all delays chosen by the strategy are greater than $1 + \beta$, Min has a family of strategies with a value that will tend to $-\infty$ by staying longer and longer in the cycle with a weight at most $-\beta$. Thus, Max does not have an optimal strategy, and the ε -optimal strategy requires infinite memory to play with delays closer and closer to 1 (for instance, after the n th round in the cycle, Max delays $\varepsilon/2^n$ time units, to sum up, all weights to a value at most $-\varepsilon$).

Such convergence phenomena needed by Max do not exist in $\bar{\mathcal{G}}$ since all guards are closed (this is not sufficient alone though) and by the regularity of Val given by Lemma 7.

► **Example 13.** We consider the closed game depicted on the right in Figure 2. The ε -optimal strategy (with memory) of Max in \mathcal{G} translates into an optimal memoryless strategy in $\bar{\mathcal{G}}$: in $(q_1, \{0\})$, Max can delay 1 time unit and jump into the location $(q_0, (1, 2))$. Then cyclic plays that Min can create have a zero weight and are thus not profitable for either player.

To generalise this explanation, we start by defining the value of cyclic paths ending with a reset in $\bar{\mathcal{G}}$. Intuitively, the value of this cyclic path is the weight that Min (or Max) can guarantee regardless of the delays chosen by Max (or Min) during this one.

► **Definition 14.** We define by induction the value $\text{Val}_{\bar{\mathcal{G}}}^{\nu}(\pi)$ of a finite path π in $\bar{\mathcal{G}}$ from an initial valuation ν of the clock: if π has length 0, we let $\text{Val}_{\bar{\mathcal{G}}}^{\nu}(\pi) = 0$, otherwise, π can be written $\ell_0 \xrightarrow{\delta_0} \pi'$ (with π' starting in location ℓ_1), and we let

$$\text{Val}_{\bar{\mathcal{G}}}^{\nu}(\pi) = \begin{cases} \inf_{t_0} (t_0 \text{wt}(\ell_0) + \text{wt}(\delta_0) + \text{Val}_{\bar{\mathcal{G}}}^{\nu'}(\pi')) & \text{if } \ell_0 \in L_{\text{Min}} \\ \sup_{t_0} (t_0 \text{wt}(\ell_0) + \text{wt}(\delta_0) + \text{Val}_{\bar{\mathcal{G}}}^{\nu'}(\pi')) & \text{if } \ell_0 \in L_{\text{Max}} \end{cases}$$

where t_0 and ν' are such that $(\ell_0, \nu) \xrightarrow{t_0, \delta_0} (\ell_1, \nu')$ is an edge of $\llbracket \bar{\mathcal{G}} \rrbracket$. Then, for a cyclic path π of $\bar{\mathcal{G}}$ ending by a transition with a reset, we let $\text{Val}_{\bar{\mathcal{G}}}(\pi) = \text{Val}_{\bar{\mathcal{G}}}^0(\pi)$.

15:10 Decidability of One-Clock Weighted Timed Games with Arbitrary Weights

The value of a cyclic path belongs to the interval $\text{wt}_\Sigma(\pi)$ and corresponds to the weight of a cyclic play that follows this path.

► **Example 15.** Let $\pi = (q_0, \{0\}) \xrightarrow{\delta_1} (q_1, (0, 1)) \xrightarrow{\delta_2} (q_0, \{0\})$ be the cyclic path of the game $\bar{\mathcal{G}}$ depicted on the right in Figure 1, for which $\text{wt}_\Sigma(\pi) = [-1, 1]$. To evaluate the value of π , Min only needs to choose a delay $t_1 \in [0, 1]$ when firing δ_1 , while Max has no choice but to play a delay $1 - t_1$ when firing δ_2 , generating a finite play ρ of weight $\text{wt}_\Sigma(\rho) = 2t_1 - 1$. We deduce that $\text{Val}_{\bar{\mathcal{G}}}(\pi) = \inf_{t_1} (2t_1 - 1) = -1$ (when Min chooses $t_1 = 0$).

A cyclic path with a negative value ensures that Min can guarantee a cyclic play with a negative weight that follows it, but there may exist other cyclic plays with a non-negative weight that follows it. It is exactly those cycles that are problematic for Max since Min can benefit from them. We now show our key lemma: in the closed game, Max can play optimally and avoid cyclic paths of negative value.

► **Lemma 16.** *In $\bar{\mathcal{G}}$ (where regions with infinite value had been remote), Max has a memoryless optimal strategy τ^* such that*

1. *all cyclic plays conforming to τ^* have a non-negative weight;*
2. *all cyclic paths ending by a reset conforming to τ^* have a non-negative value.*

Sketch of proof. We build upon the fact [8, 7] that the value function $\text{Val}_{\bar{\mathcal{G}}}: L \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$ is a fixed point (even the greatest one) of the operator \mathcal{F} defined as follows: for all configurations (ℓ, ν) and all mappings $X: L \times \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$, we let

$$\mathcal{F}(X)(\ell, \nu) = \begin{cases} \overline{\text{wt}}_t(\ell, \nu) & \text{if } \ell \in L_t \\ \inf_{\ell, \nu \xrightarrow{t, \delta} \ell', \nu'} (\overline{\text{wt}}(\delta) + t \overline{\text{wt}}(\ell) + X(\ell', \nu')) & \text{if } \ell \in L_{\text{Min}} \\ \sup_{\ell, \nu \xrightarrow{t, \delta} \ell', \nu'} (\overline{\text{wt}}(\delta) + t \overline{\text{wt}}(\ell) + X(\ell', \nu')) & \text{if } \ell \in L_{\text{Max}} \end{cases}$$

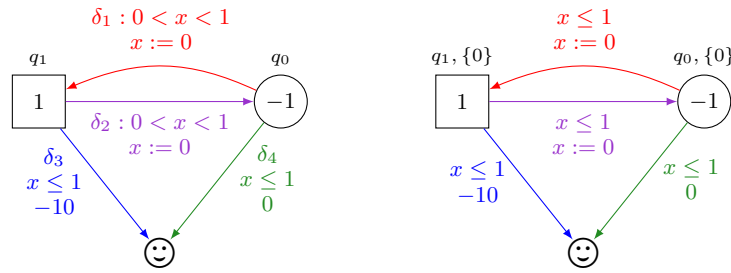
We use this fact to define the memoryless strategy τ^* . Indeed, the identity $\text{Val}_{\bar{\mathcal{G}}} = \mathcal{F}(\text{Val}_{\bar{\mathcal{G}}})$, applied over configurations belonging to Max suggests a choice of transition and delay to play almost optimally. As \mathcal{F} computes a supremum on the set of possible (transitions and) delays, this does not directly lead to a specific choice: in general, this would give rise to ε -optimal strategies and not an optimal one. This is where we rely on the continuity of $\text{Val}_{\bar{\mathcal{G}}}$ (Lemma 7) on each closure of region to deduce that this supremum is indeed a maximum. More precisely, for $\ell \in L_{\text{Max}}$, we can write $\mathcal{F}(\text{Val}_{\bar{\mathcal{G}}})(\ell, \nu)$ as

$$\max_{\delta \in \bar{\Delta}} \sup_{t \text{ s.t. } \ell, \nu \xrightarrow{t, \delta} \ell', \nu'} (\overline{\text{wt}}(\delta) + t \overline{\text{wt}}(\ell) + \text{Val}_{\bar{\mathcal{G}}}(\ell', \nu')).$$

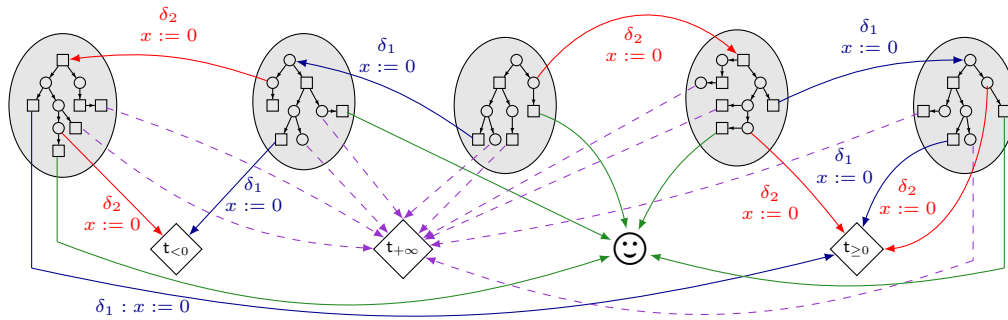
The guard of transition δ is the closure \bar{I} of a region $I \in \text{Reg}_{\mathcal{G}}$, therefore, t is in a closed interval J of values such that $\nu + t$ falls in \bar{I} . Notice that ν' is either 0 if δ contains a reset or is $\nu + t$: in both cases, this is a continuous function of t . Relying on the continuity of $\text{Val}_{\bar{\mathcal{G}}}$, the mapping $t \in J \mapsto \overline{\text{wt}}(\delta) + t \overline{\text{wt}}(\ell) + \text{Val}_{\bar{\mathcal{G}}}(\ell', \nu')$ is thus continuous over a compact set so that its supremum is indeed a maximum. We thus let the memoryless strategy τ^* be such that, for all configurations (ℓ, ν) , $\tau^*(\ell, \nu)$ is chosen arbitrarily in

$$\arg\max_{\delta \in \bar{\Delta}} \arg\max_{t \text{ s.t. } \ell, \nu \xrightarrow{t, \delta} \ell', \nu'} (\overline{\text{wt}}(\delta) + t \overline{\text{wt}}(\ell) + \text{Val}_{\bar{\mathcal{G}}}(\ell', \nu')) \quad (1)$$

The strategy τ^* is then extended to finite plays by considering only the last configuration of the play. We can show that τ^* is an optimal strategy that satisfies the two properties of the lemma. ◀



■ **Figure 3** On the left, a WTG such that its closure on the right contains a cyclic path with a weight $[-1, 1]$ and a value 0. Moreover Max uses the cyclic path to play optimally.



■ **Figure 4** Scheme of the unfolding of a closed game.

Lemma 16 does not allow us to conclude on the decidability of the value problem since we use the unknown value $\text{Val}_{\overline{\mathcal{G}}}$ to define the optimal strategy. However, it will help us in the final step of the proof (see Appendix A).

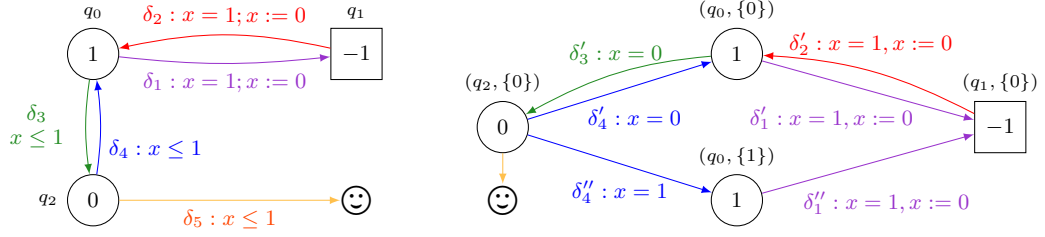
As a side note, it is tempting to strengthen Lemma 16-2 so as to ensure that all cyclic paths ending by a reset conforming to τ^* have a non-negative weight (and not only the value), i.e. an interval of weights entirely included in $[0, +\infty)$. Unfortunately, this does not hold, as shown in the following example:

▶ **Example 17.** We consider the closed game depicted on the right in Figure 3. Let $\pi = (q_0, \{0\}) \xrightarrow{\delta_1} (q_1, \{0\}) \xrightarrow{\delta_2} (q_0, \{0\})$ be the cyclic path for which $\text{wt}_{\Sigma}(\pi) = [-1, 1]$. To evaluate the value of π , Min and Max need to choose delays $t_1, t_2 \in [0, 1]$ when firing δ_1 and δ_2 . We obtain a set of finite plays ρ parametrised by t_1 and t_2 of weight $\text{wt}_{\Sigma}(\rho) = -t_1 + t_2$. We deduce that $\text{Val}_{\mathcal{G}}(\pi) = \inf_{t_1} \sup_{t_2} (t_2 - t_1) = 0$ (when Min and Max choose $t_1 = t_2 = 1$). The optimal strategy of Max does not use the transition δ_3 and is thus forced to play in the previous cyclic path (with a non-negative value but with a negative weight): from the configuration $((q_0, \{0\}), 0)$, Min has thus no other choice than playing transition δ_4 after a delay of 1 unit of time, leading to a value of -1 .

4 Unfolding

We now define the partial *unfolding* of the game $\overline{\mathcal{G}}$ that we need in order to compute $\text{Val}_{\overline{\mathcal{G}}}$, stopping the unfolding when too many transitions with a reset have been taken or when the play is too long since the last reset. About the transitions with a reset, when such a transition is taken for the first time, we go into anew copy of the game, from which, if this

15:12 Decidability of One-Clock Weighted Timed Games with Arbitrary Weights



■ **Figure 5** A WTG (left), and a portion of its closure (right) where δ'_2 belongs to a cyclic path of non-negative value and another cyclic path of negative value.

transition happens to be chosen one more time, we stop the game by jumping into a new target location. The final weight of this target location is determined by the value of the cyclic path (ending with a reset) that would have just been closed. If the cyclic path has a negative value, then we go in a leaf $t_{<0}$ of final weight $-\infty$ since this is a desirable cycle for Min. Otherwise, we go in a leaf $t_{\geq 0}$ of final weight big enough $|L|(W_{\text{tr}} + M W_{\text{loc}}) + W_{\text{fin}}$ (for technical reasons that will become clear later, we cannot simply put a final weight $+\infty$) so as it remains an undesirable behaviour for Min.

A single transition with a reset can be part of two distinct cyclic paths, one of negative value and the other of non-negative value, as demonstrated in the following example:

► **Example 18.** In Figure 5, we have depicted a WTG (left) and a portion of its closure (right), where δ'_2 is contained in a cyclic path of negative value

$$(q_0, \{0\}) \xrightarrow{\delta'_3} (q_2, \{0\}) \xrightarrow{\delta''_4} (q_0, \{1\}) \xrightarrow{\delta''_1} (q_1, \{0\}) \xrightarrow{\delta'_2} (q_0, \{0\})$$

and another cyclic path of non-negative (zero) value

$$(q_0, \{0\}) \xrightarrow{\delta'_1} (q_1, \{0\}) \xrightarrow{\delta'_2} (q_0, \{0\}).$$

Thus, knowing the last transition of the cycle is not enough to compute the value of the cyclic path. Instead, we need to record the whole cyclic path: copying the game (as done in the non-negative setting [10]) is not enough, our unfolding needs to remember the path followed so far. Locations of the unfolding are thus finite paths of $\bar{\mathcal{G}}$.

In order to obtain an acyclic unfolding, we will rely on a property of *reset-acyclic* WTG, i.e that do not contain cyclic paths with a transition with a reset. For such WTG, [13] shows the existence of an ε -optimal strategy for Min with a particular shape. This strategy is also called as *switching strategy* [14], as defined by the following:

► **Definition 19.** A switching strategy σ is described by two memoryless strategies σ^1 and σ^2 , as well as a switching threshold κ' . The strategy σ then consists in playing strategy σ^1 until either we reach a target location or the finite play has a length of at least κ' , in which case we switch to strategy σ^2 .

Intuitively, σ^1 aims at reaching a cyclic play with negative weight, while σ^2 is an attractor to the target. As a consequence, we can estimate the maximal number of steps needed by σ^2 to reach the target. Combining this with the switching threshold κ' we can deduce a threshold κ that upper bounds the number of steps under the switching strategy σ to reach the target. We obtain the following result with an explicit bound κ given by the previous work of [13]. From a combination of their Lemma 5 and Theorem 25, we know that the switching threshold κ' is in

$$O(|L| \times [W_{\text{loc}} + W_{\text{tr}}^4 |L|^9 \times |L| W_{\text{tr}} + W_{\text{tr}}^4 |L|^9]) = O(|L|^{11} (W_{\text{loc}} + W_{\text{tr}}^4))$$

■ **Algorithm 1** Function NEXT that maps pairs $(\pi, \delta) \in \text{FPaths}_{\overline{\mathcal{G}}} \times \overline{\Delta}$ to pairs (π', δ') composed of a finite path π' of $\overline{\mathcal{G}}$ (or $t_{\geq 0}$, or $t_{< 0}$, or $t_{+\infty}$) and a new transition δ' of the unfolding \mathcal{U} .

```

1: function NEXT( $\pi, \delta = \ell_1 \xrightarrow{I, R, w} \ell_2$ ): ▷ last( $\pi$ ) =  $\ell_1$ 
2:   if  $\ell_2 \in L_t$  then  $\pi' := \ell_2$ 
3:   else if  $R = \{x\}$  then
4:     if  $|\pi|_\delta = 0$  then  $\pi' := \pi \cdot \delta$ 
5:     else { let  $\pi = \pi_1 \cdot \delta \cdot \pi_2$ 
6:       if  $\text{Val}_{\overline{\mathcal{G}}}(\pi_2 \cdot \delta) \geq 0$  then  $\pi' := t_{\geq 0}$  else  $\pi' := t_{< 0}$  }
7:   else { let  $\pi = \pi_1 \cdot \pi_2$  where  $\pi_2$  contains no reset and  $|\pi_2|$  is maximal
8:     if  $|\pi_2| = \kappa$  then  $\pi' := t_{+\infty}$  else  $\pi' := \pi \cdot \delta$  }
9:    $\delta' := \pi \xrightarrow{I, R, w} \pi'$  ▷  $\Delta\text{proj}(\delta') := \delta$ 
10:  return  $(\pi', \delta')$ 

```

Then, we let κ'' be the number of turns taken by σ^2 to reach the target location, which is polynomial in the number of locations of the underlying region automaton, thus polynomial in the number of locations of the game (since there is only one-clock). Overall, this gives a definition for κ as

$$\kappa = \kappa' + \kappa'' = O(|L|^{12}(W_{\text{loc}} + W_{\text{tr}}^4))$$

that is polynomial in $|Q|$ (as $|L|$ is polynomial in $|Q|$) and in W .

► **Lemma 20** ([13]). *Let \mathcal{G} be a reset-acyclic WTG. Min has an ε -optimal switching strategy σ such that all plays conforming to σ reach the target within κ steps. Moreover, κ is polynomial in $|Q|$ and W .*

As a consequence, assuming that Min plays almost optimally using a switching strategy, we can bound the number of steps between two transitions with a reset by κ . This property allows us to avoid incorporating cycles in the unfolding: we cut the unfolding when the play becomes longer than κ since the last seen transition with a reset. In this case, we will jump into a new target location $t_{+\infty}$ whose final weight is equal to $+\infty$ since it is an undesirable behaviour for Min.

The scheme of the unfolding is depicted in Figure 4 when the closed game $\overline{\mathcal{G}}$ contains two transitions with a reset, δ_1 and δ_2 , each belonging to several cycles of different values (negative and non-negative). Inside each grey component, transitions with no reset are unfolded for κ steps by only keeping in the current location the path followed so far. In-between the components are transitions with a reset. The second time they are visited, the value of the cycle it closes is computed, and we jump in $t_{< 0}$ or $t_{\geq 0}$ depending on the sign of the value.

► **Definition 21.** *The unfolding of $\overline{\mathcal{G}}$ from the initial location ℓ_i is the (a priori infinite) WTG $\mathcal{U} = \langle L'_{\text{Min}}, L'_{\text{Max}}, L'_t, L'_u, \Delta', \text{wt}', \text{wt}'_t \rangle$ with $L'_{\text{Min}} \subseteq \text{FPaths}_{\text{Min}}$, $L'_{\text{Max}} \subseteq \text{FPaths}_{\text{Max}}$, $L'_t \subseteq L_t \cup \{t_{\geq 0}, t_{< 0}, t_{+\infty}\}$ such that*

- $L' = L'_{\text{Min}} \uplus L'_{\text{Max}} \uplus L'_t$ and Δ' are the smallest sets such that $\ell_i \in L'$ and for all $\pi \in L'_{\text{Min}} \uplus L'_{\text{Max}}$ and $\delta \in \Delta$, if $\text{NEXT}(\pi, \delta) = (\pi', \delta')$ then $\pi' \in L'$ and $\delta' \in \Delta'$ (where NEXT is defined in Algorithm 1);
- $L'_u = \{\pi \in L' \mid \text{last}(\pi) \in L_u\}$;
- for all $\pi \notin L'_t$, $\text{wt}'(\pi) = \overline{\text{wt}}(\text{last}(\pi))$;
- for all $\pi \in L'_t$, for all ν ,

$$\begin{aligned} \text{wt}'_t(\pi, \nu) &= \overline{\text{wt}}_t(\pi, \nu) & \text{if } \pi \in L_t & & \text{wt}'_t(t_{\geq 0}, \nu) &= |L|(W_{\text{tr}} + M W_{\text{loc}}) + W_{\text{fin}} \\ \text{wt}'_t(t_{< 0}, \nu) &= -\infty & & & \text{wt}'_t(t_{+\infty}, \nu) &= +\infty. \end{aligned}$$

15:14 Decidability of One-Clock Weighted Timed Games with Arbitrary Weights

A target location is reached when the length between two resets is too long or when a transition with a reset would appear two times. Moreover, the length of a path in the location that is not a target, given by the application of NEXT, strictly increases. This allows us to show that \mathcal{U} is a *finite* and acyclic WTG as expected.

► **Lemma 22.** *\mathcal{U} is an acyclic WTG with a finite set of locations of cardinality exponential in $|Q|$ and W_{tr} .*

Furthermore, in \mathcal{U} , as we showed in $\bar{\mathcal{G}}$, in Lemma 16, Max can play optimally with a memoryless strategy. Note that, unlike in $\bar{\mathcal{G}}$, there exist no cyclic paths in \mathcal{U} : however, we can check the positivity of the “cyclic plays” in-between two occurrences of the same transition containing a reset when we jump in $\mathbf{t}_{\geq 0}$.

► **Lemma 23.** *In \mathcal{U} , Max has a memoryless optimal strategy τ^* such that if $\rho = \rho_1 \xrightarrow{t_1, \delta'_1} \rho_2 \xrightarrow{t_2, \delta'_2} (\mathbf{t}_{\geq 0}, 0)$ is conforming to τ^* with $\Delta \text{proj}(\delta'_1) = \Delta \text{proj}(\delta'_2)$ a transition with a reset of x , then $\text{wt}_{\Sigma}(\rho_2 \xrightarrow{t_2, \delta'_2} (\mathbf{t}_{\geq 0}, 0)) \geq 0$.*

The property on the weight of plays that reach $\mathbf{t}_{\geq 0}$ is guaranteed by the structure of \mathcal{U} . Indeed, as \mathcal{U} is acyclic, we know that the value of the path followed by a play ending in $\mathbf{t}_{\geq 0}$ is non-negative. That would no longer be the case if we would have defined \mathcal{U} with grey components containing cyclic paths without reset, since the value of cyclic path do not compose, as demonstrated by the following example.

► **Example 24.** In the WTG $\bar{\mathcal{G}}$ depicted in Figure 5, we can see that $\text{Val}((q_0, \{0\}) \xrightarrow{\delta_1} (q_1, \{0\}) \xrightarrow{\delta_2} (q_0, \{0\})) = 0$: Min and Max must delay 1 in each location, and $\text{Val}^0((q_0, \{0\}) \xrightarrow{\delta_3} (q_2, \{0\}) \xrightarrow{\delta_4} (q_0, \{1\})) = 0$. However, when we composed these two cyclic path, we obtain that $\text{Val}((q_0, \{0\}) \xrightarrow{\delta_3} (q_2, \{0\}) \xrightarrow{\delta_4} (q_0, \{1\}) \xrightarrow{\delta_1} (q_1, \{0\}) \xrightarrow{\delta_2} (q_0, \{0\})) = -1$.

Now, as in Lemma 16, τ^* is defined with argmax on transitions and delays. Thus, to obtain a play ending in $\mathbf{t}_{\geq 0}$ with a non-negative weight, we constrain Max to play the value of the cycle that reached $\mathbf{t}_{\geq 0}$ by assigning it a finite final weight.

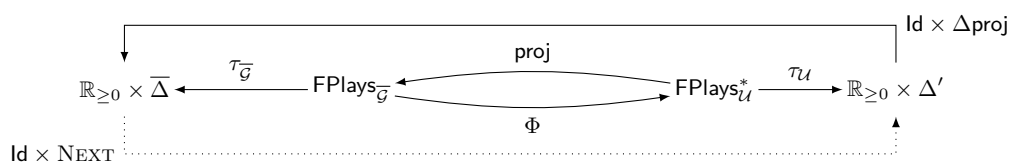
Finally, the most difficult part of the proof is to show that the unfolding preserves the value. Remember that we have fixed an initial location $\ell_i = (q_i, I_i)$ to build \mathcal{U} .

► **Theorem 25.** *For all $\nu \in I_i$, $\text{Val}_{\bar{\mathcal{G}}}(\ell_i, \nu) = \text{Val}_{\mathcal{U}}(\ell_i, \nu)$.*

Before proving Theorem 25, we show how this helps prove our main result.

Proof of Theorem 10. Remember (by Lemma 6) that we only need to explain how to compute $\nu \mapsto \text{Val}_{\bar{\mathcal{G}}}((q_i, I_i), \nu)$ over I_i . By Theorem 25, this is equivalent to computing $\nu \mapsto \text{Val}_{\mathcal{U}}((q_i, I_i), \nu)$ over I_i . We now explain why this is doable.

First, the definition of \mathcal{U} is effective: we can compute it entirely, making use of Lemma 22 showing that it is a finite WTG. The only non-trivial part is the determination of $\text{Val}_{\bar{\mathcal{G}}}(\pi_2 \cdot \delta)$ in Algorithm 1 to determine in which target location we jump. Since $\pi_2 \cdot \delta$ is a finite path, we can apply Theorem 8 to compute the value of the corresponding game, which is exactly the value $\text{Val}_{\bar{\mathcal{G}}}(\pi_2 \cdot \delta)$. The complexity of computing the value of a path is polynomial in the length of this path (that is exponential in $|Q|$ and W_{tr} , by Lemma 22) and polynomial in $|Q|$ and W (notice that weights of $\bar{\mathcal{G}}$ are the same as the ones in \mathcal{G}): this is thus of complexity exponential in $|Q|$ and W_{tr} , and polynomial in W_{loc} and W_{fin} . Since \mathcal{U} has an exponential number of locations with respect to $|Q|$ and W_{tr} , the total time required to compute \mathcal{U} is exponential with respect to $|Q|$ and W_{tr} , and polynomial with respect to W_{loc} and W_{fin} .



■ **Figure 6** Scheme showing the links between the different objects defined for the proof of Theorem 25 where $\text{FPlays}_{\mathcal{U}}^*$ is the set of finite plays of \mathcal{U} avoiding target locations $t_{\geq 0}$ and $t_{< 0}$.

Lemma 22 ensures that \mathcal{U} is acyclic, so we can apply Theorem 8 to compute the value mapping $\nu \mapsto \text{Val}_{\mathcal{U}}((q_i, I_i), \nu)$ as a piecewise affine and continuous function. It requires a complexity polynomial in the number of locations of \mathcal{U} , and in W_{loc} , W_{tr} , and W_{fin} (since weights of \mathcal{U} all come from \mathcal{G}). Knowing the previous bound on the number of locations of \mathcal{U} , this complexity translates into an exponential time complexity with respect to $|Q|$ and W_{tr} , and polynomial with respect to W_{loc} and W_{fin} . ◀

The proof of Theorem 25 splits into two inequalities. We prove in Appendix A that $\text{Val}_{\bar{\mathcal{G}}}(\ell_i, \nu) \leq \text{Val}_{\mathcal{U}}(\ell_i, \nu)$, i.e. that Max can guarantee to always do at least as good in \mathcal{U} as in $\bar{\mathcal{G}}$. We thus show that for an optimal strategy $\tau_{\bar{\mathcal{G}}}$ in $\bar{\mathcal{G}}$ (defined by Lemma 16), there exists a strategy $\tau_{\mathcal{U}}$ in \mathcal{U} such that for all plays ρ conforming to $\tau_{\mathcal{U}}$, there exists a play conforming to $\tau_{\bar{\mathcal{G}}}$ with a weight at most the weight of ρ . As it is depicted in Figure 6, the strategy $\tau_{\mathcal{U}}$ is defined via a *projection* of plays of \mathcal{U} in $\bar{\mathcal{G}}$: we use the mapping NEXT to send back transitions of $\bar{\Delta}$ to Δ' .

We then prove in Appendix B that $\text{Val}_{\bar{\mathcal{G}}}(\ell_i, \nu) \geq \text{Val}_{\mathcal{U}}(\ell_i, \nu)$, i.e. Max can guarantee to always do at least as good in $\bar{\mathcal{G}}$ as in \mathcal{U} . We thus show that for an optimal strategy $\tau_{\mathcal{U}}$ in \mathcal{U} (defined by Lemma 23), there exists a strategy $\tau_{\bar{\mathcal{G}}}$ in $\bar{\mathcal{G}}$ such that for the unique play ρ conforming to $\tau_{\bar{\mathcal{G}}}$ and the *switching* strategy (see Definition 19), there exists a play conforming to $\tau_{\mathcal{U}}$ with a weight at most the weight of ρ . As depicted in Figure 6, the strategy $\tau_{\bar{\mathcal{G}}}$ is defined via a function Φ that puts plays of $\bar{\mathcal{G}}$ in \mathcal{U} . Intuitively, this function removes all cyclic plays ending with a reset from plays in $\bar{\mathcal{G}}$.

5 Conclusion

We solve one-clock WTG with arbitrary weights, an open problem for several years. We strongly rely on the determinacy of the game, taking the point of view of Max, instead of the one of Min as was done in previous work with only non-negative weights. We also use technical ingredients such as the closure of a game, switching strategies for Min, and acyclic unfoldings. Regarding the complexity, our algorithm runs in exponential time (with weights encoded in unary), which does not match the known PSPACE lower bound with weights in binary [16]. Observe that this lower bound only uses non-negative weights. This complexity gap deserves further study. Our work also opens three research directions. First, as we unfold the game into a finite tree, it would be interesting to develop a symbolic approach that shares computation between subtrees in order to obtain a more efficient algorithm. Second, playing stochastically in WTG with shortest path objectives has been recently studied in [20]. One could study an extension of one-clock WTG with stochastic transitions. In this context, Min aims at minimizing the expectation of the accumulated weight. Third, the analysis of cycles that we have done in the setting of one-clock WTG can be an inspiration to identify new decidable classes of WTG with arbitrarily many clocks.

References

- 1 Rajeev Alur, Mikhail Bernadsky, and P. Madhusudan. Optimal reachability for weighted timed games. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP'04)*, volume 3142 of *LNCS*, pages 122–133. Springer, 2004. doi:10.1007/978-3-540-27836-8_13.
- 2 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.
- 3 Eugene Asarin and Oded Maler. As soon as possible: Time optimal control for timed automata. In *Hybrid Systems: Computation and Control*, volume 1569 of *LNCS*, pages 19–30. Springer, 1999. doi:10.1007/3-540-48983-5_6.
- 4 Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. Uppaal-tiga: Time for playing games! In Werner Damm and Holger Hermanns, editors, *Proceedings of the 19th International Conference on Computer Aided Verification (CAV 2007)*, volume 4590 of *LNCS*, pages 121–125. Springer, 2007. doi:10.1007/978-3-540-73368-3_14.
- 5 Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced time automata. In *International Workshop on Hybrid Systems: Computation and Control*, pages 147–161. Springer, 2001. doi:10.1007/3-540-45351-2_15.
- 6 Béatrice Bérard, Antoine Petit, Volker Diekert, and Paul Gastin. Characterization of the expressive power of silent transitions in timed automata. *Fundamenta Informaticae*, 36(2-3):145–182, 1998. doi:10.3233/FI-1998-36233.
- 7 Patricia Bouyer. Erratum to the FSTTCS'04 paper “optimal strategies in priced timed game automata”. Personal Communication, 2016.
- 8 Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen. Optimal strategies in priced timed game automata. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science*, pages 148–160, Berlin, Heidelberg, 2005. Springer. doi:10.1007/978-3-540-30538-5_13.
- 9 Patricia Bouyer, Samy Jaziri, and Nicolas Markey. On the value problem in weighted timed games. In *Proceedings of the 26th International Conference on Concurrency Theory (CONCUR'15)*, volume 42 of *Leibniz International Proceedings in Informatics*, pages 311–324. Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CONCUR.2015.311.
- 10 Patricia Bouyer, Kim G. Larsen, Nicolas Markey, and Jacob Illum Rasmussen. Almost optimal strategies in one clock priced timed games. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*, pages 345–356, Berlin, Heidelberg, 2006. Springer. doi:10.1007/11944836_32.
- 11 Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In Paul Pettersson and Wang Yi, editors, *Formal Modeling and Analysis of Timed Systems*, pages 49–64, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. doi:10.1007/11603009_5.
- 12 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, Engel Lefaucheux, and Benjamin Monmege. Simple priced timed games are not that simple. In *Proceedings of the 35th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'15)*, volume 45 of *LIPIcs*, pages 278–292. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.FSTTCS.2015.278.
- 13 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, Engel Lefaucheux, and Benjamin Monmege. One-clock priced timed games with negative weights. Research Report 2009.03074, arXiv, 2021.
- 14 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.

- 15 Damien Busatto-Gaston, Benjamin Monmege, and Pierre-Alain Reynier. Optimal reachability in divergent weighted timed games. In *Proceedings of the 20th International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2017)*, LNCS, pages 162–178. Springer, 2017. doi:10.1007/978-3-662-54458-7_10.
- 16 John Fearnley, Rasmus Ibsen-Jensen, and Rahul Savani. One-clock priced timed games are PSPACE-hard. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Sciences (LICS'20)*, pages 397–409. ACM, 2020. doi:10.1145/3373718.3394772.
- 17 Thomas Dueholm Hansen, Rasmus Ibsen-Jensen, and Peter Bro Miltersen. A faster algorithm for solving one-clock priced timed games. In Pedro R. D'Argenio and Hernán C. Melgratti, editors, *Proceedings of the 24th International Conference on Concurrency Theory (CONCUR'13)*, volume 8052 of *Lecture Notes in Computer Science*, pages 531–545. Springer, 2013. doi:10.1007/978-3-642-40184-8_37.
- 18 Marcin Jurdziński and Ashutosh Trivedi. Reachability-time games on timed automata. In *Proceedings of the 34th International Colloquium on Automata, Languages and Programming (ICALP'07)*, volume 4596 of *LNCS*, pages 838–849. Springer, 2007.
- 19 François Laroussinie, Nicolas Markey, and Philippe Schnoebelen. Model checking timed automata with one or two clocks. In *Proceedings of CONCUR'04*, pages 387–401, 2004. doi:10.1007/978-3-540-28644-8_25.
- 20 Benjamin Monmege, Julie Parreaux, and Pierre-Alain Reynier. Playing Stochastically in Weighted Timed Games to Emulate Memory. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *LIPICs*, pages 137:1–137:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPICs.ICALP.2021.137.
- 21 Benjamin Monmege, Julie Parreaux, and Pierre-Alain Reynier. Decidability of one-clock weighted timed games with arbitrary weights. Research Report 2207.01608, arXiv, 2022. arXiv:2207.01608.
- 22 Michał Rutkowski. Two-player reachability-price games on single-clock timed automata. In *Proceedings of the Ninth Workshop on Quantitative Aspects of Programming Languages (QAPL'11)*, volume 57 of *EPTCS*, pages 31–46, 2011.

A Proof of $\text{Val}_{\bar{\mathcal{G}}}(\ell_i, \nu) \leq \text{Val}_{\mathcal{U}}(\ell_i, \nu)$

We show this first inequality by rewriting it $\text{Val}_{\bar{\mathcal{G}}}(\ell_i, \nu) \leq \sup_{\tau_{\mathcal{U}}} \text{Val}_{\mathcal{U}}^{\tau_{\mathcal{U}}}(\ell_i, \nu)$. Let $\tau_{\bar{\mathcal{G}}}$ be a memoryless optimal strategy of Max in $\bar{\mathcal{G}}$ satisfying the conditions of Lemma 16: in particular, $\text{Val}_{\bar{\mathcal{G}}}(\ell_i, \nu) = \text{Val}_{\bar{\mathcal{G}}}^{\tau_{\bar{\mathcal{G}}}}(\ell_i, \nu)$. To conclude, it is thus sufficient to build from $\tau_{\bar{\mathcal{G}}}$ a strategy $\tau_{\mathcal{U}}$ in \mathcal{U} such that

► **Proposition 26.** $\text{Val}_{\bar{\mathcal{G}}}^{\tau_{\bar{\mathcal{G}}}}(\ell_i, \nu) \leq \text{Val}_{\mathcal{U}}^{\tau_{\mathcal{U}}}(\ell_i, \nu)$

Following Figure 6, we use a projection operator to do so. It projects finite plays of \mathcal{U} starting in ℓ_i (since these are the only ones we need to take care of) to finite plays of $\bar{\mathcal{G}}$. For this reason, from now on, $\text{FPlays}_{\mathcal{U}}$ and $\text{FPlays}_{\bar{\mathcal{G}}}$ denote the subsets of plays that start in location ℓ_i . Moreover, we limit ourselves to projecting plays of \mathcal{U} that do not reach the targets $t_{<0}$ and $t_{\geq 0}$, since otherwise there is no canonical projection in $\bar{\mathcal{G}}$. Formally, we thus let $\text{FPlays}_{\mathcal{U}}^*$ be all such finite plays of $\text{FPlays}_{\mathcal{U}}$ that do not end in $t_{<0}$ or $t_{\geq 0}$. The projection function $\text{proj}: \text{FPlays}_{\mathcal{U}}^* \rightarrow \text{FPlays}_{\bar{\mathcal{G}}}$ is defined inductively on finite plays $\rho \in \text{FPlays}_{\mathcal{U}}^*$ by letting $\text{proj}(\rho)$ be

$$\left\{ \begin{array}{ll} (\ell_i, \nu) & \text{if } \rho = (\ell_i, \nu) \in L' \\ \text{proj}(\rho') \xrightarrow{t, \Delta \text{proj}(\delta')} (\text{last}(\pi), \nu) & \text{if } \rho = \rho' \xrightarrow{t, \delta'} (\pi, \nu) \\ \text{proj}(\rho') \xrightarrow{t, \Delta \text{proj}(\delta')} (\ell', \nu) & \text{if } \rho = \rho' \xrightarrow{t, \delta'} (t_{+\infty}, \nu) \text{ and } \Delta \text{proj}(\delta') = \ell \xrightarrow{I, R} \ell' \end{array} \right.$$

where $\Delta \text{proj}(\delta')$ is defined on line 9 of the NEXT function (see Algorithm 1). It fulfils the following properties:

15:18 Decidability of One-Clock Weighted Timed Games with Arbitrary Weights

► **Lemma 27.** For all plays $\rho \in \text{FPlays}_{\mathcal{U}}^*$ such that ρ does not end in $t_{+\infty}$,

1. if $\text{last}(\rho) = (\pi, \nu)$ then $\text{last}(\text{proj}(\rho)) = (\text{last}(\pi), \nu)$;
2. $\text{wt}_{\Sigma}(\rho) = \text{wt}_{\Sigma}(\text{proj}(\rho))$;
3. if $\text{last}(\rho) = (\pi, \nu)$ with $\pi \notin L_t$, then $\text{proj}(\rho)$ follows π .

Proof.

1. This is direct from a case analysis on the definition of proj .
2. We reason by induction on the length of $\rho \in \text{FPlays}_{\mathcal{U}}^*$. If $\rho = (\ell_i, \nu)$, then we have $\text{proj}(\rho) = \rho$, so $\text{wt}_{\Sigma}(\rho) = 0 = \text{wt}_{\Sigma}(\text{proj}(\rho))$. Now, we suppose that $\rho = \rho' \xrightarrow{t, \delta'} (\pi, \nu)$, with ρ' ending in location π' . Then

$$\text{wt}_{\Sigma}(\rho) = \text{wt}_{\Sigma}(\rho') + t \text{wt}'(\pi') + \text{wt}'(\delta')$$

This is equal to

$$\text{wt}_{\Sigma}(\rho') + t \overline{\text{wt}}(\text{last}(\pi')) + \overline{\text{wt}}(\Delta \text{proj}(\delta'))$$

since $\text{wt}'(\pi') = \overline{\text{wt}}(\text{last}(\pi'))$, and $\text{wt}'(\delta') = \overline{\text{wt}}(\Delta \text{proj}(\delta'))$ by definition of \mathcal{U} . By induction hypothesis, this implies that $\text{wt}_{\Sigma}(\rho)$ is equal to

$$\text{wt}_{\Sigma}(\text{proj}(\rho')) + t \overline{\text{wt}}(\text{last}(\pi')) + \overline{\text{wt}}(\Delta \text{proj}(\delta'))$$

By the first item and by definition of $\text{proj}(\rho)$, we conclude that $\text{wt}_{\Sigma}(\rho) = \text{wt}_{\Sigma}(\text{proj}(\rho))$.

3. We reason by induction on the length of ρ . If $\rho = (\ell_i, \nu)$, the property is trivial. Now, we suppose that $\rho' = \rho \xrightarrow{t, \delta'} (\pi', \nu')$. We have $\text{proj}(\rho') = \text{proj}(\rho) \xrightarrow{t, \delta} (\text{last}(\pi'), \nu')$ with $\delta = \Delta \text{proj}(\delta')$. Since ρ is a prefix of $\rho' \in \text{FPlays}_{\mathcal{U}}^*$, ρ belongs to $\text{FPlays}_{\mathcal{U}}^*$ too and does not end in L_t . Thus, letting $\text{last}(\rho) = (\pi, \nu)$, by induction hypothesis, $\text{proj}(\rho)$ follows π . Moreover, we have $\text{NEXT}(\pi, \delta) = (\pi', \delta')$. By definition of NEXT , the value of π' must be obtained from π on lines 4, or 8, and thus $\pi' = \pi \cdot \delta$. In particular, we can deduce that $\text{proj}(\rho')$ follows π . ◀

Then, for all plays $\rho \in \text{FPlays}_{\mathcal{U}}^*$ (for plays not starting in ℓ_i or plays ending in the target, the decision of ρ is irrelevant) such that $\text{last}(\rho) = (\pi, \nu)$ and $\pi \in L'_{\text{Max}}$, we define

$$\tau_{\mathcal{U}}(\rho) = (t, \delta') \quad \text{if } \tau_{\overline{\mathcal{G}}}(\text{proj}(\rho)) = (t, \delta) \text{ and } \text{NEXT}(\pi, \delta) = (\pi', \delta') \quad (2)$$

This is a valid decision for Max . First, by Lemma 27-1, we have $\text{last}(\text{proj}(\rho)) = (\text{last}(\pi), \nu)$. Moreover, delays chosen in $\tau_{\overline{\mathcal{G}}}$ and $\tau_{\mathcal{U}}$ are the same, and the guards of δ and δ' are identical. Thus, whether or not the location π is urgent (i.e. $\text{last}(\pi)$ is urgent), the decision (t, δ') gives rise to an edge in $\llbracket \mathcal{U} \rrbracket$.

Since the definition of $\tau_{\mathcal{U}}$ relies on the projection, it is of no surprise that:

► **Lemma 28.** Let $\rho \in \text{FPlays}_{\mathcal{U}}^*$ be a play conforming to $\tau_{\mathcal{U}}$. Then $\text{proj}(\rho)$ is conforming to $\tau_{\overline{\mathcal{G}}}$.

Proof. We reason by induction on the length of ρ . If $\rho = (\ell_i, \nu)$, then $\text{proj}(\rho) = (\ell_i, \nu)$ and the property is trivial. Otherwise, let $\rho = \rho^1 \xrightarrow{t, \delta'} (\pi, \nu)$. Then, $\text{proj}(\rho) = \text{proj}(\rho^1) \xrightarrow{t, \delta} (\text{last}(\pi), \nu)$ where $\delta = \Delta \text{proj}(\delta')$. By induction hypothesis, $\text{proj}(\rho^1)$ is conforming to $\tau_{\overline{\mathcal{G}}}$. Let $\text{last}(\text{proj}(\rho^1)) = (\ell^1, \nu^1)$. If $\ell^1 \in L_{\text{Min}}$, we directly conclude that $\text{proj}(\rho)$ is conforming to $\tau_{\overline{\mathcal{G}}}$ too. Otherwise, and since ρ is conforming to $\tau_{\mathcal{U}}$ and the last location of ρ^1 also belongs to Max (by Lemma 27-1), we have $\tau_{\mathcal{U}}(\rho^1) = (t, \delta')$. In particular, by definition of $\tau_{\mathcal{U}}$ (see (2)), $\tau_{\overline{\mathcal{G}}}(\text{proj}(\rho^1)) = (t, \Delta \text{proj}(\delta')) = (t, \delta)$. Thus $\rho_{\overline{\mathcal{G}}}$ is conforming to $\tau_{\overline{\mathcal{G}}}$. ◀

Now, we prove Proposition 26. To do so, we show that for all plays $\rho_{\mathcal{U}}$ from (ℓ, ν) conforming to $\tau_{\mathcal{U}}$, there exists a play $\rho_{\overline{\mathcal{G}}}$ from (ℓ, ν) conforming to $\tau_{\overline{\mathcal{G}}}$ such that $P(\rho_{\overline{\mathcal{G}}}) \leq P(\rho_{\mathcal{U}})$. We cannot directly use the projection operator, since some plays $\rho_{\mathcal{U}}$ may end up in $\mathbf{t}_{<0}$ or $\mathbf{t}_{\geq 0}$. We treat the ones ending in $\mathbf{t}_{\geq 0}$ by making use of the final weight function we have chosen for $\mathbf{t}_{\geq 0}$ (bigger than any acyclic play of $\overline{\mathcal{G}}$). We show that there cannot be such plays $\rho_{\mathcal{U}}$ ending in $\mathbf{t}_{<0}$, since they would contradict Lemma 16-2.

Proof of Proposition 26. Let $\rho_{\mathcal{U}}$ be a play conforming to $\tau_{\mathcal{U}}$. If $\rho_{\mathcal{U}}$ does not reach a target location of \mathcal{U} or reaches target $\mathbf{t}_{+\infty}$, then $P(\rho_{\mathcal{U}}) = +\infty$, and for all plays $\rho_{\overline{\mathcal{G}}}$ conforming to $\tau_{\overline{\mathcal{G}}}$, we have $P(\rho_{\overline{\mathcal{G}}}) \leq +\infty = P(\rho_{\mathcal{U}})$. Now, suppose that $\rho_{\mathcal{U}}$ reaches a target location different from $\mathbf{t}_{+\infty}$.

- If the target location reached by $\rho_{\mathcal{U}}$ is not in $\{\mathbf{t}_{\geq 0}, \mathbf{t}_{<0}\}$, then $\rho_{\mathcal{U}} \in \text{FPlays}_{\mathcal{U}}^*$ and we can thus let $\rho_{\overline{\mathcal{G}}} = \text{proj}(\rho_{\mathcal{U}})$. Lemma 28 ensures that $\rho_{\overline{\mathcal{G}}}$ is conforming to $\tau_{\overline{\mathcal{G}}}$. Moreover, Lemma 27-1 ensures that if $\text{last}(\rho_{\mathcal{U}}) = (\pi, \nu)$ then $\text{last}(\rho_{\overline{\mathcal{G}}}) = (\text{last}(\pi), \nu)$ so that $\text{wt}'_t(\pi, \nu) = \overline{\text{wt}}'_t(\text{last}(\pi), \nu)$. Since proj preserves the weight (see Lemma 27-2), we obtain $P(\rho_{\overline{\mathcal{G}}}) = P(\rho_{\mathcal{U}})$.
- If the target location reached by $\rho_{\mathcal{U}}$ is $\mathbf{t}_{\geq 0}$, then we decompose $\rho_{\mathcal{U}}$ as $\rho_{\mathcal{U}} = \rho_{\mathcal{U}}^1 \xrightarrow{t, \delta'} (\mathbf{t}_{\geq 0}, \nu)$ with $\rho_{\mathcal{U}}^1 \in \text{FPlays}_{\mathcal{U}}^*$. Let $\rho_{\overline{\mathcal{G}}} = \rho_{\overline{\mathcal{G}}}^1 \rho_{\overline{\mathcal{G}}}^2$ be a play such that $\rho_{\overline{\mathcal{G}}}^1 = \text{proj}(\rho_{\mathcal{U}}^1) \xrightarrow{t, \delta} (\ell, \nu)$ with $\delta = \Delta \text{proj}(\delta')$, and $\rho_{\overline{\mathcal{G}}}^2$ be the play from (ℓ, ν) conforming to $\tau_{\overline{\mathcal{G}}}$ and an attractor of Min to L_t . We note that $\rho_{\overline{\mathcal{G}}}^2$ exists since the value in \mathcal{G} is supposed finite, thus Min can always guarantee to reach the target, moreover in at most $|L|$ steps (since regions are already encoded in this game). Letting $(\pi', \nu') = \text{last}(\rho_{\mathcal{U}}^1)$, Lemma 27-1 ensures that $(\text{last}(\pi'), \nu') = \text{last}(\text{proj}(\rho_{\mathcal{U}}^1))$. If $\pi' \in L'_{\text{Max}}$, since $\tau_{\mathcal{U}}(\rho_{\mathcal{U}}^1) = (t, \delta')$ and $\text{NEXT}(\pi', \delta) = (\mathbf{t}_{<0}, \delta')$, by construction of $\tau_{\mathcal{U}}$, this implies that $\tau_{\overline{\mathcal{G}}}(\text{proj}(\rho_{\mathcal{U}}^1)) = (t, \delta)$. The last move of $\rho_{\mathcal{U}}^1$ is thus conforming to $\tau_{\overline{\mathcal{G}}}$. By Lemma 28 and the choice of $\rho_{\overline{\mathcal{G}}}^2$, $\rho_{\overline{\mathcal{G}}}$ is thus entirely conforming to $\tau_{\overline{\mathcal{G}}}$. Moreover, $\overline{\text{wt}}(\text{last}(\pi')) = \text{wt}'(\pi')$ by definition of the unfolding. Thus, also using Lemma 27-2, we obtain

$$\begin{aligned} \text{wt}_{\Sigma}(\rho_{\overline{\mathcal{G}}}^1) &= \text{wt}_{\Sigma}(\text{proj}(\rho_{\mathcal{U}}^1)) + t \overline{\text{wt}}(\text{last}(\pi')) + \overline{\text{wt}}(\delta) \\ &= \text{wt}_{\Sigma}(\rho_{\mathcal{U}}^1) + t \text{wt}'(\pi') + \text{wt}'(\delta') \\ &= \text{wt}_{\Sigma}(\rho_{\mathcal{U}}) \end{aligned}$$

Moreover, as $\rho_{\overline{\mathcal{G}}}^1$ is conforming to an attractor, its length is bounded by $|L|$. Each of its edges has a weight bounded in absolute values by $W_{\text{tr}} + M W_{\text{loc}}$. By adding its final weight, we obtain

$$P(\rho_{\overline{\mathcal{G}}}^2) \leq |L|(W_{\text{tr}} + M W_{\text{loc}}) + W_{\text{fin}}$$

To conclude, we remark that $\rho_{\mathcal{U}}$ reaches $\mathbf{t}_{\geq 0}$, and its weight is thus

$$P(\rho_{\mathcal{U}}) = \text{wt}_{\Sigma}(\rho_{\mathcal{U}}) + |L|(W_{\text{tr}} + M W_{\text{loc}}) + W_{\text{fin}}$$

Therefore $P(\rho_{\overline{\mathcal{G}}}) = \text{wt}_{\Sigma}(\rho_{\overline{\mathcal{G}}}^1) + P(\rho_{\overline{\mathcal{G}}}^2) \leq P(\rho_{\mathcal{U}})$.

- If the target location reached by $\rho_{\mathcal{U}}$ is $\mathbf{t}_{<0}$, as before we decompose $\rho_{\mathcal{U}}$ as $\rho_{\mathcal{U}} = \rho_{\mathcal{U}}^1 \xrightarrow{t, \delta'} (\mathbf{t}_{\geq 0}, \nu)$ with $\rho_{\mathcal{U}}^1 \in \text{FPlays}_{\mathcal{U}}^*$. Let $\rho_{\overline{\mathcal{G}}}^1 = \text{proj}(\rho_{\mathcal{U}}^1) \xrightarrow{t, \delta} (\ell, \nu)$ with $\delta = \Delta \text{proj}(\delta')$. As in the previous case, $\rho_{\overline{\mathcal{G}}}^1$ is conforming to $\tau_{\overline{\mathcal{G}}}$. By definition of \mathcal{U} , letting π the last location of $\rho_{\mathcal{U}}^1$ (not in L'_t), we have $\text{NEXT}(\pi, \delta) = (\mathbf{t}_{<0}, \delta')$ with $|\pi|_{\delta} > 0$: by letting $\pi = \pi_1 \delta \pi_2$ with $|\pi_2|_{\delta} = 0$, we have $\text{Val}_{\overline{\mathcal{G}}}(\pi_2 \delta) < 0$. By Lemma 27-3, we know that $\text{proj}(\rho_{\mathcal{U}}^1)$ follows π . Thus, $\rho_{\overline{\mathcal{G}}}^1$ follows $\pi \delta$, and as a consequence, finishes by a play that follows the cyclic path $\pi_2 \delta$ of negative value. Since it is conforming to $\tau_{\overline{\mathcal{G}}}$, it contradicts Lemma 16-2.

15:20 Decidability of One-Clock Weighted Timed Games with Arbitrary Weights

To conclude, we have shown that for all plays $\rho_{\mathcal{U}}$ from (ℓ_i, ν) conforming to $\tau_{\mathcal{U}}$, we can build a play $\rho_{\overline{\mathcal{G}}}$ from (ℓ_i, ν) conforming to $\tau_{\overline{\mathcal{G}}}$ such that $P(\rho_{\overline{\mathcal{G}}}) \leq P(\rho_{\mathcal{U}})$. In particular,

$$\text{Val}_{\overline{\mathcal{G}}}^{\tau_{\overline{\mathcal{G}}}}(\ell_i, \nu) = \inf_{\tau_{\overline{\mathcal{G}}} \in \text{Strat}_{\text{Min}, \overline{\mathcal{G}}}} P(\text{Play}((\ell_i, \nu), \sigma_{\overline{\mathcal{G}}}, \tau_{\overline{\mathcal{G}}})) \leq \inf_{\tau_{\mathcal{U}} \in \text{Strat}_{\text{Min}, \mathcal{U}}} P(\text{Play}((\ell_i, \nu), \sigma_{\mathcal{U}}, \tau_{\mathcal{U}})) = \text{Val}_{\mathcal{U}}^{\tau_{\mathcal{U}}}(\ell_i, \nu)$$

◀

B Proof of $\text{Val}_{\overline{\mathcal{G}}}(\ell_i, \nu) \geq \text{Val}_{\mathcal{U}}(\ell_i, \nu)$

We show this second inequality slightly differently. First we rewrite it: $\text{Val}_{\overline{\mathcal{G}}}(\ell_i, \nu) \geq \sup_{\tau_{\mathcal{U}}} \text{Val}_{\mathcal{U}}^{\tau_{\mathcal{U}}}(\ell_i, \nu)$. Considering for $\tau_{\mathcal{U}}$ the memoryless optimal strategy of Max in \mathcal{U} satisfying the conditions of Lemma 23, we therefore show that

► **Proposition 29.** $\text{Val}_{\overline{\mathcal{G}}}(\ell_i, \nu) \geq \text{Val}_{\mathcal{U}}^{\tau_{\mathcal{U}}}(\ell_i, \nu)$

To do so, following Figure 6, we first define the function Φ , mapping plays of $\overline{\mathcal{G}}$ in plays of \mathcal{U} . It needs to take care of the appearance of more than one occurrence of a transition with a reset in plays of $\overline{\mathcal{G}}$. Formally, it is defined by induction on the length of the plays by letting $\Phi(\ell_i, \nu) = (\ell_i, \nu)$, and for all plays $\rho \in \text{FPlays}_{\overline{\mathcal{G}}}$, letting $\rho' = \rho \xrightarrow{t, \delta} (\ell, \nu)$,

1. if $\Phi(\rho)$ ends in $t_{+\infty}$, we let $\Phi(\rho') = \Phi(\rho)$;
2. else if δ contains a reset and $\Phi(\rho) = \rho_1 \xrightarrow{t', \delta'} \rho_2$ with $\Delta \text{proj}(\delta') = \delta$, letting π the first location of ρ_2 , we let $\Phi(\rho') = \rho_1 \xrightarrow{t', \delta'} (\pi, 0)$;
3. otherwise, $\Phi(\rho') = \Phi(\rho) \xrightarrow{t, \delta} (\pi', \nu)$ if $\text{NEXT}(\pi, \delta) = (\pi', \delta')$ with π the last location of $\Phi(\rho)$.

This function satisfies the following properties:

► **Lemma 30.** For all plays $\rho \in \text{FPlays}_{\overline{\mathcal{G}}}$, if $\text{last}(\Phi(\rho)) = (\pi, \nu)$ with $\pi \notin t_{+\infty}$, then we have $\pi \notin \{t_{<0}, t_{\geq 0}\}$ and

$$\text{last}(\rho) = \begin{cases} (\text{last}(\pi), \nu) & \text{if } \pi \notin L_t \\ (\pi, \nu) & \text{otherwise} \end{cases}$$

Proof. We show the property by induction on the length of ρ . If $\rho = (\ell_i, \nu)$, then $\Phi(\rho) = \rho$ and the property holds. Otherwise, we let $\rho' = \rho \xrightarrow{t, \delta} (\ell, \nu)$, suppose that the property holds for ρ (that does not end in L'_t) and follow the definition of Φ .

1. If $\Phi(\rho)$ ends in $t_{+\infty}$, we have $\Phi(\rho') = \Phi(\rho)$ and this case is thus not possible (since $\Phi(\rho')$ is supposed to not end in $t_{+\infty}$).
2. Else if δ contains a reset and $\Phi(\rho) = \rho_1 \xrightarrow{t', \delta'} \rho_2$ with $\Delta \text{proj}(\delta') = \delta$, letting π' the first location of ρ_2 , we have $\Phi(\rho') = \rho_1 \xrightarrow{t', \delta'} (\pi', 0)$. Letting π_1 the last location of ρ_1 , we have $\text{NEXT}(\pi_1, \delta) = (\pi', \delta')$. If δ goes to location $\ell \in L_t$, then $\pi' = \ell \in L_t$, so that $\text{last}(\rho') = (\ell, 0) = (\text{last}(\Phi(\rho')), 0)$ as expected. Since ρ_1 does not contain a transition δ'_1 such that $\Delta \text{proj}(\delta'_1) = \delta$ (otherwise, in $\Phi(\rho)$, we would have already fired twice the transition δ with a reset, before trying to fire it a third time), we have $\text{last}(\Phi(\rho)) = (\pi, 0)$ with $\pi \notin \{t_{<0}, t_{\geq 0}\}$. Thus $\pi = \pi' \cdot \delta$ (and thus $\pi \notin \{t_{<0}, t_{\geq 0}\}$) so that $\text{last}(\pi) = \ell$, and we conclude.
3. Otherwise $\Phi(\rho') = \Phi(\rho) \xrightarrow{t, \delta} (\pi', \nu)$ if $\text{NEXT}(\pi, \delta) = (\pi', \delta')$ with π the last location of $\Phi(\rho)$. Once again, we are in a case where $\pi' = \pi \cdot \delta$ which allows us to conclude as before. ◀

Then, we define $\tau_{\bar{\mathcal{G}}}$ such that its behaviour is the same as the one given by $\tau_{\mathcal{U}}$ after the application of Φ on the finite play, i.e. after the removal of all cyclic paths ending by a transition with a reset. Formally, for all plays $\rho \in \text{FPlays}_{\bar{\mathcal{G}}}$, we let $\tau_{\bar{\mathcal{G}}}(\rho)$ be defined as any valid move (t, δ) if $\Phi(\rho)$ ends in $\mathfrak{t}_{+\infty}$, and otherwise

$$\tau_{\bar{\mathcal{G}}}(\rho) = (t, \Delta\text{proj}(\delta')) \quad \text{if } \tau_{\mathcal{U}}(\Phi(\rho)) = (t, \delta') \quad (3)$$

This is a valid decision for **Max**. First, by Lemma 30, $\text{last}(\rho) = (\text{last}(\pi), \nu)$ when $\text{last}(\Phi(\rho)) = (\pi, \nu)$. Moreover, delays chosen in $\tau_{\bar{\mathcal{G}}}$ and $\tau_{\mathcal{U}}$ are the same, and the guards of δ' and $\Delta\text{proj}(\delta')$ are identical. Thus, whether or not the location π is urgent, the decision $(t, \Delta\text{proj}(\delta'))$ gives rise to an edge in $\llbracket \bar{\mathcal{G}} \rrbracket$.

Since the definition of $\tau_{\bar{\mathcal{G}}}$ relies on the operation Φ , it is again not surprising that:

► **Lemma 31.** *Let $\rho \in \text{FPlays}_{\bar{\mathcal{G}}}$ be a play conforming to $\tau_{\bar{\mathcal{G}}}$. Then $\Phi(\rho)$ is conforming to $\tau_{\mathcal{U}}$.*

Proof. We reason by induction on the length of ρ . If $\rho = (\ell_i, \nu)$, then $\Phi(\rho) = (\ell_i, \nu)$ and the property is trivial. Otherwise, we suppose that $\rho' = \rho \xrightarrow{t, \delta} (\ell, \nu)$. By induction hypothesis, $\Phi(\rho)$ conforms to $\tau_{\mathcal{U}}$.

1. If $\Phi(\rho)$ ends in $\mathfrak{t}_{+\infty}$, we have $\Phi(\rho') = \Phi(\rho)$ that conforms to $\tau_{\mathcal{U}}$.
2. If δ contains a reset and $\Phi(\rho) = \rho_1 \xrightarrow{t', \delta'} \rho_2$ with $\Delta\text{proj}(\delta') = \delta$, letting π the first location of ρ_2 , we have $\Phi(\rho') = \rho_1 \xrightarrow{t', \delta'} (\pi, 0)$. This is a prefix of $\Phi(\rho)$ that conforms to $\tau_{\mathcal{U}}$, so $\Phi(\rho')$ conforms to $\tau_{\mathcal{U}}$ too.
3. Otherwise, $\Phi(\rho') = \Phi(\rho) \xrightarrow{t, \delta'} (\pi', \nu)$ if $\text{NEXT}(\pi, \delta) = (\pi', \delta')$ with π the last location of $\Phi(\rho)$. If $\Phi(\rho)$ ends in a location of **Min**, since it is conforming to $\tau_{\mathcal{U}}$, so does $\Phi(\rho')$. Otherwise, $\tau_{\bar{\mathcal{G}}}(\rho) = (t, \delta)$ which implies that $\tau_{\mathcal{U}}(\Phi(\rho)) = (t, \delta')$ with $\Delta\text{proj}(\delta'') = \delta$, meaning that $\text{NEXT}(\pi, \delta) = (\pi', \delta'')$, i.e. $\delta'' = \delta'$: in this case too, $\Phi(\rho')$ is conforming to $\tau_{\mathcal{U}}$. ◀

Now, we prove Proposition 29. Notice that contrary to Proposition 26, we do not aim at comparing $\text{Val}_{\mathcal{U}}^{\tau_{\mathcal{U}}}(\ell_i, \nu)$ with $\text{Val}_{\bar{\mathcal{G}}}^{\tau_{\bar{\mathcal{G}}}}(\ell_i, \nu)$ but instead directly with $\text{Val}_{\bar{\mathcal{G}}}(\ell_i, \nu)$. This is helpful here, since we do not need to start with any play ρ conforming to $\tau_{\bar{\mathcal{G}}}$. Instead, we pick a special play, choosing well the strategy followed by **Min**. Indeed, let **Min** follow an ε -optimal (switching) strategy σ in $\bar{\mathcal{G}}$, as given in [12, 13]. As we explained before Definition 21, in WTG without resets, this ensures that in all plays $\rho_{\bar{\mathcal{G}}}$ conforming to σ , the target is reached fast enough (with a number of transitions bounded by κ). We can easily enrich the result of [12, 13] to take into account resets. Indeed, as performed in [12, Theorem 10], to show that all one-clock WTG have a (a priori non computable) value function that is piecewise affine with a finite number of cutpoints, we can replace each transition with a reset with a new transition jumping in a fresh target location of value given by the value function we aim at computing. From a strategy perspective, this means that in each component of our unfolding (in-between two transitions with a reset), **Min** follows a switching strategy. Notice that such strategies are a priori not knowing to be computable (since we cannot perform the transformation described above, using the value function), but we use only its existence in this proof.

We finally obtain an ε -optimal strategy σ for **Min** in $\bar{\mathcal{G}}$ such that that in all plays $\rho_{\bar{\mathcal{G}}}$ conforming to σ , in-between two transitions with a reset and after the last such transition, the number of transitions is bounded by κ .

Proof of Proposition 29. We now consider the special play ρ from (ℓ_i, ν) conforming to σ and $\tau_{\bar{\mathcal{G}}}$. It reaches a target, since σ is ε -optimal and $\text{Val}_{\bar{\mathcal{G}}}(\ell_i, \nu) \neq +\infty$. We show that

$$\exists \rho_{\mathcal{U}} \in \text{FPlays}_{\mathcal{U}} \text{ conforming to } \tau_{\mathcal{U}} \quad \text{P}(\rho_{\mathcal{U}}) \leq \text{P}(\rho) \quad (\star)$$

15:22 Decidability of One-Clock Weighted Timed Games with Arbitrary Weights

As a consequence, we obtain

$$\text{Val}_{\mathcal{U}}^{\tau_{\mathcal{U}}}(\ell_i, \nu) = \inf_{\sigma_{\mathcal{U}} \in \text{Strat}_{\text{Min}, \mathcal{U}}} \text{P}(\text{Play}((\ell_i, \nu), \sigma_{\mathcal{U}}, \tau_{\mathcal{U}})) \leq \text{P}(\rho_{\mathcal{U}}) \leq \text{P}(\rho) \leq \text{Val}_{\overline{\mathcal{G}}}(\ell_i, \nu) + \varepsilon$$

Since this holds for all $\varepsilon > 0$, we have $\text{Val}_{\mathcal{U}}^{\tau_{\mathcal{U}}}(\ell_i, \nu) \leq \text{Val}_{\overline{\mathcal{G}}}(\ell_i, \nu)$ as expected.

To show (\star) , we proceed by induction on the prefixes ρ' of ρ , proving that (\star) holds, or that $\Phi(\rho')$ does not end in $\mathbf{t}_{+\infty}$ and $\text{wt}_{\Sigma}(\Phi(\rho')) \leq \text{wt}_{\Sigma}(\rho')$. At the end of the induction, we therefore obtain (\star) or that $\Phi(\rho)$ does not end in $\mathbf{t}_{+\infty}$ and $\text{wt}_{\Sigma}(\Phi(\rho)) \leq \text{wt}_{\Sigma}(\rho)$. We let $\text{last}(\Phi(\rho)) = (\pi, \nu)$. By Lemma 30, if $\pi \notin L_t$, then $\text{last}(\rho) = (\text{last}(\pi), \nu)$, with $\text{last}(\pi) \notin L_t$: this contradicts the fact that ρ reaches the target. Thus, $\pi \in L_t$, and $\text{last}(\rho) = (\pi, \nu)$. Therefore, $\text{P}(\Phi(\rho)) = \text{wt}_{\Sigma}(\Phi(\rho)) + \text{wt}'_t(\pi, \nu) \leq \text{wt}_{\Sigma}(\rho) + \text{wt}'_t(\pi, \nu) = \text{P}(\rho)$. Since $\Phi(\rho)$ conforms to $\tau_{\mathcal{U}}$, we obtain (\star) here too.

For $\rho' = (\ell_i, \nu)$, $\text{wt}_{\Sigma}(\Phi(\rho')) = 0 = \text{wt}_{\Sigma}(\rho')$. Suppose then that $\rho' = \rho'' \xrightarrow{t, \delta} (\ell, \nu)$. By induction on ρ'' , if (\star) does not (already) hold, we know that $\Phi(\rho'')$ does not end in $\mathbf{t}_{+\infty}$ and $\text{wt}_{\Sigma}(\Phi(\rho'')) \leq \text{wt}_{\Sigma}(\rho'')$. We follow the three cases of the definition of $\Phi(\rho')$.

1. We cannot have $\Phi(\rho'')$ ending in $\mathbf{t}_{+\infty}$ by hypothesis.
2. Suppose now that δ contains a reset and $\Phi(\rho'') = \rho_1 \xrightarrow{t', \delta'} \rho_2$ with $\Delta \text{proj}(\delta') = \delta$. Letting π the first location of ρ_2 , we have $\Phi(\rho') = \rho_1 \xrightarrow{t', \delta'} (\pi, 0)$. Thus

$$\text{wt}_{\Sigma}(\Phi(\rho')) = \text{wt}_{\Sigma}(\Phi(\rho'')) - \text{wt}_{\Sigma}(\rho_2) \leq \text{wt}_{\Sigma}(\rho'') - \text{wt}_{\Sigma}(\rho_2) \quad (4)$$

Let $(\pi', \nu') = \text{last}(\rho_2)$, and $\rho_{\mathcal{U}} = \Phi(\rho'') \xrightarrow{t, \delta''} (\pi'', 0)$, with $\text{NEXT}(\pi', \delta) = (\pi'', \delta'')$. It contains twice a transition with a reset coming from the same transition δ of $\overline{\mathcal{G}}$, therefore $\pi'' \in \{\mathbf{t}_{<0}, \mathbf{t}_{\geq 0}\}$. Notice that $\rho_{\mathcal{U}}$ is conforming to $\tau_{\mathcal{U}}$, since $\Phi(\rho'')$ does and if π' belongs to Max , this follows directly from the definition of $\tau_{\overline{\mathcal{G}}}$ from $\tau_{\mathcal{U}}$ (since $\tau_{\overline{\mathcal{G}}}(\rho''_{\overline{\mathcal{G}}}) = (t, \delta)$ and $\Phi(\rho'') \notin \mathbf{t}_{+\infty}$). Therefore, if $\pi'' = \mathbf{t}_{<0}$, $\text{P}(\rho_{\mathcal{U}}) = -\infty$ and (\star) holds. If $\pi'' = \mathbf{t}_{\geq 0}$, by Lemma 23 applied on $\rho_{\mathcal{U}}$, $\text{wt}_{\Sigma}(\rho_2 \xrightarrow{t, \delta''} (\mathbf{t}_{\geq 0}, 0)) \geq 0$. Combined with (4), we obtain that

$$\begin{aligned} \text{wt}_{\Sigma}(\Phi(\rho')) &\leq \text{wt}_{\Sigma}(\rho'') + \text{wt}_{\Sigma}((\pi', \nu') \xrightarrow{t, \delta''} (\mathbf{t}_{\geq 0}, 0)) \\ &= \text{wt}_{\Sigma}(\rho'') + t \text{wt}'(\pi') + \text{wt}'(\delta'') \\ &= \text{wt}_{\Sigma}(\rho'') + t \overline{\text{wt}}(\ell') + \overline{\text{wt}}(\delta) = \text{wt}_{\Sigma}(\rho') \end{aligned}$$

where we have set ℓ' the last location of ρ'' , that is also the last location of π' .

3. Otherwise, $\Phi(\rho') = \Phi(\rho'') \xrightarrow{t, \delta'} (\pi', \nu)$ if $\text{NEXT}(\pi, \delta) = (\pi', \delta')$ with π the last location of $\Phi(\rho'')$. In this case,

$$\text{wt}_{\Sigma}(\Phi(\rho')) = \text{wt}_{\Sigma}(\Phi(\rho'')) + t \text{wt}'(\pi) + \text{wt}'(\delta') \leq \text{wt}_{\Sigma}(\rho'') + t \overline{\text{wt}}(\ell') + \overline{\text{wt}}(\delta) = \text{wt}_{\Sigma}(\rho')$$

where we have written ℓ' the last location of ρ'' .

This ends the proof by induction. ◀

Language Inclusion for Boundedly-Ambiguous Vector Addition Systems Is Decidable

Wojciech Czerwiński ✉ 

University of Warsaw, Poland

Piotr Hofman ✉ 

University of Warsaw, Poland

Abstract

We consider the problems of language inclusion and language equivalence for Vector Addition Systems with States (VASSes) with the acceptance condition defined by the set of accepting states (and more generally by some upward-closed conditions). In general the problem of language equivalence is undecidable even for one-dimensional VASSes, thus to get decidability we investigate restricted subclasses. On one hand we show that the problem of language inclusion of a VASS in k -ambiguous VASS (for any natural k) is decidable and even in Ackermann. On the other hand we prove that the language equivalence problem is Ackermann-hard already for deterministic VASSes. These two results imply Ackermann-completeness for language inclusion and equivalence in several possible restrictions. Some of our techniques can be also applied in much broader generality in infinite-state systems, namely for some subclass of well-structured transition systems.

2012 ACM Subject Classification Theory of computation → Parallel computing models

Keywords and phrases vector addition systems, language inclusion, language equivalence, determinism, unambiguity, bounded ambiguity, Petri nets, well-structured transition systems

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.16

Related Version *Full Version:* <https://arxiv.org/abs/2202.08033> [8]

Funding *Wojciech Czerwiński:* Supported by the ERC grant INFSYS, agreement no. 950398.

Piotr Hofman: Supported by the ERC grant INFSYS, agreement no. 950398.

Acknowledgements We thank Filip Mazowiecki for asking the question for boundedly-ambiguous VASSes and formulating the conjecture that control automata of boundedly-ambiguous VASSes can be made boundedly-ambiguous. We also thank him and David Purser for inspiring discussions on the problem. We thank Thomas Colcombet for suggesting the way of proving Theorem 25, Mahsa Shirmohammadi for pointing us to the undecidability result [20] and Lorenzo Clemente for inspiring discussions on weighted models.

1 Introduction

Vector Addition Systems (VASes) together with almost equivalent Petri Nets and Vector Addition Systems with States (VASSes) are one of the most fundamental computational models with a lot of applications in practice for modelling concurrent behaviour. There is also an active field of theoretical research on VASes, with a prominent example being the reachability problem whose complexity was established recently to be Ackermann-complete [23, 11] and [24]. An important type of questions that can be asked for any pair of systems is whether they are equivalent in a certain sense. The problem of language equivalence (acceptance by configuration) was already proven to be undecidable in 1975 by Araki and Kasami [1] (Theorem 3). They also have shown that the language equivalence (acceptance by configuration) for deterministic VASes is reducible to the reachability problem, thus decidable, as the reachability problem was shown to be decidable by Mayr a few years later in 1981 [25]. The equality of the reachability sets of two given VASes was also shown



© Wojciech Czerwiński and Piotr Hofman;

licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Sławomir Lasota, and Anca Muscholl; Article No. 16; pp. 16:1–16:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

undecidable in the 70-ties by Hack [16]. Jančar has proven in 1995 that the most natural behavioural equivalence, namely the bisimilarity equivalence is undecidable for VASSes [19]. His proof works for only two dimensions (improving the previous results [1]) and is applicable also to language equivalence (this time as well for acceptance by states). A few years later in 2001 Jančar has shown in [20] that any reasonable equivalence in-between language equivalence (with acceptance by states) and bisimilarity is undecidable (Theorem 3) and Ackermann-hard even for systems with finite reachability set (Theorem 4). For the language equivalence problem the state-of-the-art was improved a few years ago. In [17] (Theorem 20) it was shown that already for one-dimensional VASSes the language equivalence (and even the trace equivalence, namely language equivalence with all the states accepting) is undecidable.

As the problem of language equivalence (and similar ones) is undecidable for general VASSes (even in very small dimensions) it is natural to search for subclasses in which the problem is decidable. Decidability of the problem for deterministic VASSes [1, 25] suggests that restricting nondeterminism might be a good idea. Recently a lot of attention was drawn to unambiguous systems [6], namely systems in which each word is accepted by at most one accepting run, but can potentially have many non-accepting runs. Such systems are often more expressive than the deterministic ones however they share some of their good properties, for example [5]. In particular many problems are more tractable in the unambiguous case than in the general nondeterministic case. This difference is already visible for finite automata. The language universality and the language equivalence problems for unambiguous finite automata are in NC^2 [32] (so also in PTime) while they are in general PSpace-complete for nondeterministic finite automata. Recently it was shown that for some infinite-state systems the language universality, equivalence and inclusion problems are much more tractable in the unambiguous case than in the general one. There was a line of research investigating the problem for register automata [26, 2, 10] culminating in the work of Bojańczyk, Klin and Moerman [3]. They have shown that for unambiguous register automata with guessing the language equivalence problem is in ExpTime (and in PTime for a fixed number of registers). This result is in a sheer contrast with the undecidability of the problem in the general case even for two register automata without guessing [27] or one register automata with guessing (the proof can be obtained following the lines of [12] as explained in [10]). Recently it was also shown in [7] that the language universality problem for VASSes accepting with states is ExpSpace-complete in the unambiguous case in contrast to Ackermann-hardness in the nondeterministic case (even for one-dimensional VASSes) [18].

Our contribution. In this article we follow the line of [7] and consider problems of language equivalence and inclusion for unambiguous VASSes and also for their generalisations k -ambiguous VASSes (for $k \in \mathbb{N}$) in which each word can have at most k accepting runs. The acceptance condition is defined by some upward-closed set of configurations which generalises a bit the acceptance by states considered in [7]. Notice that the equivalence problem can be easily reduced to the inclusion problem, so we prove lower complexity bounds for the equivalence problem and upper complexity bounds for the inclusion problem.

Our main lower bound result is the following one.

► **Theorem 1.** *The language equivalence problem for deterministic VASSes is Ackermann-hard.*

Our first important upper bound result is the following one.

► **Theorem 2.** *The inclusion problem of a nondeterministic VASS language in an unambiguous VASS language is in Ackermann.*

The proof of Theorem 2 is quite simple, but it uses a novel technique. We add a regular lookahead to a VASS and use results about regular-separability of VASSes from [9] to reduce the problem, roughly speaking, to the deterministic case. This technique can be applied to more general systems namely well-structures transition-systems [14]. We believe that it might be interesting on its own and reveal some connection between separability problems and the notion of unambiguity.

Our main technical result concerns VASSes with bounded ambiguity.

► **Theorem 3.** *For each $k \in \mathbb{N}$ the language inclusion problem of a VASS in a k -ambiguous VASS is in Ackermann.*

Notice that Theorem 3 generalises Theorem 2. We however decided to present separately the proof of Theorem 2 because it presents a different technique of independent interest, which can be applied more generally. Additionally it is a good introduction to a more technically challenging proof of Theorem 3. The proof of Theorem 3 proceeds in three steps. First we show that the problem for k -ambiguous VASS can be reduced to the case when the control automaton of the VASS is k -ambiguous. Next, we show that the control automaton can be even made k -deterministic (roughly speaking for each word there are at most k runs). Finally we show that the problem of inclusion of a VASS language in a k -deterministic VASS can be reduced to the reachability problem for VASSes which is in Ackermann [24].

On a way to show Theorem 3 we also prove several other lemmas and theorems, which we believe may be interesting on their own. Theorems 1 and 3 together easily imply the following corollary.

► **Corollary 4.** *The language equivalence problem is Ackermann-complete for:*

- deterministic VASSes
- unambiguous VASSes
- k -ambiguous VASSes for any $k \in \mathbb{N}$

Organisation of the paper. In Section 2 we introduce the needed notions. Then in Section 3 we present results concerning deterministic VASSes. First we show Theorem 1. Next, we prove that the inclusion problem of a VASS language in a language of a deterministic VASS, a k -deterministic VASS or a VASS with holes (to be defined) is in Ackermann. This is achieved by a reduction to the VASS reachability problem. In Section 4 we define adding a regular lookahead to VASSes. Then we show that with a carefully chosen lookahead we can reduce the inclusion problem of a VASS language in an unambiguous VASS language into the inclusion problem of a VASS language in language of deterministic VASS with holes. This latter one is in Ackermann due to Section 3 so the former one is also in Ackermann. In Section 5 we present the proof of Theorem 3 which is our most technically involved contribution. We also use the idea of a regular lookahead and the result proved in Section 3 about k -deterministic VASSes. Many of the technically involved proofs are moved to the appendix.

2 Preliminaries

Basic notions. For $a, b \in \mathbb{N}$ we write $[a, b]$ to denote the set $\{a, a + 1, \dots, b - 1, b\}$. For a vector $v \in \mathbb{N}^d$ and $i \in [1, d]$ we write $v[i]$ to denote the i -th coordinate of vector v . By 0^d we denote the vector $v \in \mathbb{N}^d$ with all the coordinates equal to zero. For a word $w = a_1 \cdot \dots \cdot a_n$ and $1 \leq i \leq j \leq n$ we write $w[i..j] = a_i \cdot \dots \cdot a_j$ for the infix of w starting at position i and ending at position j . We also write $w[i] = a_i$. For any $1 \leq i \leq d$ by $e_i \in \mathbb{N}^d$ we denote the vector with all coordinates equal zero except of the i -th coordinate, which is equal to one. For a finite alphabet Σ we denote $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ the extension of Σ by the empty word ε .

Upward and downward-closed sets. For two vectors $u, v \in \mathbb{N}^d$ we say that $u \preceq v$ if for all $i \in [1, d]$ we have $u[i] \leq v[i]$. A set $S \subseteq \mathbb{N}^d$ is *upward-closed* if for each $u, v \in \mathbb{N}^d$ it holds that $u \in S$ and $u \preceq v$ implies $v \in S$. Similarly a set $S \subseteq \mathbb{N}^d$ is *downward-closed* if for each $u, v \in \mathbb{N}^d$ it holds that $u \in S$ and $v \preceq u$ implies $v \in S$. For $u \in \mathbb{N}^d$ we write $u \uparrow = \{v \mid u \preceq v\}$ for the set of all vectors bigger than u w.r.t. \preceq and $u \downarrow = \{v \mid v \preceq u\}$ for the set of all vectors smaller than u w.r.t. \preceq . If an upward-closed set is of the form $u \uparrow$ we call it an *up-atom*. Notice that if a one-dimensional set $S \subseteq \mathbb{N}$ is downward-closed then either $S = \mathbb{N}$ or $S = [0, n]$ for some $n \in \mathbb{N}$. In the first case we write $S = \omega \downarrow$ and in the second case $S = n \downarrow$. If a downward-closed set $D \subseteq \mathbb{N}^d$ is of a form $D = D_1 \times \dots \times D_d$, where all D_i for $i \in [1, d]$ are downward-closed one dimensional sets then we call D a *down-atom*. In the literature sometimes *up-atoms* are called principal filters and *down-atoms* are called ideals. If $D_i = (n_i) \downarrow$ then we also write $D = (n_1, n_2, \dots, n_d) \downarrow$. In that sense each down-atom is of a form $u \downarrow$ for $u \in (\mathbb{N} \cup \{\omega\})^d$. Notice that a down-atom does not have to be of a form $u \downarrow$ for $u \in \mathbb{N}^d$, for example $D = \mathbb{N}^d$ is not of this form, but $D = (\omega, \dots, \omega) \downarrow$.

The following two propositions will be helpful in our considerations.

► **Proposition 5** ([9] Lemma 17, [21], [13]). *Each downward-closed set in \mathbb{N}^d is a finite union of down-atoms. Similarly, each upward-closed set in \mathbb{N}^d is a finite union of up-atoms.*

We represent upward-closed sets as finite unions of up-atoms and downward-closed sets as finite unions of down-atoms, numbers are encoded in binary. The size of representation of upward- or downward-closed set S is denoted $\|S\|$. The following proposition helps to control the blowup of the representations of upward- and downward-closed sets.

► **Proposition 6.** *Let $U \subseteq \mathbb{N}^d$ be an upward-closed set and $D \subseteq \mathbb{N}^d$ be downward-closed set. Then the size of representation of their complements $\bar{U} = \mathbb{N}^d \setminus U$ and $\bar{D} = \mathbb{N}^d \setminus D$ is at most exponential wrt. the sizes $\|U\|$ and $\|D\|$, respectively and can be computed in exponential time.*

We prove the Proposition 6 in the appendix. For a more general study (for arbitrary well-quasi orders) see [15].

Vector Addition Systems with States. A d -dimensional Vector Addition System with States (d -VASS or simply VASS) V consists of a finite *alphabet* Σ , a finite set of *states* Q , a finite set of *transitions* $T \subseteq Q \times \Sigma \times \mathbb{Z}^d \times Q$, a distinguished *initial configuration* $c_I \in Q \times \mathbb{N}^d$, and a set of distinguished *final configurations* $F \subseteq Q \times \mathbb{N}^d$. We write $V = (\Sigma, Q, T, c_I, F)$. Sometimes we ignore some of the components in a VASS if they are not relevant, for example we write $V = (Q, T)$ if Σ , c_I , and F do not matter. *Configuration* of a d -VASS is a pair $(q, v) \in Q \times \mathbb{N}^d$, we often write it $q(v)$ instead of (q, v) . We write $\text{state}(q(v)) = q$. The set of all the configurations is denoted $\text{Conf} = Q \times \mathbb{N}^d$. For a state $q \in Q$ and a set $U \subseteq \mathbb{N}^d$ we write $q(U) = \{q(u) \mid u \in U\}$. A transition $t = (p, a, u, q) \in T$ can be *fired* in a configuration $r(v)$ if $p = r$ and $u + v \in \mathbb{N}^d$. We write then $p(v) \xrightarrow{t} q(u + v)$. We say that the transition $t \in T$ is *over* the letter $a \in \Sigma$ or the letter a *labels* the transition t . We write $p(v) \xrightarrow{a} q(u + v)$ slightly overloading the notation, when we want to emphasise that the transition is over the letter a . The *effect* of a transition $t = (p, a, u, q)$ is vector u , we write $\text{eff}(t) = u$. The size of VASS V is the total number of bits needed to represent the tuple (Σ, Q, T, c_I, F) , we do not specify here how we represent F as it may depend a lot on the form of F . A sequence $\rho = (c_1, t_1, c'_1), (c_2, t_2, c'_2), \dots, (c_n, t_n, c'_n) \in \text{Conf} \times T \times \text{Conf}$ is a *run* of VASS $V = (Q, T)$ if for all $i \in [1, n]$ we have $c_i \xrightarrow{t_i} c'_i$ and for all $i \in [1, n - 1]$ we have $c'_i = c_{i+1}$. We write $\text{trans}(\rho) = t_1 \cdot \dots \cdot t_n$. We extend the notion of the *labelling* to runs, labelling of a run ρ is

the concatenation of labels of its transitions. Such a run ρ is *from* the configuration c_1 to the configuration c'_n and configuration c'_n is *reachable* from configuration c_1 by the run ρ . We write then $c_1 \xrightarrow{\rho} c'_n$, $c_1 \xrightarrow{w} c'_n$ if w labels ρ slightly overloading the notation or simply $c_1 \rightarrow c'_n$ if the run ρ is not relevant, we say that the run ρ is *over* the word w .

VASS languages. A run ρ is *accepting* if it is from the initial configuration to some final configuration. For a VASS $V = (\Sigma, Q, T, c_I, F)$ we define the language of V as the set of all labellings of accepting runs, namely

$$L(V) = \{w \in \Sigma^* \mid c_I \xrightarrow{w} c_F \text{ for some } c_F \in F\}.$$

For any configuration c of V we define the *language of configuration c* , denoted $L_c(V)$ to be the language of VASS (Σ, Q, T, c, F) , namely the language of VASS V with the initial configuration c_I substituted by c . Sometimes we simply write $L(c)$ instead of $L_c(V)$ if V is clear from the context. Further, we say that the *configuration c has the empty language* if $L(c) = \emptyset$. For a VASS $V = (\Sigma, Q, T, c_I, F)$ its *control automaton* is intuitively VASS V after ignoring its counters. Precisely speaking, the control automaton is (Σ, Q, T', q_I, F') where $q_I = \text{state}(c_I)$, $F' = \{q \in Q \mid \exists v \in \mathbb{N}^d \ q(v) \in F\}$ and for each $(q, a, v, q') \in T$ we have $(q, a, q') \in T'$.

Notice that a 0-VASS, namely a VASS with no counters is just a finite automaton, so all the VASS terminology works also for finite automata. In particular, a configuration of a 0-VASS is simply an automaton state. In that special case for each state $q \in Q$ we call the $L(q)$ the language of state q .

A VASS is *deterministic* if for each configuration c reachable from the initial configuration c_I and for each letter $a \in \Sigma$ there is at most one configuration c' such that $c \xrightarrow{a} c'$. A VASS is *k-ambiguous* for $k \in \mathbb{N}$ if for each word $w \in \Sigma^*$ there are at most k accepting runs over w . If a VASS is 1-ambiguous we also call it *unambiguous*.

Note that, the set of languages accepted by unambiguous VASSes is a strict superset of the languages accepted by deterministic VASSes. To see that unambiguous VASSes can indeed accept more consider a language $(a^*b)^*a^n c^m$ where $n \geq m$. On one hand, an unambiguous VASS that accepts the language guesses where the last block of letter a starts, then it counts the number of a 's in this last block, and finally, it counts down reading c 's. As there is exactly one correct guess this VASS is indeed unambiguous. On the other hand, deterministic system can not accept the language, as intuitively speaking it does not know whether the last block of a 's has already started or not. To formulate the argument precisely one should use rather easy pumping techniques.

The following two problems are the main focus of this paper, for different subclasses of VASSes:

Inclusion problem for VASSes

Input Two VASSes V_1 and V_2 .

Question Does $L(V_1) \subseteq L(V_2)$?

Equivalence problem for VASSes

Input Two VASSes V_1 and V_2 .

Question Does $L(V_1) = L(V_2)$?

In the sequel, we are mostly interested in VASSes with the set of final configurations F of some special form. We extend the order \preceq on vectors from \mathbb{N}^d to configurations from $Q \times \mathbb{N}^d$ in a natural way: we say that $q_1(v_1) \preceq q_2(v_2)$ if $q_1 = q_2$ and $v_1 \preceq v_2$. We define the notions of upward-closed, downward-closed, up-atom and down-atom the same as for vectors. As Proposition 5 holds for any well quasi-order, it applies also to $Q \times \mathbb{N}^d$. Proposition 6 applies here as well, as the upper bound on the size can be shown separately for each state. Let the set of final configurations of VASS V be F . If F is upward-closed then we call V an *upward-VASS*. If F is downward-closed then we call V a *downward-VASS*. For two sets

$A \subseteq \mathbb{N}^a$, $B \subseteq \mathbb{N}^b$ and a subset of coordinates $J \subseteq [1, a + b]$ by $A \times_J B$ we denote the set of vectors in \mathbb{N}^{a+b} which projected into coordinates in J belong to A and projected into coordinates outside J belong to B . If $F = \bigcup_{i \in [1, n]} q_i(U_i \times_{J_i} D_i)$ where for all $i \in [1, n]$ we have $J_i \subseteq [1, d]$, $U_i \subseteq \mathbb{N}^{|J_i|}$ are up-atoms and $D_i \subseteq \mathbb{N}^{d-|J_i|}$ are down-atoms then we call V an *updown-VASS*. In the sequel we write simply \times instead of \times_J , as the set of coordinates J is never relevant. If $F = \{c_F\}$ is a singleton then we call V a *singleton-VASS*. As in this paper we mostly work with upward-VASSes we often say simply a VASS instead of an upward-VASS. In other words, if not indicated otherwise we assume that the set of final configurations F is upward-closed.

For the complexity analysis we assume that whenever F is upward- or downward-closed then it is given as a union of atoms. If $F = \bigcup_{i \in [1, n]} q_i(U_i \times D_i)$ then in the input we get a sequence of q_i and representations of atoms U_i, D_i defining individual sets $q_i(U_i \times D_i)$.

Language emptiness problem for VASSes. The following emptiness problem is the central problem for VASSes.

Emptiness problem for VASSes

Input A VASS $V = (\Sigma, Q, T, c_I, F)$

Question Does $c_I \rightarrow c_F$ in V for some $c_F \in F$?

Observe that the emptiness problem is not influenced in any way by labels of the transitions, so sometimes we will not even specify transition labels when we work with the emptiness problem. If we want to emphasise that labels of transitions do not matter for some problem then we write $V = (Q, T, c_I, F)$ ignoring the Σ component. In such cases we also assume that transitions do not contain the Σ component, namely $T \subseteq Q \times \mathbb{Z}^d \times Q$.

Note also that the celebrated reachability problem and the coverability problem for VASSes are special cases of the emptiness problem. The reachability problem is the case when F is a singleton set $\{c_F\}$, classically it is formulated as the question whether there is a run from c_I to c_F . The coverability problem is the case when F is an up-atom c_F , classically it is formulated as the question whether there is a run from c_I to any c such that $c_F \preceq c$. Recall that the reachability problem, so the emptiness problem for singleton-VASSes is in Ackermann [24] and actually Ackermann-complete [23, 11].

A special case of the emptiness problem is helpful for us in Section 3.

► **Lemma 7.** *The emptiness problem for VASSes with the acceptance condition $F = q_F(U \times D)$ where D is a down-atom and U is an up-atom is in Ackermann.*

We prove Lemma 7 in the appendix. The following is a simple and useful corollary of Lemma 7.

► **Corollary 8.** *The emptiness problem for updown-VASSes is in Ackermann.*

Proof. Recall that for updown-VASSes the acceptance condition is a finite union of $q(U \times D)$ for some up-atom $U \subseteq \mathbb{N}^{d_1}$ and down-atom $D \subseteq \mathbb{N}^{d_2}$ where d_1 and d_2 sums to the dimension of the VASS V . Thus emptiness of the updown-VASS can be reduced to finitely many emptiness queries of the form $q(U \times D)$ which can be decided in Ackermann due to Lemma 7. Notice that the number of queries is not bigger than the size of the representation of F thus the emptiness problem for updown-VASSes is also in Ackermann. ◀

By Proposition 5 each downward-VASS is also an updown-VASS, thus Corollary 8 implies the following one.

► **Corollary 9.** *The emptiness problem for downward-VASSes is in Ackermann.*

Recall that the coverability problem in VASSes is in ExpSpace [29], and the coverability problem is equivalent to the emptiness problem for the set of final configurations being an up-atom. By Proposition 5 we have the following simple corollary which creates an elegant duality for the emptiness problems in VASSes.

► **Corollary 10.** *The emptiness problem for upward-VASSes is in ExpSpace.*

Actually, even the following stronger fact is true and helpful for us in the remaining part of the paper, it is shown in [22].

► **Proposition 11.** *For each upward-VASS the representation of the downward-closed set of configurations with the empty language can be computed in doubly-exponential time.*

3 Deterministic VASSes

3.1 Lower bound

First we prove a lemma, which easily implies Theorem 1.

► **Lemma 12.** *For each d -dimensional singleton-VASS V with final configuration being $c_F = q_F(0^d)$ one can construct in polynomial time two deterministic $(d + 1)$ -dimensional upward-VASSes V_1 and V_2 such that*

$$L(V_1) = L(V_2) \iff L(V) = \emptyset.$$

The sketch of the proof. To prove the lemma we take V and we add to it one transition labelled with a new letter. In V_1 the added transition can be performed if we have reached a configuration bigger than or equal to c_F . In V_2 the added transition can be performed only if we have reached a configuration strictly bigger than c_F . Now it is easy to see that $L(V_1) \neq L(V_2)$ if and only if c_F can be reached. Detailed proof is in the appendix.

Notice that Lemma 12 shows that the emptiness problem for a singleton-VASS with the final configuration having zero counter values can be reduced in polynomial time to the language equivalence for deterministic VASSes. This proves Theorem 1 as the emptiness problem, even with zero counter values of the final configuration is Ackermann-hard [23, 11].

3.2 Upper bounds

In this Section we prove three results of the form: if V_1 is a VASS and V_2 is a VASS of some special type then deciding whether $L(V_1) \subseteq L(V_2)$ is in Ackermann. Our approach to these problems is the same, namely we first prove that complement of $L(V_2)$ for V_2 of the special type is also a language of some VASS V_2' . Then to decide the inclusion problem it is enough to construct VASS V such that $L(V) = L(V_1) \cap L(V_2') = L(V_1) \setminus L(V_2)$ and check it for emptiness. In the description above using the term VASS we do not specify the form of its set of accepting configurations. Starting from now on we call upward-VASSes simply VASSes and for VASSes with other acceptance conditions we use their full name (like downward-VASSes or updown-VASSes) to distinguish them from upward-VASSes. The following lemma is very useful in our strategy of deciding the inclusion problem for VASS languages.

► **Lemma 13.** *For a VASS V_1 and a downward-VASS V_2 one can construct in polynomial time an updown-VASS V such that $L(V) = L(V_1) \cap L(V_2)$.*

The proof is in the appendix.

Deterministic VASSes. We first show the following theorem that will be generalised by the other results in this section. We aim to prove it independently in order to mildly introduce our techniques.

► **Theorem 14.** *For a deterministic VASS one can build in exponential time a downward-VASS which recognises the complement of its language.*

Sketch of the proof. A word may be in the complement of our VASS language for the following reasons: (1) the run reaches a configuration that is not accepted, (2) the run does not exist as one of the counters would drop below zero, (3) the run is not possible due to the structure of the control automaton. For each case we separately design a part of a downward-VASS accepting it. Cases (1) and (3) are simple. For the case (2) we nondeterministically guess the moment when the run would go below zero and freeze the configuration at that moment. Then at the end of the word we check if our guess was correct. Notice that the set of configurations from which a step labelled with a letter a would take a counter below zero is downward-closed, so we can check the correctness of our guess using a downward-closed accepting condition. Detailed proof is in the appendix.

The following theorem is a simple corollary of Theorem 14, Lemma 13 and Corollary 8.

► **Theorem 15.** *The inclusion problem of a VASS language in a deterministic VASS language is in Ackermann.*

Deterministic VASSes with holes. We define here VASSes with holes, which are a useful tool to obtain our results about unambiguous VASSes in Section 4. A d -VASS with holes (or shortly d -HVASS) V is defined exactly as a standard VASS, but with an additional downward-closed set $H \subseteq Q \times \mathbb{N}^d$ which affects the semantics of V . Namely the set of configurations of V is $Q \times \mathbb{N}^d \setminus H$. Thus each configuration on a run of V needs not only to have nonnegative counters, but in addition to that it can not be in the set of holes H . Additionally in HVASSes we allow for transitions labelled by the empty word ε , in contrast to the rest of our paper. Due to that fact in this paragraph we often work also with VASSes having ε -labelled transitions, we call such VASSes the ε -VASSes. As an illustration of the HVASS notion let us consider the zero-dimensional case. In that case the set of holes is just a subset of states. Therefore HVASSes in dimension zero are exactly VASSes in dimension zero, so finite automata. However, for higher dimensions the notions of HVASSes and VASSes differ.

We present here a few results about languages for HVASSes. First notice that for nondeterministic HVASSes it is easy to construct a language equivalent ε -VASS.

► **Lemma 16.** *For each HVASS one can compute in exponential time a language equivalent ε -VASS.*

Sketch of the proof. First we observe that the complement of the set of holes is an upward-closed set U . The idea behind the construction is that after every step we test if the current configuration is in U . We nondeterministically guess a minimal element x_i of U above which the current configuration is, then we subtract x_i and add it back. If our guess was not correct then the run is blocked.

It is important to emphasise that the above construction applied to a deterministic HVASS does not give us a deterministic VASS, so we cannot simply reuse Theorem 14. Thus in order to prove the decidability of the inclusion problem for HVASSes we need to generalise Theorem 14 to HVASSes.

► **Theorem 17.** *For a deterministic HVASS one can compute in exponential time a downward- ε -VASS which recognises the complement of its language.*

Sketch of the proof. The proof is very similar to the proof of Theorem 14. In the case (1) we have to check if the accepting run stays above the holes, do perform it we use the trick from Lemma 16. In the case (2) we freeze the counter when the run would have to drop below zero or enter the hole. The case (3) is the same as in Theorem 14.

Now the following theorem is an easy consequence of the shown facts. We need only to observe that proofs of Lemma 13 and Corollary 8 work as well for ε -VASSes.

► **Theorem 18.** *The inclusion problem of an HVASS language in a deterministic HVASS language is in Ackermann.*

Boundedly-deterministic VASSes. We define here a generalisation of a deterministic VASS, namely a k -deterministic VASS for $k \in \mathbb{N}$. Such VASSes are later used as a tool for deriving results about k -ambiguous VASSes in Section 5.

A VASS $V = (\Sigma, Q, T, c_I, F)$ is k -deterministic if for each word $w \in \Sigma^*$ there are at most k maximal runs over w . We call a run ρ a maximal run over w if either (1) it is a run over w or (2) $w = uav$ for $u, v \in \Sigma^*$, $a \in \Sigma$ such that the run ρ is over the prefix u of w but there is no possible way of extending ρ by any transition labelled with the letter $a \in \Sigma$. Let us emphasise that here we count runs in a subtle way. We do not count only the maximal number of active runs throughout the word but the total number of different runs which have ever been started during the word. To illustrate the difference better let us consider an example 0-VASS (a finite automaton) V over $\Sigma = \{a, b\}$ with two states p, q and with three transitions: (p, a, p) , (p, a, q) and (q, b, q) . Then V has $n + 1$ maximal runs over the word a^n although only two of these runs actually survive till the end of the input word. So V is not 2-deterministic even though for each input word it has at most two runs.

► **Theorem 19.** *For a k -deterministic d -VASS one can build in exponential time a $(k \cdot d)$ -dimensional downward-VASS which recognises the complement of its language.*

Sketch of the proof. In the construction $(k \cdot d)$ -dimensional downward-VASS V' simulates k copies of V which take care of at most k different maximal runs of V . The accepting condition F' of V' verifies whether in all the copies there is a reason that the simulated maximal runs do not accept. The reasons why each individual copy do not accepts are the same as in Theorem 14.

Theorem 19 together with Lemma 13 and Corollary 8 easily implies (analogously as in the proof of Theorem 18) the following theorem.

► **Theorem 20.** *The inclusion problem of a VASS language in a k -deterministic VASS language is in Ackermann.*

4 Unambiguous VASSes

In this section we aim to prove Theorem 2. However, possibly a more valuable contribution of this section is a novel technique which we introduce in order to show Theorem 2. The essence of this technique is to introduce a regular lookahead to words, namely to decorate each letter of a word with a piece of information regarding some regular properties of the suffix of this word. For technical reasons we realise it by the use of finite monoids.

The high level intuition behind the proof of Theorem 2 is the following. We first introduce the notion of (M, h) -decoration of words, languages and VASSes, where M is a monoid and $h : \Sigma^* \rightarrow M$ is a homomorphism. Proposition 23 states that language inclusion of

16:10 Language Inclusion for Boundedly-Ambiguous Vector Addition Systems Is Decidable

two VASSes can be reduced to language inclusion of its decorations. On the other hand Theorem 26 shows that for appropriately chosen pair (M, h) the decorations of unambiguous VASSes are deterministic HVASSes. Theorem 25 states that such an appropriate pair can be computed quickly enough. Thus language inclusion of unambiguous VASSes reduces to language inclusion of deterministic HVASSes, which is in Ackermann due to Theorem 18.

Recall that a monoid M together with a homomorphism $h : \Sigma^* \rightarrow M$ and an accepting subset $F \subseteq M$ recognises a language L if $L = h^{-1}(F)$. In other words L is exactly the set of words w such that $h(w) \in F$. The following proposition is folklore, for details see [28] (Proposition 3.12).

► **Proposition 21.** *A language of finite words is regular if and only if it is recognised by some finite monoid.*

For that reason monoids are a good tool for working with regular languages. In particular Proposition 21 implies that for each finite family of regular languages there is a monoid, which recognises all of them, this fact is useful in Theorem 26. Let us fix a finite monoid M and a homomorphism $h : \Sigma^* \rightarrow M$. For a word $w = a_1 \cdot \dots \cdot a_n \in \Sigma^*$ we define its (M, h) -decoration to be the following word over an alphabet $\Sigma_\varepsilon \times M$:

$$(\varepsilon, h(a_1 \cdot \dots \cdot a_n)) \cdot (a_1, h(a_2 \cdot \dots \cdot a_n)) \cdot \dots \cdot (a_{n-1}, h(a_n)) \cdot (a_n, h(\varepsilon)).$$

In other words, the (M, h) -decoration of a word w of length n has length $n + 1$, where the i -th letter has the form $(a_{i-1}, h(a_i \cdot \dots \cdot a_n))$. We denote the (M, h) -decoration of a word w as $w_{(M, h)}$. If $h(w) = m$ then we say that word w has *type* $m \in M$. The intuition behind the (M, h) -decoration of w is that for each language L which is recognised by the pair (M, h) the i -th letter of w is extended with an information whether the suffix of w after this letter belongs to L or does not belong. This information can be extracted from the monoid element $h(a_{i+1} \cdot \dots \cdot a_n)$ by which letter a_i is extended. As an illustration consider words over alphabet $\Sigma = \{a, b\}$, monoid $M = \mathbb{Z}_2$ counting modulo two and homomorphism $h : \Sigma \rightarrow M$ defined as $h(a) = 1$, $h(b) = 0$. In that case for each $w \in \Sigma^*$ the element $h(w)$ indicates whether the number of letters a in the word w is odd or even. The decoration of $w = aabab$ is then $w_{(M, h)} = (\varepsilon, 1)(a, 0)(a, 1)(b, 1)(a, 0)(b, 0)$.

We say that a word $u \in (\Sigma_\varepsilon \times M)^*$ is *well-formed* if $u = (\varepsilon, m_0) \cdot (a_1, m_1) \cdot \dots \cdot (a_n, m_n)$ such that all $a_i \in \Sigma$, and for each $i \in [0, n]$ the type of $a_{i+1} \cdot \dots \cdot a_n$ is m_i (in particular type of ε is m_n). We say that such a word u *projects* into word $a_1 \cdot \dots \cdot a_n$. It is easy to observe that $w_{(M, h)}$ is the only well-formed word that projects into w . The following proposition is useful in Section 5, an appropriate finite automaton can be easily constructed.

► **Proposition 22.** *The set of all well-formed words is regular.*

A word is *almost well-formed* if it satisfies all the conditions of well-formedness, but the first letter is not necessarily of the form (ε, m) for $m \in M$, it can as well belong to $\Sigma \times M$.

The (M, h) -decoration of a language L , denoted $L_{(M, h)}$, is the set of all (M, h) -decorations of all words in L , namely

$$L_{(M, h)} = \{w_{(M, h)} \mid w \in L\}.$$

As the (M, h) -decoration is a function from words over Σ to words over $\Sigma_\varepsilon \times M$ we observe that $u = v$ iff $u_{(M, h)} = v_{(M, h)}$ and clearly the following proposition holds.

► **Proposition 23.** *For each finite alphabet Σ , two languages $K, L \subseteq \Sigma^*$, monoid M and homomorphism $h : \Sigma^* \rightarrow M$ we have*

$$K \subseteq L \iff K_{(M, h)} \subseteq L_{(M, h)}.$$

Recall now that HVASS (VASS with holes) is a VASS with some downward-closed set H of prohibited configurations (see Section 3, paragraph Deterministic VASSES with holes). For each d -VASS $V = (\Sigma, Q, T, c_I, F)$, a monoid M and a homomorphism $h : \Sigma^* \rightarrow M$ we can define in a natural way a d -HVASS $V_{(M,h)} = (\Sigma_\varepsilon \times M, Q', T', c'_I, F')$ accepting the (M, h) -decoration of $L(V)$. The set of states Q' equals $Q \times (M \cup \{\perp\})$. The intuition is that $V_{(M,h)}$ is designed in such a way that for any state $(q, m) \in Q \times M$ and vector $v \in \mathbb{N}^d$ if $(q, m)(v) \xrightarrow{w'} F'$ then w' is almost well-formed and w' projects into some $w \in \Sigma^*$ such that $h(w) = m$. If $c_I = q_I(v_I)$ then configuration $c'_I = (q_I, \perp)(v_I)$ is the initial configuration of $V_{(M,h)}$. The set of final configurations F' is defined as $F' = \{(q, h(\varepsilon))(v) \mid q(v) \in F\}$. Finally we define the set of transitions T' of V' as follows. First, for each $m \in M$ we add the following transition $((q_I, \perp), (\varepsilon, m), 0^d, (q_I, m))$ to T' . Then for each transition $(p, a, v, q) \in T$ and for each $m \in M$ we add to T' the transition (p', a', v, q') where $a' = (a, m)$, $q' = (q, m)$ and $p' = (p, h(a) \cdot m)$. It is now easy to see that for any word $w = a_1 \cdot \dots \cdot a_n \in \Sigma^*$ we have

$$q_I(v_I) \xrightarrow{a_1} q_1(v_1) \xrightarrow{a_2} \dots \xrightarrow{a_{n-1}} q_{n-1}(v_{n-1}) \xrightarrow{a_n} q_n(v_n)$$

if and only if

$$(q_I, \perp)(v_I) \xrightarrow{(\varepsilon, m_1)} (q_I, m_1)(v_I) \xrightarrow{(a_1, m_2)} (q_1, m_2)(v_1) \xrightarrow{(a_2, m_3)} \dots \\ \xrightarrow{(a_{n-1}, m_n)} (q_{n-1}, m_n)(v_{n-1}) \xrightarrow{(a_n, m_{n+1})} (q_n, m_{n+1})(v_n),$$

where $m_i = h(w[i..n])$ for all $i \in [1, n+1]$, in particular $m_{n+1} = h(\varepsilon)$. Therefore indeed $L(V_{(M,h)}) = L(V)_{(M,h)}$. Till now the defined HVASS is actually a VASS, we have not defined any holes. Our aim is now to remove configurations with the empty language, namely $(q, m)(v)$ for which there is no word $w \in (\Sigma_\varepsilon \times M)^*$ such that $(q, m)(v) \xrightarrow{w} c'_F$ for some $c'_F \in F'$. Notice that as F' is upward-closed we know that the set of configurations with the empty language is downward-closed. This is how we define the set of holes H , it is exactly the set of configurations with the empty language. We can compute the set of holes in doubly-exponential time by Proposition 11.

By Proposition 23 we know that for two VASSES U, V we have $L(U) \subseteq L(V)$ if and only if $L(U_{(M,h)}) \subseteq L(V_{(M,h)})$. This equivalence is useful as we show in a moment that for an unambiguous VASS V and suitably chosen (M, h) the HVASS $V_{(M,h)}$ is deterministic.

Regular separability. We use here the notion of regular separability. We say that two languages $K, L \subseteq \Sigma^*$ are *regular-separable* if there exists a regular language $S \subseteq \Sigma^*$ such that $K \subseteq S$ and $S \cap L = \emptyset$. We say that S *separates* K and L and S is a *separator* of K and L . We recall here a theorem about regular-separability of VASS languages (importantly upward-VASS languages, not downward-VASS languages) from [9].

► **Theorem 24** (Theorem 24 in [9]). *For any two VASS languages $L_1, L_2 \subseteq \Sigma^*$ if $L_1 \cap L_2 = \emptyset$ then L_1 and L_2 are regular-separable and one can compute the regular separator in elementary time.*

16:12 Language Inclusion for Boundedly-Ambiguous Vector Addition Systems Is Decidable

Proof. Theorem 24 in [9] says that there exists a regular separator of L_1 and L_2 of size at most triply-exponential. In order to compute it we can simply enumerate all the possible separators of at most triply-exponential size and check them one by one. For a given regular language and a given VASS language by Proposition 10 one can check in doubly-exponential time whether they nonempty intersect. \blacktriangleleft

For our purposes we need a bit stronger version of this theorem. We say that a family of regular languages \mathcal{F} *separates languages of a VASS* V if for any two configurations c_1, c_2 such that languages $L(c_1)$ and $L(c_2)$ are disjoint there exists a language $S \in \mathcal{F}$ that separates $L(c_1)$ and $L(c_2)$.

► **Theorem 25.** *For any VASS one can compute in an elementary time a finite family of regular languages which separates its languages.*

Proof. Let us fix a d -VASS $V = (\Sigma, Q, T, c_I, F)$. Let us define the set of pairs of configurations of V with disjoint languages $D = \{(c_1, c_2) \mid L(c_1) \cap L(c_2) = \emptyset\} \subseteq Q \times \mathbb{N}^d \times Q \times \mathbb{N}^d$. One can easily see that the set D is exactly the set of configurations with empty language in the synchronised product of VASS V with itself. Thus by Proposition 11 we can compute in doubly-exponential time its representation as a finite union of down-atoms $D = A_1 \cup \dots \cup A_n$. We show now that for each $i \in [1, n]$ one can compute in elementary time a regular language S_i such that for all $(c_1, c_2) \in A_i$ the language S_i separates $L(c_1)$ and $L(c_2)$. This will finish the proof showing that one of S_1, \dots, S_n separates $L(c_1)$ and $L(c_2)$ whenever they are disjoint.

Let $A \subseteq Q \times \mathbb{N}^d \times Q \times \mathbb{N}^d$ be a down-atom. Therefore $A = D_1 \times D_2$ where $D_1 = p_1(u_1 \downarrow)$ and $D_2 = p_2(u_2 \downarrow)$ for some $u_1, u_2 \in (\mathbb{N} \cup \{\omega\})^d$. Let $L_1 = \bigcup_{c \in D_1} L(c)$ and $L_2 = \bigcup_{c \in D_2} L(c)$. Languages L_1 and L_2 are disjoint as $w \in L_1 \cap L_2$ would imply $w \in L(c_1) \cap L(c_2)$ for some $c_1 \in D_1$ and $c_2 \in D_2$. Now observe that L_1 is not only an infinite union of VASS languages but also a VASS language itself. Indeed, let $V_1 = (\Sigma, Q, T_1, c_I, F_1)$ be the VASS V where all coordinates $i \in [1, d]$ such that $u_1[i] = \omega$ are ignored. Concretely,

- $(p, a, v_1, q) \in T_1$ if there exists $(p, a, v, q) \in T$ such that for every i holds either $v_1[i] = v[i]$ or $v_1[i] = 0$ and $u_1[i] = \omega$,
- $(q, v_1) \in F_1$ if there exists $(q, v) \in F$ such that for every i holds either $v_1[i] = v[i]$ or $u_1[i] = \omega$.

Then it is easy to observe that V_1 accepts exactly the language L_1 . Similarly one can define VASS V_2 accepting the language L_2 . By Theorem 24 we can compute in elementary time some regular separator S of $L(V_1)$ and $L(V_2)$. It is now easy to see that for any configurations $c_1 \in D_1$ and $c_2 \in D_2$ languages $L(c_1)$ and $L(c_2)$ are separated by S . \blacktriangleleft

Now we are ready to use the notion of (M, h) -decoration of a VASS language. Let us recall that a regular language L is recognised by a monoid M and homomorphism $h : \Sigma^* \rightarrow M$ if there is $F \subseteq M$ such that $L = h^{-1}(F)$.

► **Theorem 26.** *Let V be an unambiguous VASS over Σ and \mathcal{F} be a finite family of regular languages separating languages of V . Suppose M is a monoid with homomorphism $h : \Sigma^* \rightarrow M$ recognising every language in \mathcal{F} . Then the HVASS $V_{(M,h)}$ is deterministic.*

Proof. Let $V = (\Sigma, Q, T, c_I, F)$ and let $c_I = q_I(v_I)$. We aim to show that HVASS $V_{(M,h)} = (\Sigma', Q', T', c'_I, F')$ is deterministic, where $\Sigma' = \Sigma_\varepsilon \times M$ and $Q' = Q \times (M \cup \{\perp\})$. It is easy to see from the definition of $V_{(M,h)}$ that for each $(a, m) \in \Sigma'$ and each $q \in Q$ the state (q, \perp) has at most one outgoing transition over (a, m) . Indeed, there is exactly one transition over (ε, m) outgoing from (q_I, \perp) and no outgoing transitions in the other cases. Assume now

towards a contradiction that $V_{(M,h)}$ is not deterministic. Then there is some configuration $c = (q, m)(v)$ with $(q, m) \in Q \times M$ such that $c_I \xrightarrow{u} c$ for some word u over Σ' and a letter $(a, m') \in \Sigma'$ such that a transition from c over (a, m') leads to some two configurations $c_1 = (q_1, m')(v_1)$ and $c_2 = (q_2, m')(v_2)$. Recall that a transition over (a, m') has to lead to some state with the second component equal m' . As configurations with empty language are not present in $V_{(M,h)}$ we know that there exist words $w_1 \in L(c_1)$ and $w_2 \in L(c_2)$. Recall that as $c_1 = (q_1, m')(v_1)$ and $c_2 = (q_2, m')(v_2)$ we have $h(w_1) = m' = h(w_2)$. We show now that $L(c_1)$ and $L(c_2)$ are disjoint. Assume otherwise that there exists $w \in L(c_1) \cap L(c_2)$. Then there are at least two accepting runs over the word $u \cdot (a, m') \cdot w$ in $V_{(M,h)}$. This means however that there are at least two accepting runs over the projection of $u \cdot (a, m') \cdot w$ in V , which contradicts unambiguity of V . Thus $L(c_1)$ and $L(c_2)$ are disjoint and therefore separable by some language from \mathcal{F} . Recall that all the languages in \mathcal{F} are recognisable by (M, h) thus words from $L(c_1)$ should be mapped by the homomorphism h to different elements of M than words from $L(c_2)$. However $h(w_1) = m'$ for $w_1 \in L(c_1)$ and $h(w_2) = m'$ for $w_2 \in L(c_2)$ which leads to the contradiction. ◀

Now we are ready to prove Theorem 2. Let V_1 be a VASS and V_2 be an unambiguous VASS, both with labels from Σ . We first compute a finite family \mathcal{F} separating languages of V_2 which can be performed in elementary time by Theorem 25 and then we compute a finite monoid M together with a homomorphism $h : \Sigma^* \rightarrow M$ recognising all the languages from \mathcal{F} . By Proposition 23 we get that $L(V_1) \subseteq L(V_2)$ if and only if $L_{(M,h)}(V_1) \subseteq L_{(M,h)}(V_2)$. We now compute HVASSes $V'_1 = V_{1(M,h)}$ and $V'_2 = V_{2(M,h)}$. By Theorem 26 the HVASS V'_2 is deterministic. Thus it remains to check whether the language of a HVASS V'_1 is included in the language of a deterministic HVASS V'_2 , which is in Ackermann due to Theorem 18.

► **Remark 27.** We remark that our technique can be applied not only to VASSes but also in a more general setting of well-structured transition systems. In [9] it was shown that for any well-structured transition systems fulfilling some mild conditions (finite branching is enough) disjointness of two languages implies regular separability of these languages. We claim that an analogue of our Theorem 25 can be obtained in that case as well. Assume now that $\mathcal{W}_1, \mathcal{W}_2$ are two classes of finitely branching well-structured transition systems, such that for any two systems $V_1 \in \mathcal{W}_1, V_2 \in \mathcal{W}_2$ where V_2 is deterministic the language inclusion problem is decidable. Then this problem is also likely to be decidable if we weaken the condition of determinism to unambiguity. More concretely speaking this seems to be the case if it is possible to perform the construction analogous to Theorem 14 in \mathcal{W}_2 , namely if one can compute the system recognising the complement of deterministic language without leaving the class \mathcal{W}_2 . We claim that an example of such a class \mathcal{W}_2 is the class of VASSes with one reset. The emptiness problem for VASSes with one zero-test (and thus also for VASSes with one reset) is decidable due to [30, 4]. Then following our techniques it seems that one can show that inclusion of a VASS language in a language of an unambiguous VASS with one reset is decidable.

5 Boundedly-ambiguous VASSes

In this section we aim to prove Theorem 3. It is an easy consequence of the following theorem.

► **Theorem 28.** *For any $k \in \mathbb{N}$ and a k -ambiguous VASS one can build in elementary time a downward-VASS which recognises the complement of its language.*

Let us show how Theorem 28 implies Theorem 3. Let V_1 be a VASS and V_2 be a k -ambiguous VASS. By Theorem 28 one can compute in elementary time a downward-VASS V'_2 such that $L(V'_2) = \Sigma^* \setminus L(V_2)$. By Lemma 13 one can construct in time polynomial wrt.

the size of V_1 and V_2' an updown-VASS V such that $L(V) = L(V_1) \cap L(V_2') = L(V_1) \setminus L(V_2)$. By Corollary 8 emptiness of V is decidable in Ackermann which in consequence proves Theorem 3.

Thus the rest of this section focuses on the proof of Theorem 28.

Proof of Theorem 28. We prove now Theorem 28 using Lemmas 29 and 30 stated below. Then in Sections D and D in the appendix we prove the formulated lemmas. Let V be a k -ambiguous VASS over an alphabet Σ . In the proof we construct a sequence of VASSes V^1, V^2, \dots, V^6 related in various ways to V with the property that V^6 is a downward-VASS and $L(V^6)$ is exactly the complement of $L(V)$. More concretely $L(V^1)$ equals $L(V)$, $L(V^2)$ is a decoration of $L(V)$, $L(V^3)$ is the complement of $L(V^2)$, while V^4, V^5 recognise more sophisticated languages related to $L(V^3)$.

First due to Lemma 29 proved in Section D we construct a VASS V^1 which is language equivalent to V and additionally has the control automaton being k -ambiguous.

► **Lemma 29.** *For each k -ambiguous VASS V one can construct in doubly-exponential time a language equivalent VASS V' with the property that its control automaton is k -ambiguous.*

Now our aim is to get a k -deterministic VASS V^2 which is language equivalent to V^1 . We are not able to achieve it literally, but using the notion of (M, h) -decoration from Section 4 we can compute a somehow connected k -deterministic VASS V^2 . We use the following lemma which is proved in Section D.

► **Lemma 30.** *Let $\mathcal{A} = (\Sigma, Q, T, q, F)$ be a k -ambiguous finite automaton for some $k \in \mathbb{N}$. Let M be a finite monoid and $h : \Sigma^* \rightarrow M$ be a homomorphism recognising all the state languages of the automaton \mathcal{A} . Then the decoration $\mathcal{A}_{(M, h)}$ is a k -deterministic finite automaton.*

Now we consider the control automaton \mathcal{A} of VASS V^1 . We compute a monoid M together with a homomorphism $h : \Sigma^* \rightarrow M$ which recognises all the state languages of \mathcal{A} . Then we construct the automaton $\mathcal{A}_{(M, h)}$. Note that the decoration of a VASS produces an HVASS, but as we decorate an automaton i.e. 0-VASS we get a 0-HVASS which is also a finite automaton. Based on $\mathcal{A}_{(M, h)}$ we construct a VASS V^2 . We add a vector to every transition in $\mathcal{A}_{(M, h)}$ to produce a VASS that recognises the (M, h) -decoration of the language of VASS V^1 . Precisely, if we have a transition $((p, m), (a, m'), (q, m'))$ in $\mathcal{A}_{(M, h)}$ then it is created from the transition (p, a, q) in \mathcal{A} , which originates from the transition (p, a, v, q) in V^1 . So in V^2 we label $((p, m), (a, m'), (q, m'))$ with v i.e. we have the transition $((p, m), (a, m'), v, (q, m'))$. Similarly, based on V^1 , we define initial and final configurations in V^2 . It is easy to see that there is a bijection between accepting runs in V^1 and accepting runs in V^2 . By Lemma 30 $\mathcal{A}_{(M, h)}$ is k -deterministic which immediately implies that V^2 is k -deterministic as well.

Now by Theorem 19 we compute a downward-VASS V^3 which recognises the complement of $L(V^2)$. Notice that for each $w \in \Sigma^*$ there is exactly one well-formed word in $\Sigma_\varepsilon \times M$ which projects into w , namely the (M, h) -decoration of w . Therefore V^3 accepts all the not well-formed words and all the well-formed words which project into the complement of $L(V)$. By Proposition 22 the set of all well-formed words is recognised by some finite automaton \mathcal{B} . Computing a synchronised product of \mathcal{B} and V^3 one can obtain a downward-VASS V^4 which recognises the intersection of languages $L(\mathcal{B})$ and $L(V^3)$, namely all the well-formed words which project into the complement of $L(V)$. It is easy now to compute a downward- ε -VASS V^5 recognising the projection of $L(V^4)$ into the first component of the alphabet $\Sigma_\varepsilon \times M$. We obtain V^5 just by ignoring the second component of the alphabet. Thus V^5 recognises exactly the complement of $L(V)$. However V^5 is not a downward-VASS as it contains a few ε -labelled transitions leaving the initial state. We aim to eliminate these ε -labelled transitions. Recall that in the construction of the (M, h) -decoration the (ε, m) -labelled

transitions leaving the initial configuration have effect 0^d . Thus it is easy to eliminate them and obtain a downward-VASS V^6 which recognises exactly the complement of $L(V)$, which finishes the proof of Theorem 28. Let us remark here that even ignoring the last step of elimination and obtaining a downward- ε -VASS recognising the complement of $L(V)$ would be enough to prove Theorem 3 along the same lines as it is proved now. ◀

6 Future research

VASSes accepting by configuration. In our work we prove Theorem 28 stating that for a k -ambiguous upward-VASS one can compute a downward-VASS recognising the complement of its language. This theorem implies all our upper bound results, namely decidability of language inclusion of an upward-VASS in a k -ambiguous upward-VASS and language equivalence of k -ambiguous upward-VASSes. The most natural question which can be asked in this context is whether Theorem 28 or some of its consequences generalises to singleton-VASSes (so VASSes accepting by a single configuration) or more generally to downward-VASSes. Our results about complementing deterministic VASSes apply also to downward-VASSes. However generalising our results for nondeterministic (but k -ambiguous or unambiguous) VASSes encounter essential barriers. Techniques from Section 4 do not work as the regular-separability result from [9] applies only to upward-VASSes. Techniques from Section 5 break as the proof of Lemma 29 essentially uses the fact that the acceptance condition is upward-closed. Thus it seems that one would need to develop novel techniques to handle the language equivalence problem for unambiguous VASSes accepting by configuration.

Weighted models. Efficient decidability procedures for language equivalence were obtained for finite automata and for register automata with the use of weighted models [31, 3]. For many kinds of systems one can naturally define weighted models by adding weights and computing value of a word in the field $(\mathbb{Q}, +, \cdot)$. Decidability of equivalence for weighted models easily implies language equivalence for unambiguous models as accepted words always have the output equal one while rejected words always have the output equal zero. Thus one can pose a natural conjecture that decidability of language equivalence for unambiguous models always comes as a byproduct of equivalence of the weighted model. Our results show that this is however not always the case as VASSes are a counterexample to this conjecture. In the case of upward-VASSes language equivalence for unambiguous models is decidable. However equivalence for weighted VASSes is undecidable as it would imply decidability of path equivalence (for each word both systems need to accept by the same number of accepting runs) which is undecidable for VASSes [20].

Unambiguity and separability. Our result from Section 4 uses the notion of regular-separability in order to obtain a result for unambiguous VASSes. This technique seems to generalise for some other well-structured transition systems. It is natural to ask whether there is some deeper connection between the notions of separability and unambiguity which can be explored in future research.

References

- 1 Toshiro Araki and Tadao Kasami. Some decision problems related to the reachability problem for Petri nets. *Theor. Comput. Sci.*, 3(1):85–104, 1976.
- 2 Corentin Barloy and Lorenzo Clemente. Bidimensional linear recursive sequences and universality of unambiguous register automata. In *Proceedings of STACS 2021*, volume 187 of *LIPICs*, pages 8:1–8:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

- 3 Mikolaj Bojanczyk, Bartek Klin, and Joshua Moerman. Orbit-finite-dimensional vector spaces and weighted register automata. In *Proceedings of LICS 2021*, pages 1–13. IEEE, 2021.
- 4 Rémi Bonnet. The reachability problem for vector addition system with one zero-test. In *Proceedings of MFCS 2011*, volume 6907 of *Lecture Notes in Computer Science*, pages 145–157. Springer, 2011.
- 5 Michaël Cadilhac, Alain Finkel, and Pierre McKenzie. Unambiguous constrained automata. *Int. J. Found. Comput. Sci.*, 24(7):1099–1116, 2013. doi:10.1142/S0129054113400339.
- 6 Thomas Colcombet. Unambiguity in automata theory. In *Proceedings of DDFS 2015*, pages 3–18, 2015.
- 7 Wojciech Czerwinski, Diego Figueira, and Piotr Hofman. Universality problem for unambiguous VASS. In *Proceedings of CONCUR 2020*, pages 36:1–36:15, 2020.
- 8 Wojciech Czerwinski and Piotr Hofman. Language inclusion for boundedly-ambiguous vector addition systems is decidable. *CoRR*, abs/2202.08033, 2022. arXiv:2202.08033.
- 9 Wojciech Czerwinski, Slawomir Lasota, Roland Meyer, Sebastian Muskalla, K. Narayan Kumar, and Prakash Saivasan. Regular separability of well-structured transition systems. In *Proceedings of CONCUR 2018*, volume 118 of *LIPICs*, pages 35:1–35:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.
- 10 Wojciech Czerwinski, Antoine Mottet, and Karin Quaas. New techniques for universality in unambiguous register automata. In *Proceedings of ICALP 2021*, volume 198 of *LIPICs*, pages 129:1–129:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.
- 11 Wojciech Czerwinski and Lukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. In *Proceedings of FOCS 2021*, pages 1229–1240, 2021.
- 12 Stéphane Demri and Ranko Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009.
- 13 L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *American Journal of Mathematics*, 35((4)):413–422, 1913.
- 14 Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.
- 15 Jean Goubault-Larrecq, Simon Halfon, Prateek Karandikar, K. Narayan Kumar, and Philippe Schnoebelen. *The Ideal Approach to Computing Closed Subsets in Well-Quasi-orderings*, pages 55–105. Springer International Publishing, Cham, 2020.
- 16 Michel Hack. The equality problem for vector addition systems is undecidable. *Theor. Comput. Sci.*, 2(1):77–95, 1976.
- 17 Piotr Hofman, Richard Mayr, and Patrick Totzke. Decidability of weak simulation on one-counter nets. In *Proceedings of LICS 2013*, pages 203–212. IEEE Computer Society, 2013.
- 18 Piotr Hofman and Patrick Totzke. Trace inclusion for one-counter nets revisited. In *Proceedings of RP 2014*, volume 8762 of *Lecture Notes in Computer Science*, pages 151–162. Springer, 2014.
- 19 Petr Jancar. Undecidability of bisimilarity for Petri nets and some related problems. *Theor. Comput. Sci.*, 148(2):281–301, 1995.
- 20 Petr Jancar. Nonprimitive recursive complexity and undecidability for petri net equivalences. *Theor. Comput. Sci.*, 256(1-2):23–30, 2001.
- 21 M. Kabil and M. Pouzet. Une extension d’un théorème de P. Jullien sur les âges de mots. *RAIRO – Theoretical Informatics and Applications – Informatique Théorique et Applications*, 26(5):449–482, 1992.
- 22 Ranko Lazic and Sylvain Schmitz. The ideal view on Rackoff’s coverability technique. *Inf. Comput.*, 277:104582, 2021. doi:10.1016/j.ic.2020.104582.
- 23 Jérôme Leroux. The reachability problem for petri nets is not primitive recursive. In *Proceedings of FOCS 2021*, pages 1241–1252, 2021.
- 24 Jérôme Leroux and Sylvain Schmitz. Reachability in vector addition systems is primitive-recursive in fixed dimension. In *Proceedings of LICS 2019*, pages 1–13. IEEE, 2019.

- 25 Ernst W. Mayr. An algorithm for the general Petri net reachability problem. In *Proceedings of STOC 1981*, pages 238–246, 1981.
- 26 Antoine Mottet and Karin Quaas. The containment problem for unambiguous register automata. In *Proceedings of STACS 2019*, pages 53:1–53:15, 2019.
- 27 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.
- 28 Jean-Eric Pin. Syntactic semigroups. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Volume 1: Word, Language, Grammar*, pages 679–746. Springer, 1997.
- 29 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theor. Comput. Sci.*, 6:223–231, 1978.
- 30 Klaus Reinhardt. Reachability in Petri nets with inhibitor arcs. *Electron. Notes Theor. Comput. Sci.*, 223:239–264, 2008.
- 31 Marcel Paul Schützenberger. On the definition of a family of automata. *Inf. Control.*, 4(2-3):245–270, 1961.
- 32 Wen-Guey Tzeng. On path equivalence of nondeterministic finite automata. *Inf. Process. Lett.*, 58(1):43–46, 1996.

A Missing proofs from Section 2

We recall the statement of Proposition 6.

Proposition 6. Let $U \subseteq \mathbb{N}^d$ be an upward-closed set and $D \subseteq \mathbb{N}^d$ be downward-closed set. Then the size of representation of their complements $\bar{U} = \mathbb{N}^d \setminus U$ and $\bar{D} = \mathbb{N}^d \setminus D$ is at most exponential wrt. the sizes $\|U\|$ and $\|D\|$, respectively and can be computed in exponential time.

Proof of Proposition 6. Here we present only the proof for the complement of the upward-closed set U as the case for downward-closed sets follows the same lines. Let $U = u_1\uparrow \cup u_2\uparrow \cup \dots \cup u_n\uparrow$. Then

$$\begin{aligned} \bar{U} &= \mathbb{N}^d \setminus U = \mathbb{N}^d \setminus (u_1\uparrow \cup u_2\uparrow \cup \dots \cup u_n\uparrow) \\ &= (\mathbb{N}^d \setminus u_1\uparrow) \cap (\mathbb{N}^d \setminus u_2\uparrow) \cap \dots \cap (\mathbb{N}^d \setminus u_n\uparrow). \end{aligned}$$

Thus in order to show that $\|\bar{U}\|$ is at most exponential wrt. $\|U\|$ we need to face two challenges. The first one is to show that representation of $(\mathbb{N}^d \setminus u\uparrow)$ for $u \in \mathbb{N}^d$ is not too big wrt. size of u and the second one is to show that the intersection of sets $(\mathbb{N}^d \setminus u\uparrow)$ does not introduce too big blowup.

Let us first focus on the first challenge. Let $|u|$ be the biggest value that appear in u i.e. $|u| = \max\{u[i] : i \in [1, d]\}$. We claim that if $v \in \mathbb{N}^d \setminus u\uparrow$ and $v[i] > |u|$ for $i \in [1, d]$ then $v + e_i \in \mathbb{N}^d \setminus u\uparrow$. Indeed, if $v \in \mathbb{N}^d \setminus u\uparrow$ then there is $j \in [1, d]$ such that $v[j] < u[j]$. Of course $i \neq j$, so $v + e_i \not\geq u$ and thus $v + e_i \in \mathbb{N}^d \setminus u\uparrow$. But this means that if $\hat{v} \in (\mathbb{N} \cup \{\omega\})^d$ such that $\hat{v}\downarrow \subseteq \mathbb{N}^d \setminus u\uparrow$ and \hat{v} is maximal (namely its entries cannot be increased without violating $\hat{v}\downarrow \subseteq \mathbb{N}^d \setminus u\uparrow$) then $\hat{v} \in ([0, |u|] \cup \{\omega\})^d$. Thus there are only exponentially many possibilities for \hat{v} and the representation of $\mathbb{N}^d \setminus u\uparrow$ is at most exponentially bigger than the representation of u .

Let us face now the second challenge. Let $\hat{v}_1, \hat{v}_2 \in ([1, |u|] \cup \{\omega\})^d$. Observe that $v \in \hat{v}_1\downarrow \cap \hat{v}_2\downarrow$ if and only if $v[i] \leq \hat{v}_1[i]$ and $v[i] \leq \hat{v}_2[i]$ for all $i \in [1, d]$. But this means that if $\hat{v} \in (\mathbb{N} \cup \{\omega\})^d$ and $\hat{v}\downarrow \subseteq \hat{v}_1\downarrow \cap \hat{v}_2\downarrow$ is maximal then $\hat{v} \in ([0, |u|] \cup \{\omega\})^d$. Thus the representation of $\mathbb{N}^d \setminus U$ is also only at most exponentially bigger than the representation of U .

In order to compute the representation of U one can simply check for all $\hat{v} \in ([0, |u|] \cup \{\omega\})^d$ whether $\hat{v}\downarrow \subseteq \mathbb{N}^d \setminus U$. ◀

We recall the statement of Lemma 7.

Lemma 7. The emptiness problem for VASSes with the acceptance condition $F = q_F(U \times D)$ where D is a down-atom and U is an up-atom is in Ackermann.

Proof of Lemma 7. We provide a polynomial reduction of the problem to the emptiness problem in singleton-VASSes which is in Ackermann. Let $V = (Q, T, c_I, q_F(U \times D))$ be a d -VASS with up-atom $U \subseteq \mathbb{N}^{d_1}$ and down-atom $D \subseteq \mathbb{N}^{d_2}$ such that $d_1 + d_2 = d$. Let $U = u \uparrow$ for some $u \in \mathbb{N}^{d_1}$ and let $D = v \downarrow$ for some $v \in (\mathbb{N} \cup \{\omega\})^{d_2}$. Let us assume wlog of generality that $d_2 = d_U + d_B$ such that for $i \in [1, d_U]$ we have $v[i] = \omega$ and for $i \in [d_U + 1, d_2]$ we have $v[i] \in \mathbb{N}$. Let a d -VASS V' be the VASS V slightly modified in the following way. First we add a new state q'_F and a transition $(q_F, 0^d, q'_F)$. Next, for each dimension $i \in [1, d_1]$ we add a loop in state q'_F (transition from q'_F to q'_F) with the effect $-e_i$, namely the one decreasing the dimension i , these are the dimensions corresponding to the up-atom U . Similarly for each dimension $i \in [d_1 + 1, d_1 + d_U]$ we add in q'_F a loop with the effect $-e_i$, these are the unbounded dimensions corresponding to the down-atom D . Finally for each dimension $i \in [d_1 + d_U + 1, d]$ we add in q'_F a loop with the effect e_i (notice that this time we increase the counter values), these are the bounded dimensions corresponding to the down-atom D . Let the initial configuration of V' be c_I (the same as in V) and the set of final configurations F' of V' be the singleton set containing $q'_F(u, (0^{d_U}, v[d_U + 1], \dots, v[d_U + d_B]))$. Clearly V' is a singleton-VASS, so the emptiness problem for V' is in Ackermann. It is easy to see that the emptiness problem in V and in V' are equivalent which finishes the proof. \blacktriangleleft

B Missing proofs from Section 3.1

We recall the statement of Lemma 12.

Lemma 12. For each d -dimensional singleton-VASS V with final configuration being $c_F = q_F(0^d)$ one can construct in polynomial time two deterministic $(d + 1)$ -dimensional upward-VASSes V_1 and V_2 such that

$$L(V_1) = L(V_2) \iff L(V) = \emptyset.$$

Proof of Lemma 12. For a given $V = (Q, T, c_I, c_F)$ we construct $V_1 = (\Sigma = T \cup \{a\}, Q \cup \{q'_F\}, T' \cup \{t_1\}, c'_I, q'_F(0^{d+1} \uparrow))$, and $V_2 = (\Sigma = T \cup \{a\}, Q \cup \{q'_F\}, T' \cup \{t_2\}, c'_I, q'_F(0^{d+1} \uparrow))$. Notice, V_1 and V_2 are pretty similar to each other and also to V . Both V_1 and V_2 have the same states as V plus one additional state q'_F . Notice that the alphabet of labels of V_1 and V_2 is the set of transitions T of V plus one additional letter a . For each transition $t = (p, v, q) \in T$ of V we create a transition $(p, t, v', q) \in T'$ where

- for each $i \in [1, d]$ we have $v'[i] = v[i]$; and
- $v'[d + 1] = v[1] + \dots + v[d]$,

so v' is identical as v on the first d dimensions and on the last $(d + 1)$ -th dimension it keeps the sum of all the others. Notice that transitions in T' are used both in V_1 and in V_2 .

We also add one additional transition t_1 to V_1 and one t_2 to V_2 . To V_1 we add a new a -labelled transition from q_F to q'_F with the effect equal 0^{d+1} for the additional letter a . To V_2 we also add an a -labelled transition between q_F and q'_F , but with an effect equal $(0^d, -1)$. This -1 on the last coordinate is the only difference between V_1 and V_2 . The starting configuration in both V_1 and V_2 is $c'_I = q_I(x_1, x_2, \dots, x_d, \sum_{i=1}^d x_i)$ where $c_I = q_I(x_1, x_2, \dots, x_d)$. The set of accepting configurations is the same in both V_1 and V_2 , namely it is $q'_F(0^{d+1} \uparrow)$. Notice that both V_1 and V_2 are deterministic upward-VASSes, as required in the lemma statement.

Now we aim to show that $L(V_1) = L(V_2)$ if and only if $L(V) = \emptyset$. First observe that $L(V_1) \supseteq L(V_2)$. Clearly if $w \in L(V_2)$ then $w = ua$ for some $u \in T^*$, where T is the set of transitions of V . For any word $ua \in L(V_2)$ we have

$$c'_I \xrightarrow{u} q_F(v) \xrightarrow{a} q'_F(v - e_{d+1})$$

in V_2 . But, then we have also

$$c'_I \xrightarrow{u} q_F(v) \xrightarrow{a} q'_F(v)$$

in V_1 . Thus $ua \in L(V_1)$.

Now we show that, if $L(V) \neq \emptyset$, so $c_I \rightarrow q_F(0^d)$ in V then $L(V_1) \neq L(V_2)$. Let the run ρ of V be such that $c_I \xrightarrow{\rho} q_F(0^d)$ and let $u = \text{trans}(\rho) \in T^*$. Then clearly $c'_I \xrightarrow{u} q_F(0^{d+1}) \xrightarrow{a} q'_F(0^{d+1})$ and $ua \in L(V_1)$. However $ua \notin L(V_2)$ as the last coordinate on the run of V_2 over ua corresponding to ρ would go below zero and this is the only possible run of V_2 over ua due to determinism of V_2 .

It remains to show that if $L(V) = \emptyset$, so $c_I \not\rightarrow q_F(0^d)$ in V , then $L(V_1) \subseteq L(V_2)$. Let $w \in L(V_1)$. Then $w = ua$ for some $u \in T^*$. Let $c'_I \xrightarrow{\rho} c$ in V_1 such that $\text{trans}(\rho) = u$. As $ua \in L(V_1)$ we know that $c = q_F(v)$. However as $c_I \not\rightarrow q_F(0^d)$ in V we know that $v \neq 0^{d+1}$. In particular $v[d+1] > 0$. Therefore $w = ua \in L(V_2)$ as the last transition over a may decrease the $(d+1)$ -th coordinate and reach an accepting configuration. This finishes the proof. \blacktriangleleft

C Missing proofs from Section 3.2

We recall the statement of Lemma 13.

Lemma 13. For a VASS V_1 and a downward-VASS V_2 one can construct in polynomial time an updown-VASS V such that $L(V) = L(V_1) \cap L(V_2)$.

Proof of Lemma 13. We construct V as the standard synchronous product of V_1 and V_2 . The set of accepting configurations in V is also the product of accepting configurations in V_1 and accepting configurations in V_2 , thus due to Proposition 5 a finite union of $q(U \times D)$ for a state q of V , an up-atom U and a down-atom D . \blacktriangleleft

We recall the statement of Theorem 14.

Theorem 14. For a deterministic VASS one can build in exponential time a downward-VASS which recognises the complement of its language.

Proof of Theorem 14. Let $V = (\Sigma, Q, T, c_I, F)$ be a deterministic d -VASS. We aim at constructing a d -dimensional downward-VASS V' such that $L(V') = \overline{L(V)}$. Before constructing V' let us observe that there are three possible scenarios for a word w to be not in $L(V)$. The first scenario (1) is that the only run over w in V finishes in a non-accepting configuration. Another possibility is that there is even no run over w . Namely for some prefix va of w where $v \in \Sigma^*$ and $a \in \Sigma$ we have $c_I \xrightarrow{v} c$ for some configuration c but there is no transition from c over the letter a as either (2) a possible transition over a would decrease some of the counters below zero, (3) there is no such transition possible in V in the state of c .

We are ready to describe VASS $V' = (\Sigma, Q', T', c'_I, F')$. Roughly speaking it consists of $|T| + |\Sigma| + 1$ copies of V . Concretely the set of states Q' is the set of pairs $Q \times (T \cup \Sigma \cup \{-\})$. Let $c_I = q_I(v_I)$. Then let $q'_I \in Q'$ be defined as $q'_I = (q_I, -)$ and we define the initial

configuration of V as $c'_I = q'_I(v_I)$. The set of accepting configurations $F' = F_1 \cup F_2 \cup F_3$ is a union of three sets F_i , each set F_i for $i \in \{1, 2, 3\}$ is responsible for accepting words rejected by VASS V because of the scenario (i) described above. We successively describe which transitions are added to T' and which configurations are added to F' in order to appropriately handle various scenarios.

We first focus on words fulfilling the scenario (1). For states of a form $(q, -)$ the VASS V' is just as V . Namely for each transition $(p, a, v, q) \in T$ we add (p', a, v, q') to T' where $p' = (p, -)$ and $q' = (q, -)$. We also add to F' the following set $F_1 = \{(q, -)(v) \mid q(v) \notin F\}$. It is easy to see that words that fulfil scenario (1) above are accepted in V' by the use of the set F_1 . The size of the description of F_1 is at most exponential wrt. the size of the description of F by Proposition 6.

Now we describe the second part of V' which is responsible for words rejected by V because of the scenario (2). The idea is that we guess when the run over w is finished. For each transition $t = (p, a, v, q) \in T$ we add $(p', a, 0^d, q')$ to T' where $p' = (p, -)$ and $q' = (q, t)$. The idea is that the run reaches the configuration in which the transition t cannot be fired. Now we have to check that our guess is correct. In the state (q, t) for $t \in T$ no transition changes the configuration. Namely for each $q' = (q, t) \in Q \times T$ and each $a \in \Sigma$ we add to T' transition $(q', a, 0^d, q')$. We add now to F' the set $F_2 = \{(q, t)(v) \mid v + \text{eff}(t) \notin \mathbb{N}^d\}$. Notice that F_2 can be easily represented as a polynomial union of down-atoms. It is easy to see that indeed V' accepts by F_2 exactly words w such that there is a run of V over some prefix v of w but reading the next letter would decrease one of the counters below zero.

The last part of V' is responsible for the words w rejected by V because of the scenario (3), namely w has a prefix va such that there is a run over $v \in \Sigma^*$ in V but then in the state of the reached configuration there is no transition over the letter $a \in \Sigma$. To accept such words for each state $p \in Q$ and letter $a \in \Sigma$ such that there is no transition of a form $(p, a, v, q) \in T$ for any $v \in \mathbb{N}^d$ and $q \in Q$ we add to T' transition $((p, -), a, 0^d, (p, a))$. In each state $p' = (p, a) \in Q \times \Sigma$ we have a transition $(p', b, 0^d, p')$ for each $b \in \Sigma$. We also add to F' the set $F_3 = \{(p, a)(v) \mid v \in \mathbb{N}^d \text{ and there is no } (p, a, u, q) \in T \text{ for } u \in \mathbb{N}^d \text{ and } q \in Q\}$. Size of F_3 is polynomial wrt. T .

Summarising V' with the accepting downward-closed set $F = F_1 \cup F_2 \cup F_3$ indeed satisfies $L(V') = \overline{L(V)}$, which finishes the construction and the proof. \blacktriangleleft

We recall the statement of Lemma 16.

Lemma 16. For each HVASS one can compute in exponential time a language equivalent ε -VASS.

Proof of Lemma 16. Let $V = (\Sigma, Q, T, q_I(v_I), F, H)$ be a d -HVASS with the set of holes H . We aim at constructing a d -VASS $V' = (\Sigma, Q', T', c'_I, F')$ such that $L(V) = L(V')$. By Proposition 6 we can compute in exponential time an upward-closed set of configurations $U = (Q \times \mathbb{N}^d) \setminus H$. In order to translate V into a d -VASS V' intuitively we need to check that each configuration on the run is not in the set H . In order to do this we use the representation of U as a finite union $U = \bigcup_{i \in [1, k]} q_i(u_i \uparrow)$ for $q_i \in Q$ and $u_i \in \mathbb{N}^d$. Now for each configuration c on the run of V the simulating VASS V' needs to check that c belongs to $q_i(u_i \uparrow)$ for some $i \in [1, k]$. That is why in V' after every step simulating a transition of V we go into a testing gadget and after performing the test we are ready to simulate the next step. For that purpose we define $Q' = (Q \times \{0, 1\}) \cup \{r_1, \dots, r_k\}$. The states in $Q \times \{0\}$ are the ones before the test and the states in $Q \times \{1\}$ are the ones after the test. States r_1, \dots, r_k are used to perform the test. The initial configuration c'_I is defined as $(q_I, 0)(v_I)$

and set of final configurations is defined as $F' = \{(q, 1)(v) \mid q(v) \in F\}$. For each transition (p, a, v, q) in T we add a corresponding transition $((p, 1), a, v, (q, 0))$ to T' . In each reachable configuration $(q, 0)(v)$ the VASS V' nondeterministically guesses for which $i \in [1, k]$ holds $q_i(u_i) \preceq q(v)$ (which guarantees that indeed $q(v) \in U$). In order to implement it for each $q \in Q$ and each $i \in [1, k]$ such that $q = \text{state}(r_i)$ we add two transitions to T' : the one from $(q, 0)$ to r_i subtracting u_i , namely $((q, 0), \varepsilon, -u_i, r_i)$ and the one coming back and restoring the counter values, namely $(r_i, \varepsilon, u_i, (q, 1))$. It is easy to see that $(q, 0)(v) \xrightarrow{\varepsilon} (q, 1)(v)$ if and only if $q(v) \in U$, which finishes the proof. ◀

We recall the statement of Theorem 17.

Theorem 17. For a deterministic HVASS one can compute in exponential time a downward- ε -VASS which recognises the complement of its language.

Proof of Theorem 17. The proof of Theorem 17 is very similar to the proof of Theorem 14 so we only sketch the key differences. Let V be a deterministic HVASS and let $H \subseteq Q \times \mathbb{N}^d$ be the set of its holes. Let $U = (Q \times \mathbb{N}^d) \setminus H$, by Proposition 6 we know that $U = \bigcup_{i \in [1, k]} q_i(u_i \uparrow)$ for some states $q_i \in Q$ and vectors $u_i \in \mathbb{N}^d$, and additionally $\|U\|$ is at most exponential wrt. the size $\|H\|$.

The construction of V' recognising the complement of $L(V)$ is almost the same as in the proof of Theorem 14, we need to introduce only small changes. The biggest changes are in the part of V' recognising words rejected by V because of scenario (1). We need to check that after each transition the current configuration is in U (so it is not in any hole from H). We perform it here in the same way as in the proof of Lemma 16. Namely we guess to which $q_i(u_i \uparrow)$ the current configuration belongs and check it by simple VASS modifications (for details look to the proof of Lemma 16). The size of this part of V' can have a blowup of at most size of U times, namely the size can be multiplied by some number, which is at most exponential wrt. the size $\|H\|$.

In the part recognising words rejected by V because of scenario (2), we need only to adjust the accepting set F_2 . Indeed, we need to accept now if we are in a configuration $(p, t)(v) \in Q \times T$ such that $v + t \notin \mathbb{N}^d$ or $v + t \in H$ (in contrast to only $v + t \notin \mathbb{N}^d$ in the proof of Theorem 14). This change does not introduce any new superlinear blowup.

Finally the part recognising words rejected by V because of scenario (3) does not need adjusting at all. It is not hard to see that the presented construction indeed accepts the complement of $L(V)$ as before. The constructed downward-VASS V' is of at most exponential size wrt. the size V as explained above, which finishes the proof. ◀

We recall the statement of Theorem 18.

Theorem 18. The inclusion problem of an HVASS language in a deterministic HVASS language is in Ackermann.

Proof of Theorem 18. Let $V_1 = (\Sigma, Q_1, T_1, c_1^1, F_1, H_1)$ be a d_1 -HVASS with holes $H_1 \subseteq Q_1 \times \mathbb{N}^{d_1}$ and let $V_2 = (\Sigma, Q_2, T_2, c_2^2, F_2, H_2)$ be a deterministic d_2 -HVASS with holes $H_2 \subseteq Q_2 \times \mathbb{N}^{d_2}$. By Lemma 16 an ε -VASS V_1' equivalent to V_1 can be computed in exponential time. By Theorem 17 a downward- ε -VASS V_2' can be computed in exponential time such that $L(V_2') = \Sigma^* \setminus L(V_2)$. It is enough to check now whether $L(V_1') \cap L(V_2') = \emptyset$. By Lemma 13 (extended to ε -VASSes) one can compute an updown- ε -VASS V such that $L(V) = L(V_1') \cap L(V_2')$. Finally by Corollary 8 (also extended to ε -VASSes) the emptiness problem for updown- ε -VASSes is in Ackermann which finishes the proof. ◀

We recall the statement of Theorem 19.

Theorem 19. For a k -deterministic d -VASS one can build in exponential time a $(k \cdot d)$ -dimensional downward-VASS which recognises the complement of its language.

Proof of Theorem 19. Before starting the proof let us remark that it would seem natural to first build a $(k \cdot d)$ -VASS equivalent to the input k -deterministic d -VASS and then apply construction from the proof of Theorem 14 to recognise the its complement. However, it is not clear how to construct a $(k \cdot d)$ -VASS equivalent to k -deterministic d -VASS, thus we compute directly a VASS recognising the complement of the input VASS language.

Let $V = (\Sigma, Q, T, c_I, F)$ be a k -deterministic d -VASS. We aim to construct $(k \cdot d)$ -dimensional downward-VASS $V' = (\Sigma, Q', T', c'_I, F')$ such that $L(V') = \Sigma^* \setminus L(V)$. Also in this proof we strongly rely on the ideas introduced in the proof of Theorem 14. The idea of the construction is that V' simulates k copies of V which take care of different maximal runs of V . Then the accepting condition F' of V' verifies whether in all the copies there is a reason that the simulated maximal runs do not accept.

Recall that for a run there are three scenarios in which it is not accepted: (1) it reaches the end of the word, but the reached configuration is not accepted, (2) at some moment it tries to decrease some counter below zero, and (3) at some moment there is no transition available over the input letter. In the proof of Theorem 14 it was shown how a VASS can handle all the three reasons. In short words: in case (1) it simulates the run till the end of the word and then checks that the reached configuration is not accepting and in cases (2) and (3) it guesses the moment in which there is no valid transition available and keeps this configuration untouched till the end of the run when it checks by the accepting condition that the guess was correct. We only sketch how the downward-VASS V' works without stating explicitly its states and transitions. It starts in the configuration c'_I which consists of k copies of c_I . Then it simulates the run in all the copies in the same way till the first moment when there is a choice of transition. Then we enforce that at least one copy follows each choice, but we allow for more than one copy to follow the same choice. In the state of V' we keep the information which copies are following the same maximal run and which have already split. Each copy is exactly as in the proof of Theorem 14, it realises one of the scenarios (1), (2) or (3). As we know that V is k -deterministic we are sure that all the possible runs of V can be simulated by V' under the condition the V' correctly guesses which copies should simulate which runs. If guesses of V' are wrong and at some point it cannot send to each branch a copy then the run of V' rejects. At the end of the run over the input word w VASS V' checks using the accepting condition F' that indeed all the copies have simulated all the possible maximal runs and that all of them reject. It is easy to see that F' is a downward-closed set, as roughly speaking it is a product of k downward-closed accepting conditions, which finishes the proof. ◀

D Missing proofs from Section 5

The proofs from this section are available only in the arxiv version of this paper because of the space limitation. Please check <https://arxiv.org/pdf/2202.08033.pdf>.

Complexity of Coverability in Depth-Bounded Processes

A. R. Balasubramanian   

Technische Universität München, Germany

Abstract

We consider the class of depth-bounded processes in π -calculus. These processes are the most expressive fragment of π -calculus, for which verification problems are known to be decidable. The decidability of the coverability problem for this class has been achieved by means of well-quasi orders. (Meyer, IFIP TCS 2008; Wies, Zufferey and Henzinger, FoSSaCS 2010). However, the precise complexity of this problem has not been known so far, with only a known EXPSPACE-lower bound.

In this paper, we prove that coverability for depth-bounded processes is \mathbf{F}_{e_0} -complete, where \mathbf{F}_{e_0} is a class in the fast-growing hierarchy of complexity classes. This solves an open problem mentioned by Haase, Schmitz, and Schnoebelen (LMCS, Vol 10, Issue 4) and also addresses a question raised by Wies, Zufferey and Henzinger (FoSSaCS 2010).

2012 ACM Subject Classification Theory of computation \rightarrow Problems, reductions and completeness; Theory of computation \rightarrow Distributed computing models

Keywords and phrases π -calculus, Depth-bounded processes, Fast-growing complexity classes

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.17

Funding A. R. Balasubramanian: Supported by 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 I am grateful to the reviewers and Prof. Javier Esparza for their useful comments and suggestions.

1 Introduction

The π -calculus [21, 22] is a well-known formalism for describing concurrent message-passing systems admitting unbounded process creation and mobility of agents. Intuitively speaking, a configuration of such a system is a graph in which each vertex is a process labelled by its current state and there is an edge between two processes if they share a channel using which they can pass messages. The flexibility of π -calculus lies in the fact that processes can transmit the names of channels using channels themselves, allowing reconfiguration of channels using process definitions itself. Due to its immense expressive power, all interesting verification problems quickly become undecidable for π -calculus processes.

Consequently, research on π -calculus has been focused on finding fragments for which certain problems are decidable. The most expressive fragment of π -calculus for which some verification problems still remain decidable is the class of depth-bounded processes [20]. Intuitively, depth-bounded processes are those in which the length of simple paths in the set of reachable configurations is bounded by a constant. It is known that depth-bounded processes can be viewed as well-structured transition systems (WSTS) [20]. This implies that the coverability problem for such systems is decidable [20, 27]. Intuitively, coverability consists of deciding if a given system can reach a configuration where some process is in an error state.

However, despite the positive decidability results known regarding this problem, the exact complexity of this problem has remained open so far. To the best of our knowledge, only an EXPSPACE-hardness result is known for this problem [27]. In this paper, we



© A. R. Balasubramanian;

licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 17; pp. 17:1–17:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

provide complexity-theoretic completeness results for this problem. More specifically, we prove that the coverability problem for depth-bounded processes is \mathbf{F}_{ϵ_0} -complete, where \mathbf{F}_{ϵ_0} is a complexity class in the *fast-growing hierarchy* of complexity classes [24]. This is a hierarchy of complexity classes which allows for a finer classification of problems that do not admit any elementary-time algorithms, i.e., problems which do not have algorithms whose running times can be upper bounded by a fixed tower of exponentials in the input size. In particular, our result proves that the coverability problem for depth-bounded processes is not primitive-recursive and indeed is harder than even problems complete for the Ackermann complexity class.

The complexity-theoretic classification of problems which are non-elementary has attracted a lot of attention in the recent years, with various techniques developed for proving both lower and upper bounds [13, 6, 25, 24, 1, 23, 8, 19, 7, 18]. While these results are obviously negative from a tractability perspective, understanding the precise complexity of a problem may help us to solve it in practice by reducing it to other well-studied problems for which tools and heuristics have been developed, like the satisfiability problem for weak S1S or the Petri net reachability problem [3, 12, 15, 4, 5, 16, 10]. The fast-growing hierarchy is of great assistance in this task. Adding new complete problems for classes in this hierarchy can help us prove hardness results for other problems in the future, without having to resort to coming up with reductions from scratch, i.e., from Turing machines or counter machines.

Our result significantly improves upon the existing lower bound of EXPSPACE-hardness, which is inherited from the coverability problem for Petri nets. Further, it settles a conjecture raised by Hasse, Schmitz and Schnoebelen (Section 8.3 of [17]) and also addresses a question raised by Wies, Zufferey and Henzinger (Section 5 of [27]).¹ To prove the lower bound, we introduce a new model of computation called *nested counter systems with levels*, which (in a manner) simplifies the already existing model of *nested counter systems* [8], while preserving the hardness of that model.

The techniques used in this paper are similar to the ones presented in [2], in order to prove \mathbf{F}_{ϵ_0} -completeness for parameterized coverability of *bounded-depth broadcast networks*. While some of the ideas between these two papers are similar, there are some differences between the models considered in these two papers. First, as the name suggests, broadcast networks allow for a process to *broadcast* to its set of neighbors, whereas processes in π -calculus interact in a manner akin to *rendez-vous* communication. One might expect that there is a drop in complexity when the communication mechanism goes from broadcast to rendez-vous. For instance, as mentioned in [11], coverability for networks with (unrestricted) broadcast communication is Ackermann-complete, while the same problem for rendez-vous networks is (only) EXPSPACE-complete. Our result suggests that this drop in complexity need not always be the case. Further, in broadcast networks, there is no process creation nor dynamic reconfiguration of channels, whereas π -calculus has both. Finally, for the lower bound construction in this paper, we also need to prove depth-boundedness of any reachable configuration in the process constructed for the reduction, whereas no such property needs to be proven for the lower bound construction for broadcast networks. We also believe that the newly introduced model of nested counter systems with levels (whose hardness we prove by using ideas from [2]), makes the proof of the lower bound for π -calculus cleaner when compared with giving a direct reduction from nested counter systems as was done in [2].

¹ The version of the problem that the authors of [27] consider does not assume that a bound on the depth of the process is given as part of the input, whereas in our setting we take this to be the case, in order to prove the upper bound. However, our lower bound result does not require this assumption.

2 Preliminaries

We first present the syntax and the semantics of the version of π -calculus that we will use. The definitions here are taken from the ones given in [27].

2.1 The π -calculus

We assume that there is a countable collection of *names* (denoted by x, y, \dots) and a countable collection of *process identifiers* (denoted by A, B, \dots). Each name and identifier has an associated *arity* in \mathbb{N} . We use boldface letters like \mathbf{x}, \mathbf{y} to denote (possibly empty) vectors over names and denote substitution of names by $[\mathbf{x}/\mathbf{y}]$, i.e., if $\mathbf{x} = x_1, \dots, x_n$ and $\mathbf{y} = y_1, \dots, y_n$, then $[\mathbf{x}/\mathbf{y}]$ denotes a mapping in which each y_i is mapped to x_i and every other name is mapped to itself.

A *process term* (or simply a term) P is either the unit process 0 , or a parameterized process identifier $A(\mathbf{x})$, or any term obtained by the standard operations of parallel composition $P_1 \mid P_2$, external choice $\pi_1 \cdot P_1 + \pi_2 \cdot P_2$ and name restriction $(\nu x)P_1$. Here P_1 and P_2 are themselves terms and π_1 and π_2 are prefixes which can either be an *input prefix* $x(\mathbf{y})$ or an *output prefix* $\bar{x}(\mathbf{y})$ or the empty string. All parameter vectors occurring in a parameterized process identifier or a prefix must respect the arity of the names and identifiers. A *thread* is a term of the form $A(\mathbf{x})$. We use Π and Σ to denote (indexed) parallel composition and external choice. We further use $(\nu \mathbf{x})$ to denote $(\nu x_1)(\nu x_2) \dots (\nu x_n)$ where $\mathbf{x} = x_1, \dots, x_n$. The application of a substitution of names σ to a term P , denoted by $\sigma(P)$, is defined in the usual way.

An occurrence of a name x in a term P is called *free* if it is not below a (νx) or an input prefix $y(x)$. We let $\text{fn}(P)$ denote the set of free names of P . A *bound name* of P is a name of P which is not free. We say that P is *closed* if $\text{fn}(P) = \emptyset$. We use the usual *structural congruence relation* $P \equiv Q$ on process terms, i.e., $P \equiv Q$ if P is syntactically equal to Q upto renaming and reordering of bound names, associativity and commutativity of parallel composition and external choice, elimination of units $((P \mid 0) \equiv P, (\nu x)0 \equiv 0)$ and *scope extrusion* $((\nu x)(P \mid Q) \equiv (\nu x)P \mid Q$ if $x \notin \text{fn}(Q)$).

A *configuration* is a closed term of the form $(\nu \mathbf{x})(\Pi_{i \in I} A_i(\mathbf{x}_i))$. A *process* \mathcal{P} is a pair (I, \mathcal{E}) where I is an *initial configuration* and \mathcal{E} is a set of *parametric equations* of the form $A(\mathbf{x}) = P$ where A is an identifier and P is a term such that 1) every identifier in \mathcal{P} is defined by exactly one equation in \mathcal{E} and 2) if $A(\mathbf{x}) = P$ is an equation, then $\text{fn}(P) \subseteq \{\mathbf{x}\}$. We assume that all the equations are given in the following form:

$$A(\mathbf{x}) = \sum_{i \in I} \pi_i \cdot (\nu \mathbf{x}_i) \left(\prod_{j \in J_i} A_j(\mathbf{x}_j) \right)$$

Operational semantics

Let $\mathcal{P} = (I, \mathcal{E})$ be a process. We define a transition relation on the set of configurations using \mathcal{E} as follows. Let P and Q be configurations. Then $P \rightarrow Q$ iff the following conditions are satisfied:

- $P \equiv (\nu \mathbf{u})(A(\mathbf{v}) \mid B(\mathbf{w}) \mid P')$,
- The defining equation of A in \mathcal{E} is of the form $A(\mathbf{x}) = x(\mathbf{x}') \cdot (\nu \mathbf{x}'')(M) + M'$,
- The defining equation of B in \mathcal{E} is of the form $B(\mathbf{y}) = \bar{y}(\mathbf{y}') \cdot (\nu \mathbf{y}'')(N) + N'$,
- $\sigma = [\mathbf{v}/\mathbf{x}, \mathbf{w}/\mathbf{y}, \mathbf{w}'/\mathbf{x}', \mathbf{z}_A/\mathbf{x}'', \mathbf{z}_B/\mathbf{y}'']$ where $\mathbf{z}_A, \mathbf{z}_B$ are fresh names and \mathbf{w}' is the set of names assigned to \mathbf{y}' under the mapping $[\mathbf{w}/\mathbf{y}]$.
- $\sigma(x) = \sigma(y)$ and
- $Q \equiv (\nu \mathbf{u}, \mathbf{z}_A, \mathbf{z}_B)(\sigma(M) \mid \sigma(N) \mid P')$

17:4 Complexity of Coverability in Depth-Bounded Processes

We denote such a step by $P \xrightarrow{A(\mathbf{v}),\sigma(x),B(\mathbf{w})} Q$ or simply by $P \rightarrow Q$. We can then define the reachability relation $\xrightarrow{*}$ as the reflexive and transitive closure of \rightarrow . We say that a configuration P is reachable in \mathcal{P} iff $I \xrightarrow{*} P$. We further say that P is coverable if $P \equiv (\nu \mathbf{x})P'$ and there exists $Q \equiv (\nu \mathbf{x})(P' \mid R)$ such that $I \xrightarrow{*} Q$. The coverability problem is to decide if a given configuration P is coverable in a given process \mathcal{P} .

Depth-bounded processes

We now define the class of depth-bounded processes. The nesting of restrictions $nest$ of a term P is defined inductively as follows: $nest(0) = nest(A(\mathbf{x})) = nest(\pi_1 \cdot P_1 + \pi_2 \cdot P_2) = 0$, $nest((\nu x)P) = 1 + nest(P)$ and $nest(P_1 \mid P_2) = \max\{nest(P_1), nest(P_2)\}$. The *depth* of a term P is the minimal nesting of restrictions of terms in the congruence class of P :

$$depth(P) := \min\{nest(Q) : Q \equiv P\}$$

► **Definition 1.** A set of configurations \mathcal{C} is called *k-depth-bounded* if the depth of all configurations in \mathcal{C} is at most k . \mathcal{C} is called *depth-bounded* if there is some k such that it is *k-depth-bounded*. A process P is called *(k-)depth-bounded* if its set of reachable configurations is *(k-)depth-bounded*.

► **Example 2.** The following example intuitively demonstrates a system in which there is one “level 0” thread which can spawn “level 1” threads by using a “New1” thread. Then, each level 1 thread can itself spawn “level 2” threads by using their own “New2” threads.

$$Level0(x) = \bar{x}().Level0(x) \quad New1(x) = x().((\nu y)(New1(x) \mid Level1(x, y) \mid New2(y)))$$

$$Level1(x, y) = \bar{y}().Level1(x, y) \quad New2(y) = y().((\nu z)(New2(y) \mid Level2(y, z) \mid New3(z)))$$

$$Level2(y, z) = \bar{z}().Level2(y, z) \quad New3(z) = z().New3(z)$$

Suppose we set $I = (\nu x)(Level0(x) \mid New1(x))$. Then the following is a valid run:

$$\begin{aligned} I &\rightarrow (\nu x)(Level0(x) \mid New1(x) \mid (\nu y)(Level1(x, y) \mid New2(y))) \\ &\rightarrow (\nu x)(Level0(x) \mid New1(x) \mid (\nu y)(Level1(x, y) \mid New2(y) \mid (\nu z)(Level2(y, z) \mid New3(z)))) \end{aligned}$$

We note that the depth of the last configuration in this run is 3. Indeed, we can show that the depth of any reachable configuration from I is at most 3. Later on, we will see that some of the ideas behind this example are relevant to our lower bound construction.

Our main theorem of the paper is that,

► **Theorem 3.** *The coverability problem for depth-bounded processes is \mathbf{F}_{ϵ_0} -complete.*

Here, we assume that the input consists of a process \mathcal{P} and a number k such that \mathcal{P} is *k-depth-bounded*. Further, \mathbf{F}_{ϵ_0} is a complexity class in the *fast-growing hierarchy* of complexity classes [24]. Due to lack of space, we do not define it here. The lower bound behind this theorem is accomplished by giving a log-space reduction from a \mathbf{F}_{ϵ_0} -hard problem. The upper bound is obtained by using results on the length of *controlled bad sequences* over a suitable well-quasi ordering.

We first explain the proof of the lower bound. To do this, we first introduce a model called *nested counter systems with levels* (NCSL) and show that the coverability problem for this model is \mathbf{F}_{ϵ_0} -hard. We then give a reduction from this problem to the coverability problem for depth-bounded processes, thereby proving the lower bound of Theorem 3.

3 Nested counter systems with levels (NCSL)

We now introduce a new model of computation called nested counter systems with levels (NCSL) and prove \mathbf{F}_{ϵ_0} -hardness of coverability for this model. NCSL are closely related to the so-called nested counter systems (NCS) [8]. Indeed, in Section 4, we will recall NCS and prove the hardness result for NCSL by giving a reduction from the coverability problem for NCS.

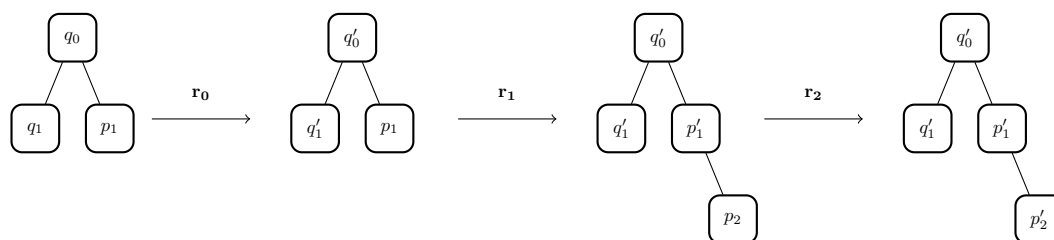
Before describing NCSL in a formal manner, we give some intuition. A k -NCSL is a generalisation of a usual counter system with *higher-order counters*. Intuitively, a 1-dimensional counter is a usual counter which can add or subtract 1. A 2-dimensional counter can add or subtract 1-dimensional counters, a 3-dimensional counter can add or subtract 2-dimensional counters and so on. A k -NCSL can produce up to k -dimensional counters and then manipulate these counters using “local” rules, i.e., rules which update at most 2 counters at a time. Later on, we will consider the NCS model [8], which allows to update multiple counters in a single step.

Formally, a k -nested counter system with levels (k -NCSL) is a tuple $\mathcal{N} = (Q, \delta_0, \dots, \delta_{k-1}, \delta_k)$ where Q is a finite set of *states* and each δ_l is a set of *level- l rules* such that $\delta_l \subseteq \bigcup_{1 \leq i \leq j \leq 2} (Q^i \times Q^j)$. We further enforce that if $l = k$ then $\delta_l \subseteq Q \times Q$. The set $\mathcal{C}_{\mathcal{N}}$ of *configurations* of \mathcal{N} is defined to be the set of all labelled rooted trees of height at most k , with labels from the set Q .

The operational semantics of \mathcal{N} is defined in terms of the following transition relation $\rightarrow \subseteq \mathcal{C}_{\mathcal{N}} \times \mathcal{C}_{\mathcal{N}}$ on configurations: Let $r := ((q_0, \dots, q_i), (q'_0, \dots, q'_j)) \in \delta_l$ be a level- l rule with $l \leq k$ and $0 \leq i \leq j \leq 1$. We say that a configuration C can move to the configuration C' using the rule r (denoted by $C \xrightarrow{r} C'$) if *there is a node v_0 at depth l in C with label q_0 and the following holds*.

- **Creation.** Suppose $r = ((q_0), (q'_0, q'_1))$. Then C' is obtained from C by changing the label of v_0 to q'_0 , creating a new vertex v_1 with label q'_1 and adding it as child to v_0 .
- **1-Preservation.** Suppose $r = ((q_0), (q'_0))$. Then C' is obtained from C by changing the label of v_0 to q'_0 .
- **2-Preservation.** Suppose $r = ((q_0, q_1), (q'_0, q'_1))$. Then there is a child v_1 of v_0 in C with label q_1 and C' is obtained from C by changing the labels of v_0 and v_1 to q'_0 and q'_1 respectively.

► **Example 4.** Let us consider the 2-NCSL \mathcal{N} given by the states $Q = \{p_i, p'_i, q_i, q'_i : 0 \leq i \leq 4\}$ and consisting of the rules $r_0 \in \delta_0, r_1 \in \delta_1, r_2 \in \delta_2$ where $r_0 = ((q_0, q_1), (q'_0, q'_1)), r_1 = ((p_1), (p'_1, p_2)), r_2 = ((p_2), (p'_2))$. In Figure 1, we illustrate the application of these rules to a configuration of \mathcal{N} .



■ **Figure 1** Application of the rules r_0, r_1 and r_2 to a configuration of \mathcal{N} , which is described in Example 4.

We say that $C \rightarrow C'$ if $C \xrightarrow{r} C'$ for some rule r . We can then define the reachability relation $\xrightarrow{*}$ in a standard manner. Given two states $q_{in}, q_f \in Q$, we say that q_{in} can cover q_f if the (unique) configuration consisting of the single root vertex labelled with q_{in} (also called the initial configuration of \mathcal{N}) can reach *some* configuration where the root is labelled by q_f . The coverability problem for an NCSL is then the following: Given an NCSL \mathcal{N} and two states q_{in}, q_f , can q_{in} cover q_f ? We prove that

► **Theorem 5.** *The coverability problem for NCSL is \mathbf{F}_{ϵ_0} -hard, even when restricted to NCSL which only have creation and 2-preservation rules.*

The proof of Theorem 5 is deferred to Section 4. We shall assume this theorem and first prove the main result of this paper (Theorem 3), i.e., that coverability for depth-bounded π -calculus processes is \mathbf{F}_{ϵ_0} -hard.

3.1 Hardness of coverability for depth-bounded π -calculus processes

Throughout this subsection, we let $\mathcal{N} = (Q, \delta_0, \dots, \delta_{k-1}, \delta_k)$ be a fixed k -NCSL which only has creation and 2-preservation rules. Note that since there are no 1-preservation rules, by definition of a k -NCSL, δ_k is empty and so we will ignore δ_k everywhere in this section. Let q_{in} and q_f be two fixed states of \mathcal{N} . We will now construct a depth-bounded process \mathcal{P} and a configuration C of \mathcal{P} such that C can be covered in \mathcal{P} iff q_f can be covered from q_{in} in \mathcal{N} .

Process identifiers, names and the initial configuration

To construct \mathcal{P} , we have to define an initial configuration and a set of parametric equations. We begin by specifying the set of names and the process identifiers that we shall use in the equations. Based on these names and identifiers, we define the initial configuration and also introduce an injective mapping \mathbb{B} from the set of configurations of \mathcal{N} to the set of configurations of \mathcal{P} . This map will be useful to prove the correctness of our reduction.

Process identifiers and names. For each $1 \leq i \leq k$, we will have a process identifier $start[i]$. For each $0 \leq i \leq k$ and each state q of \mathcal{N} , we will have an identifier $q[i]$. Notice that each process identifier is of the form $a[b]$ where $a \in Q \cup \{start\}$ and $0 \leq b \leq k$. The first part “ a ” will be called the *base* of the identifier and the second part “ b ” will be called the *grade* of the identifier. The arities of the identifiers are as follows: The arity of each $start[i]$ will be $|\delta_{i-1}|$. For every state q of \mathcal{N} , the arity of $q[0]$ will be $|\delta_0|$, the arity of $q[k]$ will be $|\delta_{k-1}|$ and the arity of every other $q[i]$ will be $|\delta_{i-1}| + |\delta_i|$.

The set of names that we will be using in the equations will be the set of rules of \mathcal{N} , i.e., $\delta_0 \cup \delta_1 \cup \dots \cup \delta_{k-1}$. For each δ_i , we let \mathbf{n}_i denote some fixed vector comprising all the names from δ_i . We also assume that there is another countably infinite set of names needed to describe the configurations of \mathcal{P} . We note that this latter set is not part of the input.

A mapping. We now introduce an injective map from the set of configurations of \mathcal{N} to the set of configurations of \mathcal{P} . Let C be a configuration of the NCSL \mathcal{N} . To C , we assign a unique configuration of \mathcal{P} (denoted by $\mathbb{B}(C)$) as follows: Let the set of vertices of C be V and let the set of internal vertices of C (the root and the other non-leaf vertices) be IV . $\mathbb{B}(C)$ is then defined as the configuration

$$(\nu \mathbf{z}) (\prod_{v \in V} A_v(\mathbf{x}_v, \mathbf{y}_v) \mid \prod_{v \in IV} B_v(\mathbf{y}_v))$$

where $\{\mathbf{z}\} = \cup_{v \in V} \{\mathbf{x}_v, \mathbf{y}_v\}$ and for each v ,

- $\{\mathbf{x}_v\} \cap \{\mathbf{y}_v\} = \emptyset$,
- If the label of v in C is q and v is at depth l , then $A_v = q[l]$ and $B_v = \text{start}[l + 1]$,
- If v is the root, then \mathbf{x}_v is the empty vector. If v is a leaf, then \mathbf{y}_v is the empty vector. Otherwise, if v is at depth l , then \mathbf{x}_v is of size $|\delta_{l-1}|$ and \mathbf{y}_v is of size $|\delta_l|$.
- For any v' , if v' is a child of v , then $\mathbf{x}_{v'} = \mathbf{y}_v$ and $\{\mathbf{y}_{v'}\} \cap \{\mathbf{x}_v\} = \emptyset$; if v' is a sibling of v , then $\mathbf{x}_{v'} = \mathbf{x}_v$ and $\{\mathbf{y}_{v'}\} \cap \{\mathbf{y}_v\} = \emptyset$; otherwise, $\{\mathbf{x}_v, \mathbf{y}_v\} \cap \{\mathbf{x}_{v'}, \mathbf{y}_{v'}\} = \emptyset$.

To give an intuition behind this mapping, let us look at $\mathbb{B}(C)$ from the perspective of graphs. We construct a graph where there is a vertex for each $A_v(\mathbf{x}_v, \mathbf{y}_v)$ and each $B_v(\mathbf{y}_v)$ and we connect two such vertices by an edge if they share at least one free name and the corresponding identifiers have different grades. By the requirements given above, this would imply that the graph that we get is a tree which has a “copy” of C as a subgraph, along with a new leaf vertex added to every internal vertex of C . Ignoring the new leaf vertices for now, this means that $\mathbb{B}(C)$ can be thought of as a “representation” of C in the process \mathcal{P} . The parametric equations that we shall construct will make sure that if $\mathbb{B}(C)$ can move to a new configuration P , then P will be a representation of C' for some C' such that $C \rightarrow C'$ in the NCSL \mathcal{N} .

We now have the following lemma which proves depth-boundedness of any configuration of the form $\mathbb{B}(C)$. The intuition behind this lemma is that the “graph” of $\mathbb{B}(C)$ contains a copy of C as a subgraph along with some other additional leaf vertices. Hence, since the depth of C is bounded by k , we can expect that the depth of $\mathbb{B}(C)$ is also bounded.

► **Lemma 6** (Depth-boundedness). *For any configuration C , the depth of $\mathbb{B}(C)$ is at most $\sum_{l=0}^{k-1} |\delta_l|$.*

Proof. Let V and IV be the set of vertices and internal vertices of C respectively. For any vertex \mathbf{n} , let $C_{\mathbf{n}}$ be the (labelled) subtree of C rooted at \mathbf{n} and let $V_{\mathbf{n}}$ and $IV_{\mathbf{n}}$ be the set of vertices and internal vertices of $C_{\mathbf{n}}$ respectively.

We know that $\mathbb{B}(C)$ is of the form $(\nu \mathbf{z}) (\prod_{v \in V} A_v(\mathbf{x}_v, \mathbf{y}_v) \mid \prod_{v \in IV} B_v(\mathbf{y}_v))$. Let $\mathbb{B}(C_{\mathbf{n}})$ be the sub-process term of $\mathbb{B}(C)$ given by $\prod_{v \in V_{\mathbf{n}}} A_v(\mathbf{x}_v, \mathbf{y}_v) \mid \prod_{v \in IV_{\mathbf{n}}} B_v(\mathbf{y}_v)$ and let $\{\mathbf{z}_{\mathbf{n}}\} = \cup_{v \in V_{\mathbf{n}}} \{\mathbf{x}_v, \mathbf{y}_v\}$.

By induction on the height h of the vertex \mathbf{n} in the tree C , we will now show that the depth of $(\nu \mathbf{z}_{\mathbf{n}}) \mathbb{B}(C_{\mathbf{n}})$ is at most $\sum_{l=\max\{k-1-h, 0\}}^{k-1} |\delta_l|$. For the base case, when \mathbf{n} is a leaf and $C_{\mathbf{n}}$ is a tree with a single node, we have that $(\nu \mathbf{z}_{\mathbf{n}}) \mathbb{B}(C_{\mathbf{n}}) \equiv (\nu \mathbf{x}_{\mathbf{n}}) q[k](\mathbf{x}_{\mathbf{n}})$ for some q and some vector $\mathbf{x}_{\mathbf{n}}$ of size $|\delta_{k-1}|$. This shows that the claim is true for the base case.

For the induction step, let $Ch(\mathbf{n})$ be the children of \mathbf{n} . By the requirements imposed upon $\mathbb{B}(C)$, we can use the scope extrusion rule to write $(\nu \mathbf{z}_{\mathbf{n}}) \mathbb{B}(C_{\mathbf{n}})$ as $(\nu \mathbf{x}_{\mathbf{n}}, \mathbf{y}_{\mathbf{n}}) (A_{\mathbf{n}}(\mathbf{x}_{\mathbf{n}}, \mathbf{y}_{\mathbf{n}}) \mid B_{\mathbf{n}}(\mathbf{y}_{\mathbf{n}}) \mid \prod_{v \in Ch(\mathbf{n})} (\nu(\mathbf{z}_v \setminus \mathbf{y}_{\mathbf{n}})) \mathbb{B}(C_v))$. By induction hypothesis, we have that the depth of each $(\nu \mathbf{z}_v) \mathbb{B}(C_v)$ is $\sum_{l=k-h}^{k-1} |\delta_l|$. This then implies that the depth of $(\nu \mathbf{x}_{\mathbf{n}}, \mathbf{y}_{\mathbf{n}}) (A_{\mathbf{n}}(\mathbf{x}_{\mathbf{n}}, \mathbf{y}_{\mathbf{n}}) \mid B_{\mathbf{n}}(\mathbf{y}_{\mathbf{n}}) \mid \prod_{v \in Ch(\mathbf{n})} (\nu(\mathbf{z}_v \setminus \mathbf{y}_{\mathbf{n}})) \mathbb{B}(C_v))$ is at most $\sum_{l=k-1-h}^{k-1} |\delta_l|$ if \mathbf{n} is not the root. If \mathbf{n} is the root, then the depth becomes at most $\sum_{l=0}^{k-1} |\delta_l|$ because $\mathbf{x}_{\mathbf{n}} = \emptyset$. Hence, the induction step is complete.

Since $\mathbb{B}(C) \equiv (\nu \mathbf{z}_{\mathbf{n}}) \mathbb{B}(C_{\mathbf{n}})$ where \mathbf{n} is the root, it follows that the depth of $\mathbb{B}(C)$ is at most $\sum_{l=0}^{k-1} |\delta_l|$. ◀

Initial configuration. Recall that for each $i \in \{0, \dots, k-1\}$, we let \mathbf{n}_i denote some fixed vector comprising all the names from δ_i . We then take the initial configuration of \mathcal{P} to be $(\nu \mathbf{n}_0)(q_{in}[0](\mathbf{n}_0) \mid \text{start}[1](\mathbf{n}_0))$. Note that the initial configuration of \mathcal{P} is the image of the initial configuration of \mathcal{N} under the \mathbb{B} mapping.

Parametric equations

Before we describe the parametric equations, we set up some notation. Let $r = ((q_0, \dots, q_i), (q'_0, \dots, q'_j))$ be a rule of the NCSL \mathcal{N} . By definition of creation and 2-preservation rules, it has to be the case that $i \leq 1$ and $j = 1$. In the sequel, for the sake of uniformity across all rules, we adopt the following nomenclature: If $i = 0$, we let $q_1 = \text{start}$. In this way, we can always associate a (unique) tuple $((q_0, q_1), (q'_0, q'_1))$ with any rule r .

Let $r = ((p, q), (p', q'))$ be a rule of \mathcal{N} . We say that the tuple (p, q) (resp. (p', q')) is the *precondition* (resp. *postcondition*) of r and we let $\text{pre}_{\text{fi}}^r := p$, $\text{pre}_{\text{se}}^r := q$, $\text{post}_{\text{fi}}^r := p'$ and $\text{post}_{\text{se}}^r := q'$.

We will set up the parametric equations in such a way so that $C \rightarrow C'$ is a step in \mathcal{N} iff $\mathbb{B}(C) \rightarrow \mathbb{B}(C')$. Intuitively this is accomplished by ensuring that if $r = ((p, q), (p', q')) \in \delta_l$ is a rule of \mathcal{N} , then a thread with identifier $p[l]$ can output along a name and go to $p'[l]$ and a thread with identifier $q[l+1]$ can receive along the same name and go to $q'[l+1]$.

Equations for identifiers of grade 0. For any $q \in Q$, the equation for $q[0]$ is,

$$q[0](\mathbf{n}_0) := \sum_{r \in \delta_0, \text{pre}_{\text{fi}}^r = q} \bar{r}(). \text{post}_{\text{fi}}^r[0](\mathbf{n}_0)$$

Intuitively, this equation corresponds to a thread with identifier $q[0]$ trying to execute some rule $r \in \delta_0$ for which $q = \text{pre}_{\text{fi}}^r$ and then becoming $\text{post}_{\text{fi}}^r[0]$.

Equations for identifiers of grade $1 \leq i \leq k-1$. Recall that the arity of any such identifier is $|\delta_{i-1}| + |\delta_i|$, except for identifiers with base *start*, for which it is $|\delta_{i-1}|$.

■ For any $q \in Q$, we have

$$q[i](\mathbf{n}_{i-1}, \mathbf{n}_i) := \sum_{r \in \delta_i, \text{pre}_{\text{fi}}^r = q} \bar{r}(). \text{post}_{\text{fi}}^r[i](\mathbf{n}_{i-1}, \mathbf{n}_i) + \sum_{r \in \delta_{i-1}, \text{pre}_{\text{se}}^r = q} r(). \text{post}_{\text{se}}^r[i](\mathbf{n}_{i-1}, \mathbf{n}_i)$$

Intuitively, the first summand of the equation corresponds to a thread with identifier $q[i]$ trying to execute some rule $r \in \delta_i$ for which $q = \text{pre}_{\text{fi}}^r$ and then becoming $\text{post}_{\text{fi}}^r[i]$. The second summand corresponds to a thread with identifier $q[i]$ trying to execute some rule $r \in \delta_{i-1}$ for which $q = \text{pre}_{\text{se}}^r$ and then becoming $\text{post}_{\text{se}}^r[i]$.

■ For the *start* base, we have

$$\text{start}[i](\mathbf{n}_{i-1}) := \sum_{r \in \delta_{i-1}, \text{pre}_{\text{se}}^r = \text{start}} r(). \left((\nu \mathbf{n}_i) \text{start}[i](\mathbf{n}_{i-1}) \mid \text{post}_{\text{se}}^r[i](\mathbf{n}_{i-1}, \mathbf{n}_i) \mid \text{start}[i+1](\mathbf{n}_i) \right)$$

Intuitively, this equation is responsible for spawning new threads of grade i with base in Q , when an appropriate output action is taken by some thread of grade $i-1$ with base in Q . First, if a thread with identifier $\text{start}[i]$ receives a message along some channel corresponding to some rule $r \in \delta_{i-1}$ with $\text{pre}_{\text{se}}^r = \text{start}$, then a fresh set of names (denoted by \mathbf{n}_i) are created. After that, the thread retains its identifier and two new threads are spawned, $\text{post}_{\text{se}}^r[i](\mathbf{n}_{i-1}, \mathbf{n}_i)$ and $\text{start}[i+1](\mathbf{n}_i)$. We note that these equations have a similar flavor to that of the equations for *New1* and *New2* given in Example 2.

Equations for identifiers of grade k . Recall that the arity of any identifier with grade k is $|\delta_{k-1}|$.

- For any $q \in Q$, we have

$$q[k](\mathbf{n}_{k-1}) := \sum_{r \in \delta_{k-1}, \text{pre}_{\text{se}}^r = q} r(). \text{post}_{\text{se}}^r[k](\mathbf{n}_{k-1})$$

- For the *start* base, we have

$$\text{start}[k](\mathbf{n}_{k-1}) := \sum_{r \in \delta_{k-1}, \text{pre}_{\text{se}}^r = \text{start}} r(). (\text{post}_{\text{se}}^r[k](\mathbf{n}_{k-1}) \mid \text{start}[k](\mathbf{n}_{k-1}))$$

The intuitions behind these equations are the same as the one for the previous case.

3.2 Proof of correctness

We now formally show the proof of correctness of our reduction. We begin with a lemma which shows that the constructed process \mathcal{P} can simulate the NCSL \mathcal{N} .

► **Lemma 7** (\mathcal{P} simulates \mathcal{N}). *Suppose $C \rightarrow C'$ is a step in \mathcal{N} . Then $\mathbb{B}(C) \rightarrow \mathbb{B}(C')$.*

Proof. Let $r = ((p, q), (p', q')) \in \delta_l$ for some $0 \leq l \leq k-1$ such that $C \xrightarrow{r} C'$. Let V be the set of vertices of C and let IV be the set of internal vertices of C . This means that there is a vertex \mathbf{n} in C at depth l such that the label of \mathbf{n} in C is p .

Let $\mathbb{B}(C) \equiv (\nu \mathbf{z}) (\prod_{v \in V} A_v(\mathbf{x}_v, \mathbf{y}_v) \mid \prod_{v \in IV} B_v(\mathbf{y}_v))$. By definition of the map \mathbb{B} , it has to be the case that $A_{\mathbf{n}} = p[l]$. We have two cases:

- Suppose $q \neq \text{start}$. Then there has to be a child \mathbf{n}' of \mathbf{n} in C such that its label in C is q . Hence, $A_{\mathbf{n}'} = q[l+1]$. Further, $\mathbf{y}_{\mathbf{n}} = \mathbf{x}_{\mathbf{n}'}$. By construction of the parametric equations, this means that $\mathbb{B}(C)$ can reach P where

$$P \equiv (\nu \mathbf{z}) (\prod_{v \in V \setminus \{\mathbf{n}, \mathbf{n}'\}} A_v(\mathbf{x}_v, \mathbf{y}_v) \mid p'[l](\mathbf{x}_{\mathbf{n}}, \mathbf{y}_{\mathbf{n}}) \mid q'[l+1](\mathbf{x}_{\mathbf{n}'}, \mathbf{y}_{\mathbf{n}'}) \mid \prod_{v \in IV} B_v(\mathbf{y}_v))$$

It is then easy to see that $P \equiv \mathbb{B}(C')$.

- Suppose $q = \text{start}$. Then $B_{\mathbf{n}}(\mathbf{y}_{\mathbf{n}}) = \text{start}[l+1](\mathbf{y}_{\mathbf{n}})$. By construction of the parametric equations, this means that $\mathbb{B}(C)$ can reach P given by

$$P \equiv (\nu \mathbf{z}, \mathbf{z}') (\prod_{v \in V \setminus \{\mathbf{n}\}} A_v(\mathbf{x}_v, \mathbf{y}_v) \mid p'[l](\mathbf{x}_{\mathbf{n}}, \mathbf{y}_{\mathbf{n}}) \mid \prod_{v \in IV} B_v(\mathbf{y}_v) \mid q'[l+1](\mathbf{y}_{\mathbf{n}}, \mathbf{z}') \mid \text{start}[l+2](\mathbf{z}'))$$

where the last term $\text{start}[l+2](\mathbf{z}')$ is not present if $l = k-1$. It is then easy to see that $P \equiv \mathbb{B}(C')$. ◀

Next we show that \mathcal{N} can also simulate \mathcal{P} .

► **Lemma 8** (\mathcal{N} simulates \mathcal{P}). *Suppose $\mathbb{B}(C) \rightarrow P$. Then there exists a configuration C' of \mathcal{N} such that $C \rightarrow C'$ and $P \equiv \mathbb{B}(C')$.*

Proof. Let V be the vertices of C and let IV be the set of internal vertices of C . Let $\mathbb{B}(C) \equiv (\nu \mathbf{z}) (\prod_{v \in V} A_v(\mathbf{x}_v, \mathbf{y}_v) \mid \prod_{v \in IV} B_v(\mathbf{y}_v))$ and let $\mathbb{B}(C) \xrightarrow{T_v(\mathbf{w}_v), c, T_{v'}(\mathbf{w}_{v'})} P$.

By construction of the parametric equations, it must be the case that $T_v(\mathbf{w}_v) = A_{\mathbf{n}}(\mathbf{x}_{\mathbf{n}}, \mathbf{y}_{\mathbf{n}})$ for some node \mathbf{n} and c must belong to $\{\mathbf{y}_{\mathbf{n}}\}$. Let $A_{\mathbf{n}} = p[l]$. Since $c \in \{\mathbf{y}_{\mathbf{n}}\}$, by definition of $\mathbb{B}(C)$, c can only be shared among the free names of the threads in $\{A_{\mathbf{n}'}(\mathbf{x}_{\mathbf{n}'}, \mathbf{y}_{\mathbf{n}'}) : \mathbf{n}' \text{ is a child of } \mathbf{n} \} \cup \{B_{\mathbf{n}}(\mathbf{y}_{\mathbf{n}})\}$. We now consider two cases:

17:10 Complexity of Coverability in Depth-Bounded Processes

- Suppose $T_{v'}(\mathbf{w}_{v'}) = A_{n'}(\mathbf{x}_{n'}, \mathbf{y}_{n'})$ for some n' which is a child of n . Let $A_{n'} = q[l+1]$. Since we have $\mathbb{B}(C) \xrightarrow{T_v(\mathbf{w}_v), c, T_{v'}(\mathbf{w}_{v'})} P$, by construction of the equations it has to be the case that there is a rule $r \in \delta_l$ of \mathcal{N} such that $\mathbf{pre}_{\mathbf{f}_i}^r = p$, $\mathbf{pre}_{\mathbf{se}}^r = q$ and

$$P \equiv (\nu \mathbf{z}) (\Pi_{v \in V \setminus \{n, n'\}} A_v(\mathbf{x}_v, \mathbf{y}_v) \mid p'[l](\mathbf{x}_n, \mathbf{y}_n) \mid q'[l+1](\mathbf{x}_{n'}, \mathbf{y}_{n'}) \mid \Pi_{v \in IV} B_v(\mathbf{y}_v))$$

where $p' = \mathbf{post}_{\mathbf{f}_i}^r$ and $q' = \mathbf{post}_{\mathbf{se}}^r$ respectively. Since $A_n = p[l]$ and $A_{n'} = q[l+1]$, it must be the case that the depth of n in C is l and the labels of n and n' in C are p and q respectively. It follows that there exists C' such that $C \xrightarrow{r} C'$. It is then easy to verify that $\mathbb{B}(C') \equiv P$.

- Suppose $T_{v'}(\mathbf{w}_{v'}) = B_n(\mathbf{y}_n)$. We know that $B_n = \mathit{start}[l+1]$. Since it is the case that $\mathbb{B}(C) \xrightarrow{T_v(\mathbf{w}_v), c, T_{v'}(\mathbf{w}_{v'})} P$, by construction of the parametric equations it must be that there is a rule $r \in \delta_l$ of \mathcal{N} such that $\mathbf{pre}_{\mathbf{f}_i}^r = p$, $\mathbf{pre}_{\mathbf{se}}^r = \mathit{start}$ and

$$P \equiv (\nu \mathbf{z}, \mathbf{z}') (\Pi_{v \in V \setminus \{n\}} A_v(\mathbf{x}_v, \mathbf{y}_v) \mid p'[l](\mathbf{x}_n, \mathbf{y}_n) \mid \Pi_{v \in IV} B_v(\mathbf{y}_v) \mid q'[l+1](\mathbf{y}_n, \mathbf{z}') \mid \mathit{start}[l+2](\mathbf{z}'))$$

where the last term $\mathit{start}[l+2](\mathbf{z}')$ is not present if $l = k-1$ and $p' = \mathbf{post}_{\mathbf{f}_i}^r, q' = \mathbf{post}_{\mathbf{se}}^r$ respectively. Since $A_n = p[l]$, it must be the case that the depth of n in C is l and the label of n in C is p . It follows then that there exists C' such that $C \xrightarrow{r} C'$. It is then easy to verify that $\mathbb{B}(C') \equiv P$. ◀

Note that the initial configuration I of \mathcal{P} is simply the image of the initial configuration of \mathcal{N} under the map \mathbb{B} . Hence, using Lemmas 6 and 8, we can conclude that

► **Corollary 9.** *The process \mathcal{P} is K -depth-bounded where $K = \sum_{l=0}^{k-1} |\delta_l|$.*

We then get the following theorem, whose proof follows in a straightforward manner by combining Lemmas 7 and 8.

► **Theorem 10.** *$C \xrightarrow{*} C'$ is a run in \mathcal{N} iff $\mathbb{B}(C) \xrightarrow{*} \mathbb{B}(C')$ is a run in the process \mathcal{P} . Consequently q_{in} can cover q_f in \mathcal{N} iff $(\nu \mathbf{n}_0) (q_f[0](\mathbf{n}_0))$ can be covered from the initial configuration I of \mathcal{P} .*

Hence, we have

► **Corollary 11.** *Coverability of depth-bounded processes is \mathbf{F}_{ϵ_0} -hard.*

4 Nested counter systems (NCS)

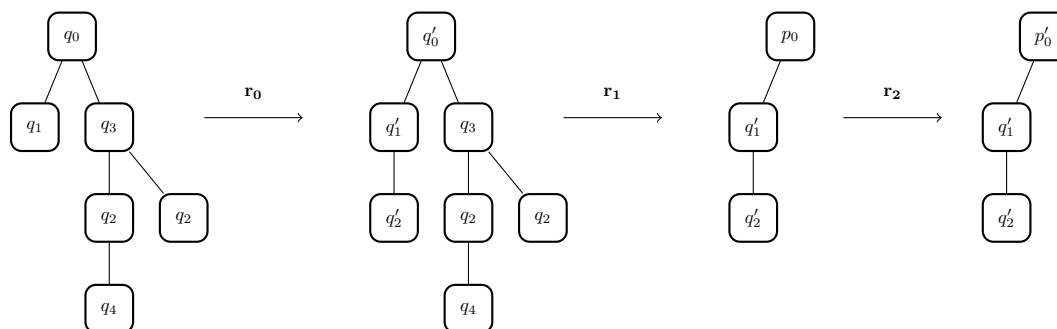
We now prove Theorem 5, by giving a reduction from the coverability problem for *nested counter systems* (NCS) which is known to be \mathbf{F}_{ϵ_0} -hard. We first recall the definition of NCS, which we present in a way that is akin to [2].

A k -nested counter system (k -NCS) is a tuple $\mathcal{N} = (Q, \delta)$ where Q is a finite set of *states* and $\delta \subseteq \bigcup_{1 \leq i, j \leq k+1} (Q^i \times Q^j)$ is a set of *rules*. The set $\mathcal{C}_{\mathcal{N}}$ of *configurations* of \mathcal{N} is defined to be the set of all labelled rooted trees of height at most k , with labels from the set Q .

The operational semantics of \mathcal{N} is defined in terms of the following transition relation $\rightarrow_{\subseteq} \mathcal{C}_{\mathcal{N}} \times \mathcal{C}_{\mathcal{N}}$ on configurations: Let $r := ((q_0, \dots, q_i), (q'_0, \dots, q'_j)) \in \delta$ be a rule with $i \leq j \leq k$. We say that a configuration C can move to the configuration C' using the rule r (denoted by $C \xrightarrow{r} C'$), if there is a path v_0, v_1, \dots, v_i in C starting at the root such that for every $0 \leq l \leq i$, the label of v_l is q_l and, C' is obtained from C by 1) for every $0 \leq l \leq i$, changing the label of each v_l to q'_l and 2) for every $i+1 \leq l \leq j$, creating a new vertex v_l with label q'_l and adding it as a child to v_{l-1} .

Similarly, suppose $r := ((q_0, \dots, q_i), (q'_0, \dots, q'_j)) \in \delta$ is a rule with $j < i \leq k$. Then $C \xrightarrow{r} C'$ if there is a path v_0, v_1, \dots, v_i in C starting at the root such that for every $0 \leq l \leq i$, the label of v_l is q_l and, C' is obtained from C by 1) for every $0 \leq l \leq j$, changing the label of each v_l to q'_l and 2) removing the subtree rooted at the node v_{j+1} .

► **Example 12** (Example from [2]). Let us consider the NCS \mathcal{N} given by the states $Q = \{p_i, p'_i, q_i, q'_i : 0 \leq i \leq 4\}$ and consisting of the following rules: $r_0 = ((q_0, q_1), (q'_0, q'_1, q'_2))$, $r_1 = ((q'_0, q_3, q_2), (p_0))$, $r_2 = ((p_0), (p'_0))$. In Figure 2, we illustrate the application of these rules to a configuration of \mathcal{N} .



■ **Figure 2** Application of the rules r_0, r_1 and r_2 to a configuration of \mathcal{N} , which is described in Example 12.

Similar to NCSL, we can define the notions of $C \rightarrow C'$, $C \xrightarrow{*} C'$ and a state q_{in} covering another state q_f . It is known that the coverability problem for NCS is \mathbf{F}_{e_0} -hard (Theorem 7 of [8]).

We note that the rules of an NCS act “globally”, in the sense that it allows to update the value of (potentially) k many counters in one step. This is in contrast to NCSL, where we can update the value of at most two counters at a time. While it is not particularly surprising that this “global” update can be replaced by a series of “local” updates (hence giving a reduction from NCS to NCSL), the construction is not entirely trivial and requires some intricate arguments in order to prove its correctness.

A special case of NCS

We make a small remark which will help us simplify our reduction later on. Let $\mathcal{N} = (Q, \delta)$ be a k -NCS and let $q_{in}, q_f \in Q$. From \mathcal{N} , we construct a new k -NCS \mathcal{N}' as follows: First we add a new state end . Then, if $r = ((q_0, \dots, q_i), (q'_0, \dots, q'_j)) \in \delta$ with $j < i \leq k$, we replace r with the rule $r' := ((q_0, \dots, q_i), (q'_0, \dots, q'_j, \underbrace{end, \dots, end}_{i-j \text{ times}}))$. Intuitively, we are replacing

all rules which destroy some counters with corresponding rules that simply convert those counters to the state end . It can be easily verified that coverability of q_f from q_{in} is preserved while doing this operation. Hence, from here on, we assume that whenever $\mathcal{N} = (Q, \delta)$ is a k -NCS and $r = ((q_0, \dots, q_i), (q'_0, \dots, q'_j)) \in \delta$ then $i \leq j$.

4.1 Hardness of coverability for NCSL

We shall prove Theorem 5 by giving a reduction from the coverability problem for NCS. Let $k \geq 1$ and let $\mathcal{N} = (Q, \delta)$ be a k -NCS with two fixed states q_{in} and q_f . By the argument given in the previous paragraph, we can assume that if $r = ((q_0, \dots, q_i), (q'_0, \dots, q'_j)) \in \delta$

17:12 Complexity of Coverability in Depth-Bounded Processes

then $i \leq j$. We shall now construct a k -NCSL $\mathcal{N}' = (Q', \delta'_0, \dots, \delta'_{k-1})$ and two states q'_{in} and q'_f of \mathcal{N}' such that q'_{in} can cover q'_f in \mathcal{N}' iff q_{in} can cover q_f in \mathcal{N} . This will then prove Theorem 5. We begin by describing the states of \mathcal{N}' .

States of \mathcal{N}' . For every state q of \mathcal{N} , we will have two states $q[\top]$ and $q[\perp]$. Further, for every rule r of \mathcal{N} , we will have four states $\mathbf{rec}^r[\top], \mathbf{rec}^r[\perp], \mathbf{fwd}^r[\top]$ and $\mathbf{fwd}^r[\perp]$. Notice that each state of \mathcal{N}' is of the form $a[b]$ where $a \in Q \cup \{\mathbf{rec}^r, \mathbf{fwd}^r : r \in \delta\}$ and $b \in \{\top, \perp\}$. If a node v in a configuration C has as its label $a[b]$, then ‘a’ will be called its *base*. Further, if $b = \top$ (resp. $b = \perp$), then v will be called as a *leader node* (resp. *follower node*).

Good configurations of \mathcal{N}' . A configuration C is called good if the root of C is a leader, all other nodes are followers and the base of all the nodes of C belong to Q . Notice that there is a straightforward bijection between the set of all configurations of \mathcal{N} and the set of all good configurations of \mathcal{N}' . This bijection will be denoted by \mathbb{M} .

Rules of \mathcal{N}' . Before we describe the rules of \mathcal{N}' , we will state two invariants that will always be maintained by our construction. The first one is that, in any configuration reachable from a good configuration, exactly one node will be a leader. The second invariant is that, every rule of \mathcal{N}' will have a leader state in its precondition. Combined with the first invariant, this will intuitively ensure that the rules that can be fired from reachable configurations are limited and will help us simplify the proof of correctness of our reduction.

We now describe the rules of \mathcal{N}' . Let $r = ((q_0, \dots, q_i), (q'_0, \dots, q'_j))$ be a rule of \mathcal{N} . Corresponding to rule r , we will have the following set of rules in \mathcal{N}' . (In the following, we adopt the convention that if the name of a rule has a subscript $0 \leq l \leq j$, then that rule belongs to δ'_l).

- $start_0^r := ((q_0[\top]), (\mathbf{rec}^r[\top]))$.
- For every $0 \leq l \leq i-1$, we have a rule $begin_l^r := ((\mathbf{rec}^r[\top], q_{l+1}[\perp]), (\mathbf{fwd}^r[\perp], \mathbf{rec}^r[\top]))$.
- For every $i \leq l \leq j-1$, we have a rule $begin_l^r := ((\mathbf{rec}^r[\top]), (\mathbf{fwd}^r[\perp], \mathbf{rec}^r[\top]))$.
- $middle_j^r := ((\mathbf{rec}^r[\top]), (\mathbf{fwd}^r[\top]))$.
- For every $0 \leq l \leq j-1$, we have a rule $end_l^r := ((\mathbf{fwd}^r[\perp], \mathbf{fwd}^r[\top]), (\mathbf{fwd}^r[\top], q'_{l+1}[\perp]))$.
- $finish_0^r := ((\mathbf{fwd}^r[\top]), q'_0[\top])$.

4.2 Proof of correctness

The intuitive idea behind the above gadget is given by the run demonstrated in the following lemma.

► **Lemma 13** (\mathcal{N}' simulates \mathcal{N}). *Suppose $C \xrightarrow{r} C'$ is a step in the NCS \mathcal{N} . Then, there is a run $\mathbb{M}(C) \xrightarrow{*} \mathbb{M}(C')$ in the NCSL \mathcal{N}' .*

Proof. Let $r = ((q_0, \dots, q_i), (q'_0, \dots, q'_j))$. Since $C \xrightarrow{r} C'$ is a step in \mathcal{N} , it follows that there is a path starting at the root of C labelled by q_0, \dots, q_i . It follows that in $\mathbb{M}(C)$ there is a path P starting at the root labelled by $q_0[\top], q_1[\perp], q_2[\perp], \dots, q_i[\perp]$. We now execute a sequence of rules according to the gadget for r as follows:

- First, using $start_0^r$, we change the label of the root from $q_0[\top]$ to $\mathbf{rec}^r[\top]$.
- Next, by firing $begin_0^r, \dots, begin_{i-1}^r$ in this order, we change the labels of the nodes in the path P to $\underbrace{\mathbf{fwd}^r[\perp], \dots, \mathbf{fwd}^r[\perp]}_{i \text{ times}}, \mathbf{rec}^r[\top]$.

- Then, by firing $begin_i^r, \dots, begin_{j-1}^r$ in this order, we add $j - i$ new nodes to the path P and get a new path P' of length $j + 1$ whose labels are $\underbrace{\mathbf{fwd}^r[\perp], \dots, \mathbf{fwd}^r[\perp]}_{j \text{ times}}, \mathbf{rec}^r[\top]$.
 - We use $middle_j^r$ to change the label of the last node in P' from $\mathbf{rec}^r[\top]$ to $\mathbf{fwd}^r[\top]$.
 - Then, by firing $end_{j-1}^r, \dots, end_0^r$ in this order, we change the labels of the nodes in the path P' to $\mathbf{fwd}^r[\top], q'_1[\perp], \dots, q'_j[\perp]$.
 - Finally, we use $finish_0^r$ to change the label of the root from $\mathbf{fwd}^r[\top]$ to $q'_0[\top]$.
- It can be easily verified that the resulting configuration D is such that $D = \mathbb{M}(C')$. ◀

We now present a converse to the above lemma which shows that a simulation in the other direction is also possible.

► **Lemma 14** (\mathcal{N} simulates \mathcal{N}'). *Suppose $C \xrightarrow{*} C'$ is a path of non-zero length in \mathcal{N}' such that 1) C is a good configuration and 2) in all the configurations between C and C' , the base of the root is not in Q . Then, C' is a good configuration and there is a rule r such that $\mathbb{M}^{-1}(C) \xrightarrow{r} \mathbb{M}^{-1}(C')$.*

Proof sketch. Let $P := C \rightarrow \gamma_0 \rightarrow \gamma_1 \dots \rightarrow C'$. The essential idea behind this lemma is that since C is a good configuration, the root node is a leader node and by the construction of the rules it must be the case that the first step must be of the form $C \xrightarrow{start_0^r} \gamma_0$ for some rule r . Then, by using the invariant that exactly one node is leader at all times and by using the construction of the rules, we can essentially show that P must be a path of the same form as the one given in the proof of Lemma 13. Having proved that, we can then show that in the NCS \mathcal{N} , $\mathbb{M}^{-1}(C) \xrightarrow{r} \mathbb{M}^{-1}(C')$. ◀

Because of these two “simulation” lemmas, we then get

► **Theorem 15.** q_{in} can cover q_f in \mathcal{N} iff $q_{in}[\top]$ can cover $q_f[\top]$ in \mathcal{N}' .

4.3 Wrapping up

The previous theorem implies that coverability for NCSL is \mathbf{F}_{ϵ_0} -hard. To prove Theorem 5, we need to show the same for NCSL with only creation and 2-preservation rules. We now show that 1-preservation rules can be replaced with creation rules in an NCSL while maintaining coverability.

Given a k -NCSL \mathcal{N} with two states q_{in}, q_f , we can remove all 1-preservation rules whilst preserving coverability as follows: We first add a new state end . Then if $r = ((q_0), (q'_0))$ is a 1-preservation rule in \mathcal{N} , we replace r with $r = ((q_0), (q'_0, end))$. It can be easily seen that doing this procedure gives us a $(k + 1)$ -NCSL \mathcal{N}' such that q_{in} can cover q_f in \mathcal{N}' iff q_{in} can cover q_f in \mathcal{N} . Hence Theorem 5 follows.

5 Upper bound for coverability of depth-bounded processes

We now prove the upper bound claim made in Theorem 3. Let $\mathcal{P} = (I, \mathcal{E})$ be a fixed k -depth-bounded process. By introducing new identifiers and equations if necessary, we can assume that at most one name or thread is created during a step between two configurations of \mathcal{P} . Let us consider the following order on the set of configurations: $P \preceq Q$ iff $P \equiv (\nu \mathbf{x})P'$ and $Q \equiv (\nu \mathbf{x})(P' \mid R)$ for some term R . It is known that this is a well-quasi order (wqo) for the set of all k -depth-bounded configurations [20, 27]. Using this fact, we can show that the set of k -depth-bounded configurations of \mathcal{P} , forms a well-structured transition system (WSTS) under the \preceq ordering and then apply the generic backward exploration algorithm for

WSTS [13, 25]. Using the standard and generic complexity arguments for WSTS [26, 13, 25], an upper bound on the the running time of this procedure simply boils down to estimating the length of *controlled bad sequences* of k -depth-bounded configurations under the \preceq order.

Let the size of a configuration C be the number of names and threads that appear in C . Let $H : \mathbb{N} \rightarrow \mathbb{N}$ be the successor function and let $n \in \mathbb{N}$. For each $i \in \mathbb{N}$, we let H^i denote the i -fold application of H to itself i times, with H^0 being the identity function.

► **Definition 16.** A sequence C_0, C_1, \dots , of configurations is called (H, n) -controlled bad if the size of each C_i is at most $H^i(n)$ and $C_i \not\preceq C_j$ for any $i < j$.

To estimate an upper bound on the length of controlled bad sequences of configurations, we first recall the *induced subgraph ordering* on bounded-depth trees.

► **Definition 17.** Let $T_1 = (V_1, E_1, L_1)$ and $T_2 = (V_2, E_2, L_2)$ be two labelled trees with labelling functions $L_1 : V_1 \rightarrow A$ and $L_2 : V_2 \rightarrow A$ for some finite set A . We say that T_1 is an induced subgraph of T_2 , if there is a label preserving injection h from V_1 to V_2 such that $(v, v') \in E_1 \iff (h(v), h(v')) \in E_2$.

It is known that for any $K \geq 1$ and for any finite set A , the set of all labelled trees of depth at most K is well-quasi ordered under the induced subgraph relation (Theorem 2.2 of [9]). Similar to configurations, we can also define controlled bad sequences of labelled bounded-depth trees.

By the arguments given in [20], it follows that the length of controlled bad sequences of k -depth-bounded configurations of \mathcal{P} under the \preceq order can be upper bounded by the length of controlled bad sequences of K -bounded-depth trees with labels from a set A , for some A and K whose sizes are primitive recursive in the size of \mathcal{P} . By the known bounds for controlled bad sequences for labelled bounded-depth trees [2, 17], it follows that

► **Theorem 18.** The length of (H, n) -controlled bad sequences for k -depth-bounded configurations of \mathcal{P} is upper bounded by the function $F_{\epsilon_0}(p(|\mathcal{P}|, k, n))$.

Here F_{ϵ_0} is the *fast-growing function* at level ϵ_0 and p is some primitive recursive function. For our purposes, we do not need the actual definition of F_{ϵ_0} , but we only need to know that \mathbf{F}_{ϵ_0} consists of problems whose running time is upper bounded by the function F_{ϵ_0} composed with any primitive recursive function (See [24]). It follows that,

► **Theorem 19.** The coverability problem for depth-bounded processes is in \mathbf{F}_{ϵ_0} and hence \mathbf{F}_{ϵ_0} -complete.

6 Conclusion

We have shown that the coverability problem for depth-bounded processes in π -calculus is \mathbf{F}_{ϵ_0} -complete. This settles the complexity of the problem and solves an open problem raised in [17] and also in [27]. However, our proof does not give any results regarding the *parameterized complexity* of this problem when the depth k is taken as a parameter, which we plan to investigate as part of future work.

References

- 1 Sergio Abriola, Santiago Figueira, and Gabriel Senno. Linearizing well quasi-orders and bounding the length of bad sequences. *Theor. Comput. Sci.*, 603:3–22, 2015. doi:10.1016/j.tcs.2015.07.012.

- 2 A. R. Balasubramanian. Complexity of coverability in bounded path broadcast networks. In Mikolaj Bojanczyk and Chandra Chekuri, editors, *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*, volume 213 of *LIPICs*, pages 35:1–35:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.FSTTCS.2021.35.
- 3 Michael Blondin. The ABCs of petri net reachability relaxations. *ACM SIGLOG News*, 7(3):29–43, 2020. doi:10.1145/3436980.3436984.
- 4 Michael Blondin, Alain Finkel, Christoph Haase, and Serge Haddad. The logical view on continuous Petri nets. *ACM Trans. Comput. Log.*, 18(3):24:1–24:28, 2017. doi:10.1145/3105908.
- 5 Michael Blondin and Christoph Haase. Logics for continuous reachability in Petri nets and vector addition systems with states. 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.2017.8005068.
- 6 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, 24-27 June 2008, Pittsburgh, PA, USA*, pages 205–216, 2008. doi:10.1109/LICS.2008.47.
- 7 Wojciech Czerwinski and Lukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1229–1240. IEEE, 2021. doi:10.1109/FOCS52979.2021.00120.
- 8 Normann Decker and Daniel Thoma. On freeze LTL with ordered attributes. In Bart Jacobs and Christof Löding, editors, *Foundations of Software Science and Computation Structures - 19th International Conference, FOSSACS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9634 of *Lecture Notes in Computer Science*, pages 269–284. Springer, 2016. doi:10.1007/978-3-662-49630-5_16.
- 9 Guoli Ding. Subgraphs and well-quasi-ordering. *Journal of Graph Theory*, 16(5):489–502, 1992. doi:10.1002/jgt.3190160509.
- 10 Jacob Elgaard, Nils Klarlund, and Anders Møller. MONA 1.x: New techniques for WS1S and WS2S. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification*, pages 516–520, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.
- 11 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In Ernst W. Mayr and Natacha Portier, editors, *31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014), STACS 2014, March 5-8, 2014, Lyon, France*, volume 25 of *LIPICs*, pages 1–10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPICs.STACS.2014.1.
- 12 Javier Esparza, Ruslán Ledesma-Garza, Rupak Majumdar, Philipp J. Meyer, and Filip Nijksic. An SMT-based approach to coverability analysis. In Armin Biere and Roderick Bloem, editors, *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 603–619. Springer, 2014. doi:10.1007/978-3-319-08867-9_40.
- 13 Diego Figueira, Santiago Figueira, Sylvain Schmitz, and Philippe Schnoebelen. Ackermannian and primitive-recursive bounds with Dickson’s lemma. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science*, pages 269–278, 2011. doi:10.1109/LICS.2011.39.
- 14 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.
- 15 Estibaliz Fraca and Serge Haddad. Complexity analysis of continuous Petri nets. *Fundam. Informaticae*, 137(1):1–28, 2015. doi:10.3233/FI-2015-1168.

- 16 Christoph Haase and Simon Halfon. Integer vector addition systems with states. In Joël Ouaknine, Igor Potapov, and James Worrell, editors, *Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*, volume 8762 of *Lecture Notes in Computer Science*, pages 112–124. Springer, 2014. doi:10.1007/978-3-319-11439-2_9.
- 17 Christoph Haase, Sylvain Schmitz, and Philippe Schnoebelen. The power of priority channel systems. *Log. Methods Comput. Sci.*, 10(4), 2014. doi:10.2168/LMCS-10(4:4)2014.
- 18 Sławomir Lasota. Improved Ackermannian lower bound for the petri nets reachability problem. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPICs*, pages 46:1–46:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.STACS.2022.46.
- 19 Jérôme Leroux. The reachability problem for petri nets is not primitive recursive. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1241–1252. IEEE, 2021. doi:10.1109/FOCS52979.2021.00121.
- 20 Roland Meyer. On boundedness in depth in the pi-calculus. In Giorgio Ausiello, Juhani Karhumäki, Giancarlo Mauri, and C.-H. Luke Ong, editors, *Fifth IFIP International Conference On Theoretical Computer Science – TCS 2008, IFIP 20th World Computer Congress, TC 1, Foundations of Computer Science, September 7-10, 2008, Milano, Italy*, volume 273 of *IFIP*, pages 477–489. Springer, 2008. doi:10.1007/978-0-387-09680-3_32.
- 21 Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992. doi:10.1016/0890-5401(92)90008-4.
- 22 Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, II. *Inf. Comput.*, 100(1):41–77, 1992. doi:10.1016/0890-5401(92)90009-5.
- 23 Sylvain Schmitz. Complexity bounds for ordinal-based termination - (invited talk). In *Reachability Problems - 8th International Workshop, RP 2014*, pages 1–19, 2014. doi:10.1007/978-3-319-11439-2_1.
- 24 Sylvain Schmitz. Complexity hierarchies beyond elementary. *ACM Trans. Comput. Theory*, 8(1):3:1–3:36, 2016. doi:10.1145/2858784.
- 25 Sylvain Schmitz and Philippe Schnoebelen. Multiply-recursive upper bounds with Higman’s lemma. In *Automata, Languages and Programming – 38th International Colloquium, ICALP 2011*, pages 441–452, 2011. doi:10.1007/978-3-642-22012-8_35.
- 26 Sylvain Schmitz and Philippe Schnoebelen. The power of well-structured systems. In Pedro R. D’Argenio and Hernán C. Melgratti, editors, *CONCUR 2013 – Concurrency Theory – 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013. Proceedings*, volume 8052 of *Lecture Notes in Computer Science*, pages 5–24. Springer, 2013. doi:10.1007/978-3-642-40184-8_2.
- 27 Thomas Wies, Damien Zufferey, and Thomas A. Henzinger. Forward analysis of depth-bounded processes. In C.-H. Luke Ong, editor, *Foundations of Software Science and Computational Structures, 13th International Conference, FOSSACS 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 6014 of *Lecture Notes in Computer Science*, pages 94–108. Springer, 2010. doi:10.1007/978-3-642-12032-9_8.

A Appendix

A.1 Proofs for subsection 4.2

► **Lemma 14** (\mathcal{N} simulates \mathcal{N}'). *Suppose $C \xrightarrow{*} C'$ is a path of non-zero length in \mathcal{N}' such that 1) C is a good configuration and 2) in all the configurations between C and C' , the base of the root is not in Q . Then, C' is a good configuration and there is a rule r such that $\mathbb{M}^{-1}(C) \xrightarrow{r} \mathbb{M}^{-1}(C')$.*

Proof. Let $P := C \rightarrow \gamma_0 \rightarrow \gamma_1 \dots \rightarrow \gamma_m \rightarrow C'$ be a path in \mathcal{N}' . We split the proof into various steps.

Step 1. Since C is a good configuration, the only node which is a leader is the root, whose base must belong to Q . By construction of the rules of \mathcal{N}' , this implies that the step $C \rightarrow \gamma_0$ must be of the form $C \xrightarrow{start_0^r} \gamma_0$ for some rule r of \mathcal{N} . Let $r = ((q_0, \dots, q_i), (q'_0, \dots, q'_j))$. This implies that the label of the root in C is $q_0[\top]$ and its label in γ_0 is $\mathbf{rec}^r[\top]$.

Step 2. Now, for each $0 \leq l \leq i$, we state two claims:

- Claim A_l : There is a path $P_l := v_l^0, \dots, v_l^l$ starting at the root in C with labels $q_0[\top], q_1[\perp], \dots, q_l[\perp]$ such that γ_l is the same as C , except now the labels along P_l are $\underbrace{\mathbf{fwd}^r[\perp], \dots, \mathbf{fwd}^r[\perp]}_{l \text{ times}}, \mathbf{rec}^r[\top]$.
- Claim B_l : If $l \neq 0$, then $\gamma_{l-1} \xrightarrow{begin_{l-1}^r} \gamma_l$.

We have already shown that claim A_0 is true in step 1. Now, for each $0 \leq l \leq i-1$, assuming claim A_l is true, we shall prove that claims A_{l+1} and B_{l+1} are true.

Because of claim A_l and because C is a good configuration, it follows that the only node which is a leader in γ_l is v_l^l . Further, the base of v_l^l is \mathbf{rec}^r . By construction of the rules in \mathcal{N}' , this implies that the only rule that can be fired from γ_l is $begin_l^r$. Hence, it must be the case that $\gamma_l \xrightarrow{begin_l^r} \gamma_{l+1}$, proving claim B_{l+1} . Further, since v_l^l is the only node which is a leader, firing this rule transforms the state of v_l^l to $\mathbf{fwd}^r[\perp]$ and transforms the state of a child of v_l^l (say v') from $q_{l+1}[\perp]$ to $\mathbf{rec}^r[\top]$. Taking P_{l+1} to be v_l^0, \dots, v_l^l, v' proves claim A_{l+1} .

In particular claim A_i implies that there is a path $\mathbf{path} := v^0, \dots, v^i$ starting at the root such that γ_i is the same as C , except that the labels of \mathbf{path} in C and γ_i are $q_0[\top], \dots, q_i[\perp]$ and $\underbrace{\mathbf{fwd}^r[\perp], \dots, \mathbf{fwd}^r[\perp]}_{i \text{ times}}, \mathbf{rec}^r[\top]$ respectively.

Step 3. For each $i \leq l \leq j$, we state two claims:

- Claim A_l : γ_l is the same as γ_i , except that \mathbf{path} is extended to include $l-i$ new nodes and the labels along this extended path in γ_l is $\underbrace{\mathbf{fwd}^r[\perp], \dots, \mathbf{fwd}^r[\perp]}_{l \text{ times}}, \mathbf{rec}^r[\top]$.
- Claim B_l : If $i \neq l$, then $\gamma_{l-1} \xrightarrow{begin_{l-1}^r} \gamma_l$.

We have already shown that claim A_i is true in step 2. Similar to the arguments given in step 2, we can prove that these new claims are also true.

Step 4. By claim A_j it follows that there is a path $\mathbf{ext-path} := n^0, \dots, n^j$ starting at the root in γ_j such that the labels along $\mathbf{ext-path}$ is $\underbrace{\mathbf{fwd}^r[\perp], \dots, \mathbf{fwd}^r[\perp]}_{j \text{ times}}, \mathbf{rec}^r[\top]$. Further, n^j is the only node which is a leader in γ_j . Hence, the only rule which can be fired from γ_j is $middle_j^r$ and so we have $\gamma_j \xrightarrow{middle_j^r} \gamma_{j+1}$. Notice that the only change that has occurred because of this step is that the label of n^j has been changed to $\mathbf{fwd}^r[\top]$.

Step 5. For each $1 \leq l \leq j$, we state two claims:

- Claim A'_l : γ_{j+l} is the same as γ_j , except that the labels along **ext-path** in γ_{j+l} is $\underbrace{\text{fwd}^r[\perp], \dots, \text{fwd}^r[\perp]}_{j-l+1 \text{ times}}, \text{fwd}^r[\top], q'_{j-l+2}[\perp], \dots, q'_j[\perp]$.
- Claim B'_l : $\gamma_{j+l} \xrightarrow{\text{end}^r_{j-l}} \gamma_{j+l+1}$.

The proof of this is accomplished by similar arguments as given in step 2.

Step 6. By claim A'_j , it follows that γ_{2j} is the same as γ_j , except that the labels along **ext-path** is now $\text{fwd}^r[\top], q'_1[\perp], \dots, q'_j[\perp]$. It follows that the only rule which can be fired from γ_{2j} is finish^r_0 , and so it follows that $\gamma_{2j} \xrightarrow{\text{finish}^r_0} \gamma_{2j+1}$, where the only difference between γ_{2j+1} and γ_{2j} is that the label of the root in γ_{2j+1} is $q'_0[\top]$. Hence, by assumption of the run P , it follows that $\gamma_{2j+1} = C'$.

By combining the arguments given above, it follows then that γ_{2j+1} is a good configuration and also that $\mathbb{M}^{-1}(C) \xrightarrow{r} \mathbb{M}^{-1}(C')$. ◀

► **Theorem 15.** q_{in} can cover q_f in \mathcal{N} iff $q_{in}[\top]$ can cover $q_f[\top]$ in \mathcal{N}' .

Proof. Suppose q_{in} can cover q_f in \mathcal{N} . Let $C_0 \rightarrow C_1 \rightarrow \dots \rightarrow C_m$ be a run in \mathcal{N} where C_0 is the initial configuration and the root of C_m is q_f . By Lemma 13, it follows that $\mathbb{M}(C_0) \xrightarrow{*} \mathbb{M}(C_1) \xrightarrow{*} \dots \xrightarrow{*} \mathbb{M}(C_m)$ and so $q_{in}[\top]$ can cover $q_f[\top]$ in \mathcal{N}' .

Suppose $C \xrightarrow{*} C'$ is a run in \mathcal{N}' such that C is the (unique good) configuration consisting of the single root vertex labelled by $q_{in}[\top]$ and C' is some configuration where the root is labelled by $q_f[\top]$. We split the run into parts of the form $C = C_0 \xrightarrow{*} C_1 \xrightarrow{*} C_2 \dots \xrightarrow{*} C_m = C'$ such that for each $1 \leq l \leq m$, C_l is the first configuration after C_{l-1} where the base of the root is in Q . By Lemma 14, it follows that each C_l is a good configuration and also that $\mathbb{M}^{-1}(C_0) \rightarrow \mathbb{M}^{-1}(C_1) \rightarrow \dots \rightarrow \mathbb{M}^{-1}(C_m)$. Hence, it follows that q_{in} can cover q_f in \mathcal{N} . ◀

A.2 Proofs for Section 5

We now give a proof of Theorem 19. We recall the backward exploration algorithm for well-structured transition systems (WSTS) here, adapted to the coverability problem for depth-bounded processes. Let $\mathcal{P} = (I, \mathcal{E})$ be a k -depth-bounded process and let P be some k -depth-bounded configuration, which we want to check is coverable in \mathcal{P} . Without loss of generality, we can assume that at most one name or thread is created during a step between two configurations of \mathcal{P} . Let \mathcal{C}_k be the set of all k -depth-bounded configurations.

Given a set S of \mathcal{C}_k we let $\uparrow S := \{\gamma' : \exists \gamma \in S, \gamma \preceq \gamma'\}$. A set S is called *upward-closed* if $S = \uparrow S$. Because \preceq is a wqo and because of the definition of the operational semantics of \mathcal{P} , we have that:

- If S is upward-closed, then there exists a finite set B such that $\uparrow B = S$. Such a B will be called the basis of S .
- If S is upward-closed and if $\text{Pre}(S)$ is the set of all configurations $\gamma' \in \mathcal{C}_k$ such that there is a configuration $\gamma \in S$ with $\gamma' \rightarrow \gamma$, then $S \cup \text{Pre}(S)$ is upward-closed. Moreover, given a basis B of S , we can compute a basis B' of $S \cup \text{Pre}(S)$ such that the size of each configuration in B' is at most one more than the maximum size of any configuration of B .

Hence, by the generic backward exploration algorithm for WSTS [14], we get that the following algorithm terminates and decides coverability: Construct a sequence of finite sets B_0, B_1, \dots , such that each $B_i \subseteq \mathcal{C}_k$, B_0 is simply $\{P\}$ and B_{i+1} is a basis for $\uparrow B_i \cup \text{Pre}(\uparrow B_i)$.

Then find the first m such that $\uparrow B_m = \uparrow B_{m+1}$ and check if there is an initial configuration in $\uparrow B_m$. If it is true, then P is coverable; otherwise P is not coverable.

The running time complexity of the algorithm is mainly dominated by the length of the sequence B_0, B_1, \dots, B_m . Since m is the first index such that $\uparrow B_m = \uparrow B_{m+1}$, we can find a minimal element $\gamma_i \in \uparrow B_{i+1} \setminus \uparrow B_i$ for each $i < m$.

Consider the sequence $\gamma_0, \dots, \gamma_{m-1}$. Notice that $\gamma_i \not\leq \gamma_j$ for any $j > i$ and further the size of each γ_i is at most $H^i(n)$, where H is the successor function and n is the size of P . It follows that $\gamma_0, \dots, \gamma_{m-1}$ is a (H, n) -controlled bad sequence. By the arguments given in [20], it follows that the length of controlled bad sequences of \mathcal{C}_k under the \leq order can be upper bounded by the length of controlled bad sequences of K -bounded-depth trees with labels from a set A , for some A and K whose sizes are primitive recursive in the size of \mathcal{P} . By the known bounds for controlled bad sequences for labelled bounded-depth trees [2, 17], it follows that



► **Theorem 18.** *The length of (H, n) -controlled bad sequences for k -depth-bounded configurations of \mathcal{P} is upper bounded by the function $F_{\epsilon_0}(p(|\mathcal{P}|, k, n))$.*

Here F_{ϵ_0} is the *fast-growing function* at level ϵ_0 and p is some primitive recursive function. For our purposes, we do not need the actual definition of F_{ϵ_0} , but we only need to know that \mathbf{F}_{ϵ_0} consists of problems whose running time is upper bounded by the function F_{ϵ_0} composed with any primitive recursive function (See [24]). Theorem 19 then follows.

Determinization of One-Counter Nets

Shaull Almagor  

Department of Computer Science, Technion – Israel Institute of Technology, Haifa, Israel

Asaf Yeshurun  

Department of Computer Science, Technion – Israel Institute of Technology, Haifa, Israel

Abstract

One-Counter Nets (OCNs) are finite-state automata equipped with a counter that is not allowed to become negative, but does not have zero tests. Their simplicity and close connection to various other models (e.g., VASS, Counter Machines and Pushdown Automata) make them an attractive model for studying the border of decidability for the classical decision problems.

The deterministic fragment of OCNs (DOCNs) typically admits more tractable decision problems, and while these problems and the expressive power of DOCNs have been studied, the determinization problem, namely deciding whether an OCN admits an equivalent DOCN, has not received attention.

We introduce four notions of OCN determinizability, which arise naturally due to intricacies in the model, and specifically, the interpretation of the initial counter value. We show that in general, determinizability is undecidable under most notions, but over a singleton alphabet (i.e., 1 dimensional VASS) one definition becomes decidable, and the rest become trivial, in that there is always an equivalent DOCN.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Determinization, One-Counter Net, Vector Addition System, Automata, Semilinear

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.18

Related Version *Previous Version:* <https://arxiv.org/abs/2112.13716>

1 Introduction

One-Counter Nets (OCNs) are finite-state machines equipped with an integer counter that cannot decrease below zero and cannot be explicitly tested for zero.

OCNs are closely related to several computational models: they are a test-free syntactic restriction of One-Counter Automata – Minsky Machines with only one counter. If counter updates are restricted to ± 1 , they are equivalent to Pushdown Automata with a single-letter stack alphabet. In addition, over a singleton alphabet, they are the same as 1-dimensional Vector Addition Systems with States.

An OCN \mathcal{A} over alphabet Σ accepts a word $w \in \Sigma^*$ from initial counter value $c \in \mathbb{N}$ if there is a run of \mathcal{A} on w from an initial state to an accepting state in which the counter, starting from value c , does not become negative. Thus, for every counter value $c \in \mathbb{N}$ the OCN \mathcal{A} defines a language $\mathcal{L}(\mathcal{A}, c) \subseteq \Sigma^*$.

OCNs are an attractive model for studying the border of decidability of classical decision problems. Indeed – several problems for them lie delicately close to the decidability border. For example, OCN universality is decidable [16], whereas parameterized-universality (in which the initial counter is existentially quantified) is undecidable [2].

As is the case with many computational models, certain decision problems for deterministic OCNs (OCNs that admit a single legal transition for each state q and letter σ), denoted DOCNs, are computationally easier than for nondeterministic OCNs (e.g., inclusion is undecidable for OCNs, but is in NL for DOCNs [16]. Universality is Ackermannian for OCNs, but is in NC for DOCNs [2]). While decision problems for DOCNs have received some attention, the *determinization* problem for OCNs, namely deciding whether an OCN admits



© Shaull Almagor and Asaf Yeshurun;

licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 18; pp. 18:1–18:23

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

an equivalent DOCN, has (to our knowledge) not been studied. Apart from the theoretical interest of OCN determinization, which would yield a better understanding of the model, it is also of practical interest: OCNs can be used to model properties of concurrent systems, so when an OCN can be determinized, automatic reasoning about correctness becomes easier.

OCN Determinization

Recall that the language $\mathcal{L}(\mathcal{A}, c)$ of an OCN \mathcal{A} depends on the initial counter c , so \mathcal{A} essentially defines a family of languages. Thus, it is not clear what we mean by “equivalent DOCN”. We argue that the definition of determinization depends on the role of the initial counter c . To this end, we identify four notions of determinization for an OCN \mathcal{A} , as follows.

- In **0-Det**, we ask whether there is a DOCN \mathcal{D} such that $\mathcal{L}(\mathcal{A}, 0) = \mathcal{L}(\mathcal{D}, 0)$.
- In **\exists -Det**, we ask whether there exist $c \in \mathbb{N}$ and a DOCN \mathcal{D} such that $\mathcal{L}(\mathcal{A}, c) = \mathcal{L}(\mathcal{D}, 0)$.
- In **\forall -Det**, we ask whether for every $c \in \mathbb{N}$, there is a DOCN \mathcal{D} such that $\mathcal{L}(\mathcal{A}, c) = \mathcal{L}(\mathcal{D}, 0)$.
- In **Uniform-Det**, we ask whether there is a DOCN \mathcal{D} such that for every $c \in \mathbb{N}$ we have $\mathcal{L}(\mathcal{A}, c) = \mathcal{L}(\mathcal{D}, c)$.

The motivation for each of the problems depends, intuitively, on the interpretation of the initial counter, and on the stage at which the equivalent DOCN is computed, as we now demonstrate.

- Consider an OCN modelling an access-control handler, where the counter corresponds to the number of access requests in a queue. Since the controller is deployed with an empty queue, an equivalent DOCN would need to be equivalent only on an initial 0 counter, so we would want to solve **0-Det**.
- Consider an OCN modelling resource handler, where the counter corresponds to the available resources. When searching for a deterministic controller, we may initialize it with some fixed amount of resources to start with, hence **\exists -Det** is suitable.
- Now consider the task of devising an OCN for the resource handler above, so that it can be deployed in many different concrete settings as a DOCN, but where each setting has its own amount of available initial resources. In order to design a single OCN that can be determinized to appropriate DOCNs, we would want to solve **\forall -Det**.
- Finally, **Uniform-Det** is of interest in any setting that is exactly modelled as an OCN, but needs to be determinized, e.g., when the resource handler above needs to be deployed but the initial resources depend on the system’s load.

Paper Organization and Contribution

In this paper, we study the decidability of the determinization problems derived from the four notions. In Section 3 we examine the relation between the notions, and demonstrate that no pair of them coincide. In Section 4 we show that **0-Det**, **\exists -Det**, and **\forall -Det** are generally undecidable. For **Uniform-Det**, we are not able to resolve decidability, but we do show an Ackermannian lower bound.

In order to recover some decidability, we turn to the fragment of OCNs over a singleton alphabet (1-dimensional VASS). There, we show that **0-Det**, **\exists -Det**, and **\forall -Det** become trivial (i.e., they always hold), whereas **Uniform-Det** becomes decidable. We conclude with a discussion and future work in Section 6.

Technically, our undecidability results for general alphabets use reductions from two different models – one from the model of Lossy Counter Machines [23, 27], and one from a careful analysis of recent results about OCNs [2]. For the singleton-alphabet case, the decidability of **Uniform-Det** requires some machinery from the theory of low-dimensional

VASS and Presburger Arithmetic, as well as some basic linear algebra and number theory. Our main contribution in this part is the introduction and analysis of the *Minimal Counter Relation (MCR)* – a sequence characterizing the minimal counter needed to accept each length of words. We characterize **Uniform-Det** using this sequence, and we suspect this sequence may prove useful in other contexts.

Related Work

The determinization problem we consider in this work assumes that the deterministic target model is also that of OCNs. An alternative approach to simplifying a nondeterministic OCN is to find an equivalent deterministic finite automaton, if one exists. This amounts to deciding whether the language of an OCN is regular. This problem was shown to be undecidable for OCNs in [32]. Interestingly, the related problem of regular separability was shown to be in PSPACE in [10]. A related result in [11] describes a determinization procedure for “unambiguous blind counter automata” over infinite words, to a Muller counter machine.

From a different viewpoint, determinization is a central problem in quantitative models, which can be thought of as counter automata where the counter value is the output, rather than a Boolean language acceptor. The decidability of determinization for Tropical Weighted Automata is famously open [9, 20] with only partial decidable fragments [20, 21]. A slightly less related model is that of discounted-sum automata, whose determinization has intricate connections to number theory [7].

Determinization of computational models is closely related to various notions of semantic equivalence. The three main concepts scrutinized in this regard are, from most restrictive to least restrictive: bisimulation, simulation and trace inclusion. Each of the three notions has strong and weak variants. Strong bisimulation was shown to be PSPACE-complete both for OCNs and OCAs [5, 6], while weak bisimulation was shown to be undecidable [23]. Conversely, trace inclusion, both weak and strong, is undecidable both for OCNs and OCAs [15, 31]. Finally, simulation, both weak and strong, is undecidable for OCAs [17], but decidable for OCNs [1, 18, 19, 28, 30].

2 Preliminaries

A *one-counter net* (OCN) is a finite automaton whose transitions are labelled both by letters and by integer weights. Formally, an OCN is a tuple $\mathcal{A} = \langle \Sigma, Q, s_0, \delta, F \rangle$ where Σ is a finite alphabet, Q is a finite set of states, $s_0 \in Q$ is the initial state, $\delta \subseteq Q \times \Sigma \times \mathbb{Z} \times Q$ is the set of transitions, and $F \subseteq Q$ are the accepting states. We say that an OCN is *deterministic* if for every $s \in Q, \sigma \in \Sigma$, there is at most one transition (s, σ, e, s') for some $e \in \mathbb{Z}$ and $s' \in Q$.

For a transition $t = (s, \sigma, e, s') \in \delta$ we define $\text{eff}(t) = e$ to be its (counter) *effect*.

A *path* in the OCN is a sequence $\pi = (s_1, \sigma_1, e_1, s_2)(s_2, \sigma_2, e_2, s_3) \dots (s_k, \sigma_k, e_k, s_{k+1}) \in \delta^*$. 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 π *reads* the word $\sigma_1 \sigma_2 \dots \sigma_k \in \Sigma^*$. The *effect* of π is $\text{eff}(\pi) = \sum_{i=1}^k e_i$, and its *nadir*, denoted $\text{nadir}(\pi)$, is the minimal effect of any prefix of π (note that the nadir is non-positive, since $\text{eff}(\epsilon) = 0$).

A *configuration* of an OCN is a pair $(s, v) \in Q \times \mathbb{N}$ comprising a state and a non-negative integer. For a letter $\sigma \in \Sigma$ and configurations $(s, v), (s', v')$ we write $(s, v) \xrightarrow{\sigma} (s', v')$ if there exists $d \in \mathbb{Z}$ such that $v' = v + d$ and $(s, \sigma, d, s') \in \delta$.

A *run* of \mathcal{A} from initial counter c on a word $w = \sigma_1 \dots \sigma_k \in \Sigma^*$ is a sequence of configurations $\rho = (q_0, v_0), (s_1, v_1), \dots, (s_k, v_k)$ such that $v_0 = c$ and for every $1 \leq i \leq k$ it holds that $(s_{i-1}, v_{i-1}) \xrightarrow{\sigma_i} (s_i, v_i)$. Since configurations may only have a non-negative counter, this enforces that the counter does not become negative.

18:4 Determinization of One-Counter Nets

Note that every run naturally induces a path in the OCN. For the converse, a path π induces a run from initial counter c iff $c \geq -\text{nadir}(\pi)$ (indeed, the minimal initial counter needed for traversing a path π is exactly $-\text{nadir}(\pi)$). We extend the definitions of effect and nadir to runs, by associating them with the corresponding path.

The run ρ is *accepting* if $s_k \in F$, and we say that \mathcal{A} *accepts* w with initial counter c if there exists an accepting run of \mathcal{A} on w from initial counter c . We define $\mathcal{L}(\mathcal{A}, c) = \{w \in \Sigma^* : \mathcal{A} \text{ accepts } w \text{ with initial counter } c\}$, and we define the complement of a language $\mathcal{L}(\mathcal{A}, c)$ to be $\overline{\mathcal{L}(\mathcal{A}, c)} = \Sigma^* \setminus \mathcal{L}(\mathcal{A}, c)$. Observe that OCNs are monotonic – if \mathcal{A} accepts w from counter c , it also accepts it from every $c' \geq c$. Thus, $\mathcal{L}(\mathcal{A}, c) \subseteq \mathcal{L}(\mathcal{A}, c')$ for $c' \geq c$.

3 OCN Determinization

In this section we examine the relationship between the four determinization notions. For brevity, we use the same term for the decision problems and the properties they represent, e.g., we say “ \mathcal{A} is 0-Det” if \mathcal{A} has an equivalent DOCN under 0-Det.

We first examine how the definitions compare in their strictness:

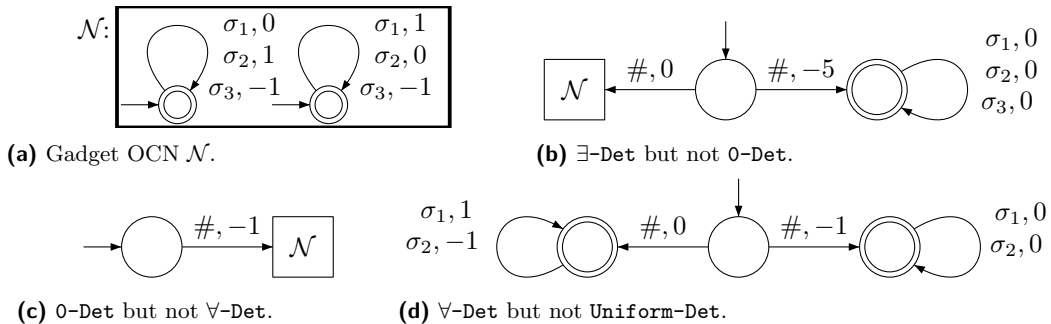
► **Observation 1.** *Consider an OCN \mathcal{A} . If \mathcal{A} is Uniform-Det, then \mathcal{A} is \forall -Det, if \mathcal{A} is \forall -Det, then \mathcal{A} is 0-Det, and if \mathcal{A} is 0-Det, then \mathcal{A} is \exists -Det.*

Next, we prove that none of the definitions coincide. Following Observation 1, it suffices to prove the following.

► **Lemma 2.** *There exist OCNs $\mathcal{A}, \mathcal{B}, \mathcal{C}$ such that \mathcal{A} is \exists -Det but not 0-Det, \mathcal{B} is 0-Det but not \forall -Det, and \mathcal{C} is \forall -Det but not Uniform-Det.*

Proof (sketch). The OCNs $\mathcal{A}, \mathcal{B}, \mathcal{C}$ are depicted in Figure 1. We demonstrate the intuition on \mathcal{C} , see Appendix A.1 for the complete proof. To show that \mathcal{C} is \forall -Det, we observe that for initial counter 0, we can omit the $(\#, -1)$ transition, thus obtaining an equivalent DOCN. For initial counter $c \geq 1$ we have that $\mathcal{L}(\mathcal{C}, c) = \# \cdot \{\sigma_1, \sigma_2\}^*$ is regular and thus has a DOCN.

We claim \mathcal{C} is not Uniform-Det. An equivalent DOCN \mathcal{D} with k states, starting from initial counter 0, must accept the word $\#\sigma_1^{k+1}\sigma_2^{k+1}$. It is easy to show that upon reading σ_2^{k+1} it must make a negative cycle. This, however, causes some word of the form $\#\sigma_1^{k+1}\sigma_2^m$ not to be accepted even with counter 1, which means $\mathcal{L}(\mathcal{D}, 1) \neq \mathcal{L}(\mathcal{C}, 1)$. ◀



■ **Figure 1** Examples separating the determinization notions.

4 Lower Bounds for Determinization

In this section we prove lower bounds for the four determinization decision problems. We show that **0-Det**, **\forall -Det**, and **\exists -Det** are undecidable, while for **Uniform-Det** we show an Ackermannian lower bound, and its decidability remains an open problem.

We start by introducing *Lossy Counter Machines* (LCMs) [23, 27], from which we will obtain some undecidability results. Intuitively, an LCM is a Minsky counter machine, whose semantics are such that counters may arbitrarily decrease at each step. Formally, an LCM is $\mathcal{M} = \langle \text{Loc}, \mathbf{Z}, \Delta \rangle$ where $\text{Loc} = \{\ell_1, \dots, \ell_m\}$ is a finite set of *locations*, $\mathbf{Z} = (z_1, \dots, z_n)$ are n counters, and $\Delta \subseteq \text{Loc} \times \text{OP}(\mathbf{Z}) \times \text{Loc}$, where $\text{OP}(\mathbf{Z}) = \mathbf{Z} \times \{++, --, = 0?\}$.

A *configuration* of \mathcal{M} is $\langle \ell, \mathbf{a} \rangle$ where $\ell \in \text{Loc}$ and $\mathbf{a} = (a_1, \dots, a_n) \in \mathbb{N}^{\mathbf{Z}}$. There is a *transition* $\langle \ell, \mathbf{a} \rangle \rightarrow \langle \ell', \mathbf{b} \rangle$ if there exists $\text{op} \in \text{OP}$ and either:

- $\text{op} = c_k++$ and $b_k \leq a_k + 1$ and $b_j \leq a_j$ for all $j \neq k$, or
- $\text{op} = c_k--$ and $b_k \leq a_k - 1$ and $b_j \leq a_j$ for all $j \neq k$, or
- $\text{op} = c_k = 0?$ and $b_k = a_k = 0$ and $b_j \leq a_j$ for all $j \neq k$.

Since we only require \leq on the counter updates, the counters nondeterministically decrease at each step.

A *run* of \mathcal{M} is a finite sequence $\langle \ell_1, \mathbf{a}_1 \rangle \rightarrow \langle \ell_2, \mathbf{a}_2 \rangle \rightarrow \dots \rightarrow \langle \ell_r, \mathbf{a}_r \rangle$. Given a configuration $\langle \ell, \mathbf{a} \rangle$, the *reachability set* of $\langle \ell, \mathbf{a} \rangle$ is the set of all configurations reachable from $\langle \ell, \mathbf{a} \rangle$ via runs of \mathcal{M} . In [27], it is shown that the problem of deciding whether the reachability set of a configuration is finite, is undecidable. A slight modification of this problem (see Appendix A.2) yields the following.

► **Lemma 3.** *The following problem, dubbed 0-FINITE-REACH, is undecidable: Given an LCM \mathcal{M} and a location ℓ_0 , decide whether the reachability set of $\langle \ell_0, (0, \dots, 0) \rangle$ is finite.*

4.1 Undecidability of 0-Det

We show that **0-Det** is undecidable using a reduction from 0-FINITE-REACH. Intuitively, given an LCM \mathcal{M} and a location ℓ_0 , we construct an OCN \mathcal{A} that accepts, from initial counter 0, all the words that do not represent runs of \mathcal{M} from $\langle \ell_0, (0, \dots, 0) \rangle$.

In order for the OCN \mathcal{A} to verify the illegality of a run, it guesses a violation in it. Control violations, i.e., illegal transitions between locations, are easily checked. In order to capture counter violations, \mathcal{A} must find a counter whose value in the current configuration is *smaller* than in the next iteration (up to ± 1 for $++$ and $--$ commands). This, however, cannot be done by an OCN, since intuitively an OCN can only check that the later number is smaller, by first incrementing the counter and then decrementing it. To overcome this, we encode runs in reverse, as follows.

Consider an LCM $\mathcal{M} = \langle \text{Loc}, \mathbf{Z}, \Delta \rangle$ with $\text{Loc} = \{\ell_1, \dots, \ell_m\}$ and $\mathbf{Z} = (z_1, \dots, z_n)$. We encode a configuration $\langle \ell, (a_1, \dots, a_n) \rangle$ over the alphabet $\Sigma = \text{Loc} \cup \mathbf{Z}$ as $\ell \cdot z_1^{a_1} \dots z_n^{a_n} \in \Sigma^*$. We then encode a run by concatenating the encoding of its configurations.

For a word $w = \sigma_1 \dots \sigma_k \in \Sigma^*$, let $w^R = \sigma_k \dots \sigma_1$ be its *reverse*, and for a language $L \subseteq \Sigma^*$, define its reverse to be $L^R = \{w^R : w \in L\}$.

We now define $L_{\mathcal{M}, \ell_0} = \{w \in \Sigma^* : w \text{ encodes a run of } \mathcal{M} \text{ from } \langle \ell_0, (0, \dots, 0) \rangle\}$.

We are now ready to describe the construction of \mathcal{A} .

► **Lemma 4.** *Given an LCM \mathcal{M} and a location ℓ_0 , we can construct an OCN \mathcal{A} such that $\mathcal{L}(\mathcal{A}, 0) = \overline{L_{\mathcal{M}, \ell_0}^R}$.*

Proof sketch: We construct \mathcal{A} such that it accepts a word w iff w^R does not describe a run of \mathcal{M} from $\langle \ell_0, (0, \dots, 0) \rangle$.

As mentioned above, \mathcal{A} reads w and guesses when a violation would occur, where control violations are relatively simple to spot, by directly encoding the structure of \mathcal{M} in \mathcal{A} .

In order to spot counter violations, namely two consecutive configurations $\langle \ell, (a_1, \dots, a_n) \rangle$ and $\langle \ell', (a'_1, \dots, a'_n) \rangle$ such that some a'_i is too large compared to its counterpart a_i (how much larger is “too large” depends on \mathcal{M} 's transitions), \mathcal{A} reads a configuration $\ell \cdot z_1^{a_1} \dots z_n^{a_n}$ and increments its counter to count up to a_i , if it guesses that z_i is the counter that violates the transition. Assume for simplicity that the command in the transition does not involve counter z_i , then upon reading the next configuration $\ell' \cdot z_1^{b_1} \dots z_n^{b_n}$, \mathcal{A} decrements its counter while reading z_i , so that the counter value is $a_i - b_i$. Then, \mathcal{A} takes another transition with counter value -1 . Since the configuration is reversed, if this is indeed a violation, then $a_i > b_i$ (since the counters are lossy), so $a_i - b_i - 1 \geq 0$, and \mathcal{A} accepts. Otherwise, $a_i \leq b_i$, so this run of \mathcal{A} cannot proceed.

In Appendix A.3 we give the complete details of the construction. \blacktriangleleft

The correctness of the construction is proved in the following lemma.

► Lemma 5. *Consider an LCM \mathcal{M} and a location ℓ_0 , and let \mathcal{A} be the OCN constructed in Lemma 4. Then (\mathcal{M}, ℓ_0) is in 0-FINITE-REACH iff \mathcal{A} is 0-Det.*

Proof sketch: Assume the reachability set of $\langle \ell_0, (0 \dots 0) \rangle$ is finite under \mathcal{M} . Then there exists an upper bound $M \in \mathbb{N}$ of all counter values in all legal runs of \mathcal{M} from $\langle \ell_0, (0 \dots 0) \rangle$. \mathcal{A} 's behaviour can then be fully captured by a DFA \mathcal{D} with the set of all states of the form $\langle \ell, a_1 \dots a_k, b_1 \dots b_k \rangle$ such that ℓ is a state in \mathcal{M} , k the number of counters, the values of $a_1 \dots a_k$ represent counter values of the “current” configuration already fully known, and the values of $b_1 \dots b_k$ represent counter values of the “previous” configuration, that \mathcal{D} is in the process of accumulating. In addition, all values of $a_1 \dots a_k, b_1 \dots b_k$ are bounded by M . by assigning the only accepting state of \mathcal{D} as $q_f = \langle \ell_0, 0 \dots 0, 0 \dots 0 \rangle$, and addressing several minor technical nuances, we can conclude $\mathcal{L}(\mathcal{D}) = \mathcal{L}(\mathcal{A}, 0)$, therefore both $\mathcal{L}(\mathcal{A}, 0)$ and $\mathcal{L}(\mathcal{A}, 0)$ are regular. Specifically, \mathcal{A} is 0-Det.

As for the other direction, assume the reachability set of $\langle \ell_0, (0 \dots 0) \rangle$ under \mathcal{M} is infinite, and assume by way of contradiction that \mathcal{A} has a deterministic equivalent \mathcal{D}' . Note that for every word $u \in \Sigma^*$, the run of \mathcal{D}' does not end due to the counter becoming negative, since we can always concatenate some $\lambda \in \Sigma^*$ such that $u\lambda$ does not correspond to a run, and is hence accepted by \mathcal{D}' , so the run on u must be able to continue reading λ .

Since the reachability set of $\langle \ell_0, (0 \dots 0) \rangle$ is infinite, there exists a counter of \mathcal{M} , w.l.o.g z_1 , that can take unbounded values (in different runs). Let w be a word corresponding to a run of \mathcal{M} that ends with the value of z_1 being N for some large N . We can then write $w = a_k^* \dots a_1^N \ell a_k^* \dots a_1^{N'} \ell' \rho$, such that ρ represents the reverse of a legal prefix of a run of \mathcal{M} . \mathcal{D}' necessarily goes through a cycle β when reading a_1^N . We pump the cycle k times until the word obtained, w' , represents an illegal run due to the difference between $N + k \cdot |\beta|$ and N' . w' should then be accepted, but is in fact rejected, either due to the run ending successfully in the same non accepting state as w , or halting ahead of time due to a counter violation. Either way, that is a contradiction.

In Appendix A.4 we give the formal construction of \mathcal{D} , and a detailed correctness proof. \blacktriangleleft

Combining Lemmas 4 and 5, we conclude the following.

► Theorem 6. *0-Det is undecidable for OCNs over a general alphabet.*

4.2 Undecidability of \forall -Det and of \exists -Det

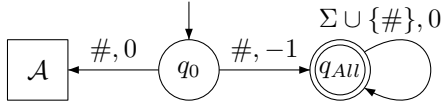
The undecidability of \forall -Det follows from that of 0-Det.

► **Theorem 7.** \forall -Det is undecidable.

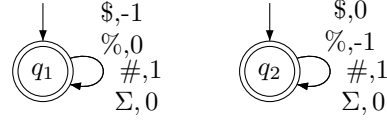
Proof. We show a reduction from 0-Det. Given an OCN $\mathcal{A} = \langle \Sigma, Q, s_0, \delta, F \rangle$, we construct an OCN $\mathcal{B} = \langle \Sigma', Q', q_0, \delta', F' \rangle$ as illustrated in figure 2. Formally, the states of \mathcal{B} are $Q' = Q \cup \{q_0, q_{All}\}$, the initial state is q_0 , its alphabet is $\Sigma' = \Sigma \cup \{\#\}$ such that $\# \notin \Sigma$, its accepting states are $F' = F \cup \{q_{All}\}$, and its transition relation is $\Delta' = \Delta \cup \{(q_0, \#, -1, q_{All}), (q_0, \#, 0, s_0)\} \cup \{(q_{All}, \sigma, 0, q_{All}) : \sigma \in \Sigma'\}$.

We claim that \mathcal{A} is 0-Det iff \mathcal{B} is \forall -Det. For the first direction, assume \mathcal{A} is 0-Det. Thus, $\mathcal{L}(\mathcal{B}, 0) = \# \cdot \mathcal{L}(\mathcal{A}, 0)$ has an equivalent DOCN. Since $\mathcal{L}(\mathcal{B}, k) = \#\Sigma'^*$ (which has a DOCN) for all $k \geq 1$, \mathcal{B} is \forall -Det.

Conversely, assume \mathcal{A} is not 0-Det. Since the transition $(q_0, \#, -1, q_{All})$ cannot be taken by \mathcal{B} with initial counter value 0, $\mathcal{L}(\mathcal{B}, 0) = \{\#w\}_{w \in \mathcal{L}(\mathcal{A}, 0)}$, hence \mathcal{B} is not 0-Det (since a DOCN for $\mathcal{L}(\mathcal{B}, 0)$ would easily imply a DOCN for $\mathcal{L}(\mathcal{A}, 0)$). Thus, \mathcal{B} is not \forall -Det. ◀



■ **Figure 2** The OCN \mathcal{B} in Theorem 7.



■ **Figure 3** Gadget OCN \mathcal{G} for Theorem 9.

Next, we show the undecidability of \exists -Det with a reduction from the halting problem for two-counter (Minsky) machines (2CM). Technically, we rely on a construction from [2], which reduces the latter problem to the “parameterized universality” problem for OCN. For our purpose, the reader need not be familiar with Minsky Machines, as it suffices to know that their halting problem is undecidable [24]. We start the reduction with the following property.

► **Theorem 8** ([2]). *Given a 2CM \mathcal{M} , we can construct an OCN \mathcal{B} over alphabet $\Sigma \cup \{\#\}$ with $\# \notin \Sigma$ such that the following holds:*

- *If \mathcal{M} halts, there exists $c \in \mathbb{N}$ such that $\mathcal{L}(\mathcal{B}, c) = \Sigma^*$,*
- *If \mathcal{M} does not halt, then for every $c \in \mathbb{N}$ there exists a word $w_c \in (\Sigma \cup \{\#\})^*$ such that every run of \mathcal{B} on w_c enters a state from which reading any word of the form $\#^*$ does not lead to an accepting state.*

We can now proceed with the reduction to \exists -Det.

► **Theorem 9.** \exists -Det is undecidable.

Proof. We reduce the halting problem for 2CM to \exists -Det. Given a 2CM \mathcal{M} , we start by constructing the OCN \mathcal{B} as per Theorem 8. We augment \mathcal{B} to work over the alphabet $\Sigma' = \Sigma \cup \{\#, \$, \%\}$, where $\$, \% \notin \Sigma$, by fixing the behaviour of $\$$ and $\%$ to be identical to $\#$.

Next, consider the gadget OCN \mathcal{G} depicted in Figure 3. A similar argument to the proof of Lemma 2 (specifically, Figure 1a), shows that \mathcal{G} does not have an equivalent DOCN for any initial counter c .

We now obtain a new OCN \mathcal{A} by taking the union of \mathcal{B} and \mathcal{G} (i.e. placing them “side by side”). We claim that \mathcal{M} halts iff \mathcal{A} is \exists -Det.

If \mathcal{M} halts, by Theorem 8 there exists an initial counter c such that $\mathcal{L}(\mathcal{B}, c) = \{\Sigma \cup \{\#\}\}^*$. Since in \mathcal{B} the letters $\$$ and $\%$ behave like $\#$, we have that $\mathcal{L}(\mathcal{A}, c) = \Sigma'^*$, so \mathcal{A} is \exists -Det.

If M does not halt, then again by Theorem 8, for every $c \in \mathbb{N}$ there exists a word w_c such that $w_c \notin \mathcal{L}(\mathcal{B}, c)$, and such every run of \mathcal{B} on w_c enters a state from which reading $\#^*$ (and hence any word from $\{\#, \% \$\}^*$) does not lead to an accepting state. Now assume by way of contradiction that \mathcal{A} has a deterministic equivalent \mathcal{D} with k states for initial counter c . \mathcal{A} accepts w_c with the runs of \mathcal{G} , since w_c does not contain $\$$ or $\%$. Thus, \mathcal{D} accepts w_c with initial counter 0. In addition, \mathcal{A} , and therefore \mathcal{D} , both accept $w'_c = w_c \#^{k+1-j} \%^{k+1} \$^{k+1}$ where j is the number of occurrences of $\#$'s in w_c . Using the fact that \mathcal{G} does not have an equivalent DOCN, we can now reach a contradiction with similar arguments as the proof of Lemma 2 (Figure 1a). ◀

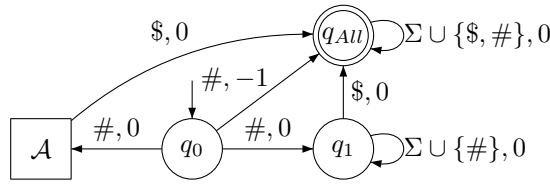
4.3 A Lower Bound for Uniform-Det

Unfortunately, as of yet we are unable to resolve the decidability of **Uniform-Det**. In this section, we show that **Uniform-Det** is Ackermann-hard, and in particular non primitive recursive.

► **Theorem 10.** *Uniform-Det is Ackermann-hard.*

Proof. We show a reduction from the OCN universality problem with initial counter 0, shown to be Ackermann-hard in [16].

Consider an OCN $\mathcal{A} = \langle \Sigma, Q, s_0, \delta, F \rangle$. We construct an OCN $\mathcal{B} = \langle \Sigma', Q', q_0, \delta', F' \rangle$ as depicted in Figure 4 (for $\#, \$ \notin \Sigma$).



■ **Figure 4** The OCN \mathcal{B} in the proof of Theorem 10.

We claim that $\mathcal{L}(\mathcal{A}, 0) = \Sigma^*$ iff \mathcal{B} is **Uniform-Det**.

Assume $\mathcal{L}(\mathcal{A}, 0) = \Sigma^*$, then $\mathcal{L}(\mathcal{B}, c) = \{\#w : w \in \Sigma'^*\}$ for every counter value c . Indeed, every word starting with $\#$ can be accepted by \mathcal{B} with initial counter value 0 either through \mathcal{A} , if it does not contain $\$$, or in q_{All} if it does. However, every word not starting with $\#$ cannot be accepted by \mathcal{B} for any initial counter value. Thus, \mathcal{B} is **Uniform-Det**.

Conversely, if $\mathcal{L}(\mathcal{A}, 0) \neq \Sigma^*$, let $w \notin \mathcal{L}(\mathcal{A}, 0)$. Assume by way of contradiction that there exists a deterministic OCN \mathcal{D} that is uniform-equivalent to \mathcal{B} .

$\#w \notin \mathcal{L}(\mathcal{B}, 0)$, so $\#w \notin \mathcal{L}(\mathcal{D}, 0)$. Moreover, the run of \mathcal{D} on $\#w$ cannot end in a non-accepting state, since $\#w \in \mathcal{L}(\mathcal{B}, 1) = \mathcal{L}(\mathcal{D}, 1)$. Thus, the run of \mathcal{D} on $\#w$ terminates due to the counter becoming negative. However, this is a contradiction, since $\#w\$ \in \mathcal{L}(\mathcal{B}, 0) = \mathcal{L}(\mathcal{D}, 0)$. We conclude that \mathcal{B} is not **Uniform-Det**. ◀

5 Singleton Alphabet

We now turn to study OCNs over a singleton alphabet denoted $\Sigma = \{\sigma\}$ throughout.

We start by briefly introducing Presburger Arithmetic (PA) [13, 26]. We refer the reader to [13] for a detailed survey. PA is the first-order theory of integers with addition and order $\text{FO}(\mathbb{Z}, 0, 1, +, <)$, and it is a decidable logic.

There is an important connection between PA and semilinear sets: for a *basis vector* $\mathbf{b} \in \mathbb{Z}^d$ and a set of *periods* $P = \{\mathbf{p}_1 \dots \mathbf{p}_k\} \subseteq \mathbb{Z}^d$, we define the *linear set* $\text{Lin}(\mathbf{b}, P) = \{\mathbf{b} + \lambda_1 \mathbf{p}_1 + \dots + \lambda_k \mathbf{p}_k : \lambda_i \in \mathbb{N} \text{ for all } 1 \leq i \leq k\}$. Then, a *semilinear set* is a finite union of linear sets.

A fundamental theorem about PA [12] shows that that for every PA formula $\varphi(\mathbf{x})$ with free variables \mathbf{x} , the set $\llbracket \varphi \rrbracket = \{\mathbf{a} : \mathbf{a} \models \varphi(\mathbf{x})\}$ is semilinear, and the converse also holds – every semilinear set is PA-definable.

Consider an OCN \mathcal{A} over $\Sigma = \{\sigma\}$. For every word σ^n , either σ^n is not accepted by \mathcal{A} for any counter value, or there exists a minimal counter value c such that $\sigma^n \in \mathcal{L}(\mathcal{A}, c')$ iff $c' \geq c$. We can therefore fully characterize the language of \mathcal{A} on any counter value using the *Minimal Counter Relation*¹ (MCR), defined as

$$\text{MCR}(\mathcal{A}) = \{(n, c) \subseteq \mathbb{N}^2, c \text{ is the minimal integer such that } \sigma^n \in \mathcal{L}(\mathcal{A}, c)\}.$$

We start by showing that $\text{MCR}(\mathcal{A})$ is semilinear.

► **Lemma 11.** *Consider an OCN \mathcal{A} over $\Sigma = \{\sigma\}$, then $\text{MCR}(\mathcal{A})$ is effectively semilinear.*

Proof. We prove the claim using well-known and deep results about low-dimensional VASS. A 2D-VASS is (for our purposes²) identical to an OCN over $\Sigma = \{\sigma\}$, but has two counters (both need to be kept non-negative). Formally, a 2D VASS is $\mathcal{V} = \langle Q, s_0, \delta, F \rangle$, where $\delta \subseteq Q \times \mathbb{Z}^2 \times Q$. The semantics are similar to OCNs, acting separately on the two counters, as follows. A *configuration* of \mathcal{V} is $(q, (c_1, c_2))$ where $q \in Q$ and $(c_1, c_2) \in \mathbb{N}^2$ are the counter values, and a *run* is a sequence of configurations $(q_1, (c_1^1, c_2^1)), \dots, (q_k, (c_1^k, c_2^k))$ that follow according to δ , i.e., for every $1 \leq i < k$ we have that $(q_i, (c_1^{i+1} - c_1^i, c_2^{i+1} - c_2^i), q_{i+1}) \in \delta$. We denote $(q_1, (c_1^1, c_2^1)) \xrightarrow{\mathcal{V}} (q_k, (c_1^k, c_2^k))$ if such a run exists.

In [22], it is proved that given a 2D-VASS, we can effectively compute a PA formula $\psi_{\text{Reach}}(q, x_1, x_2, q', y_1, y_2)$ such that $\llbracket \psi_{\text{Reach}}(q, x_1, x_2, q', y_1, y_2) \rrbracket = \{(q, c_1, c_2, q', d_1, d_2) : (q, c_1, c_2) \xrightarrow{\mathcal{V}} (q', d_1, d_2)\}$ (the states q, q' are encoded as variables taking values in $\{1, \dots, |Q|\}$).

Observe that ψ_{Reach} does not encode information about the length of the run, whereas MCR does require it. On the other hand, ψ_{Reach} works for 2D-VASS, whereas we only need an OCN (i.e., 1D-VASS). We therefore proceed by first introducing the notion of Linear Path Schemes [22, 4]. Consider the transitions of \mathcal{A} as an alphabet (i.e., each transition $(q, \sigma, v, p) \in \delta$ is a letter). A *Linear Path Scheme* is a regular expression of the form $\rho = \alpha_0 \beta_1^* \alpha_1 \dots \beta_k^* \alpha_k$ where the α_i and β_i are words in δ^* , such that each α_i represents a path in \mathcal{A} , and each β_i represents a cycle. The *length* of ρ is defined as $|\alpha_1| + |\beta_1| + \dots + |\alpha_k| + |\beta_k|$, i.e., the length of the underlying path, excluding repetitions of the β_i .

The following result can be obtained from [4] by using 2D-VASS as a proxy, as we do in Lemma 11, or directly from [2].

► **Lemma 12.** *Let \mathcal{A} be an OCN over singleton alphabet, then there exists a finite set S of linear path schemes such that the following holds:*

1. Every $\rho \in S$ has length at most $2|Q|^2$.
2. For every two configurations $(p, c_1), (q, c_2) \in Q \times \mathbb{N}$ and every $n \in \mathbb{N}$, if there is a run of \mathcal{A} on σ^n from (p, c_1) to (q, c_2) , then there is such a run of the form $\rho \in S$.

¹ We remark that $\text{MCR}(\mathcal{A})$ is in fact the graph of a partial function. For convenience of working with PA, we stick with the relation notation.

² Usually, OCNs are defined as 1D-VASS, not the other way around.

18:10 Determinization of One-Counter Nets

As a consequence of Lemma 12, in order to decide if σ^n is accepted in \mathcal{A} , it is enough to consider runs that are linear path schemes of length at most $2|Q|^2$.

Consider a linear path scheme $\rho = \alpha_0\beta_1^* \alpha_1\beta_2^* \alpha_2 \cdots \alpha_k^*$. We now construct a formula $\varphi_\rho(n, c)$ which intuitively states that the word σ^n has a run of the form ρ starting with initial counter value c . This is defined as follows.

$$\varphi_\rho(n, c) := \exists e_1 \cdots e_k, \text{CORRECT-LENGTH}_\rho(n, e_1 \cdots e_k) \wedge \text{SUFFICIENT-COUNTER}_\rho(c, e_1 \cdots e_k)$$

Intuitively, $\varphi_\rho(n, c)$ states that there exist numbers $e_1 \cdots e_k$ such that the concrete run $\alpha_0\beta_1^{e_1} \alpha_1\beta_2^{e_2} \alpha_2 \cdots \alpha_k$ takes exactly n transitions, and that starting the run with initial counter c is sufficient to complete the run.

Formally, we define the sub-formulas as follows:

$$\blacksquare \text{CORRECT-LENGTH}_\rho(n, e_1 \cdots e_k) := |\alpha_0| + e_1 \cdot |\beta_1| + |\alpha_1| + \cdots + |\alpha_k| = n.$$

■

$$\text{SUFFICIENT-COUNTER}_\rho(c, e_1 \cdots e_k) :=$$

$$\bigwedge_{i=0}^k c + \text{eff}(\alpha_0) + e_1 \cdot \text{eff}(\beta_1) + \cdots + e_i \cdot \text{eff}(\beta_i) \geq \text{nadir}(\alpha_i)$$

$$\wedge \bigwedge_{i=1}^k (c + \text{eff}(\alpha_0\beta_1^{e_1} \cdots \alpha_{i-1}) \geq \text{nadir}(\beta_i) \wedge c + \text{eff}(\alpha_0\beta_1^{e_1} \cdots \alpha_{i-1}) + (e_i - 1)\text{eff}(\beta_i) \geq \text{nadir}(\beta_i^{e_i}))$$

The correctness of $\text{CORRECT-LENGTH}_\rho$ is obvious. The correctness of the formula $\text{SUFFICIENT-COUNTER}_\rho(c, e_1 \cdots e_k)$ is based on the observation that in order to traverse the cycle β for e times, the counter c must be enough to traverse β once, and must be enough so that $c + (e - 1)\text{eff}(\beta) \geq \text{nadir}(\beta)$, so that the “last” time can be traversed³. Indeed, if the counter becomes negative during some iteration of the cycle, it will be even “more” negative at the last iteration. See [4] for an analogous proof.

We can now readily obtain the formula $\theta(n, c)$ which captures $\text{MCR}(\mathcal{A})$ as follows: define $P \subseteq S$ to be the set of linear path schemes that start in q_0 and end in an accepting state, then

$$\theta(n, c) := \bigvee_{\rho \in P} \varphi_\rho(n, c) \wedge \forall c' < c, \bigwedge_{\rho \in P} \neg \varphi_\rho(n, c').$$

Indeed, $\theta(n, c)$ is satisfied iff there exists some linear path scheme $\rho \in P$ that can be traversed with length n and counter value c , and there is no smaller counter for which this holds.

Note that we can obtain $\theta(n, c)$ from \mathcal{A} in polynomial space, by generating all possible linear path schemes of length $2|Q|^2$ and constructing the respective subformulas. In particular, the length of $\theta(n, c)$ is single exponential in the description of \mathcal{A} . Moreover, $\theta(n, c)$ has two quantifier alternations – the disjunction is an existential formula, and the conjunction of negations can be viewed as a universal formula. Since quantifier alternation counting assumes starting with an existential quantifier, the universal formula is counted as two alternations. ◀

³ This argument assumes strictly positive exponents. This assumption is safe, since we can define a set S' that contains all linear path schemes obtained by possibly omitting any number of cycles in any of the linear path schemes in S . Every legal path in S can then be represented by a path in S' whose exponents are all strictly positive. By working with S' we then circumvent this issue. Note that $|S'|$ is still single-exponential in $|\mathcal{A}|$.

5.1 Decidability of Uniform-Det over Singleton Alphabet

In this subsection we prove that **Uniform-Det** is decidable for OCN over a singleton alphabet, and we can construct an equivalent DOCN, if one exists. Our characterization of **Uniform-Det** is based on its MCR, and specifically on two notions for subsets of \mathbb{N}^2 (applied to MCR). Consider a set $S \subseteq \mathbb{N}^2$. We say that S is *increasing* if it is the graph of an increasing partial function. That is, for every $(n_1, c_1), (n_2, c_2) \in S$, if $n_1 \leq n_2$ then $c_1 \leq c_2$, and if $n_1 = n_2$ then $c_1 = c_2$. Next, we say that S is (N, k, d) -*Ultimately Periodic* for $N, k, d \in \mathbb{N}$ if for every $n \geq N, (n, x) \in S$ iff $(n + k, x + d) \in S$. We say that S is (effectively) ultimately periodic if it is (N, k, d) -ultimately periodic for some (effectively computable) parameters $N, k, d \in \mathbb{N}$.

The main technical result of this section is the following.

► **Theorem 13.** *Consider an OCN \mathcal{A} over $\Sigma = \{\sigma\}$, then the following are equivalent:*

1. $MCR(\mathcal{A})$ is increasing.
2. $MCR(\mathcal{A})$ is increasing and effectively ultimately periodic.
3. \mathcal{A} is **Uniform-Det**, and we can effectively compute an equivalent DOCN.

We prove Theorem 13 in the remainder of this section. We start with a technical lemma concerning the implication $1 \implies 2$.

► **Lemma 14.** *Consider an effectively semilinear set $S \subseteq \mathbb{N}^2$. If S is increasing, then S is effectively periodic.*

Proof. Since S is effectively semilinear, then by [12] we can write $S = \bigcup_{i=1}^M \text{Lin}(\mathbf{b}_i, P_i)$ where $\mathbf{b}_i \in \mathbb{N}^2$ and $P_i \subseteq \mathbb{N}^2$ for every $1 \leq i \leq M$. Moreover, by [12, 13], we can assume that each P_i is a linearly-independent set of vectors.

All periods are singletons. We show that since S is increasing, then $|P_i| \leq 1$ for every $1 \leq i \leq M$. Assume $(n_1, c_1), (n_2, c_2) \in P_i$, and denote $\mathbf{b}_i = (a, b)$, then by the definition of a linear set, for every $\lambda_1, \lambda_2 \in \mathbb{N}$ we have that $(a, b) + \lambda_1(n_1, c_1) + \lambda_2(n_2, c_2) \in S$. Setting $\lambda_1 = 0$ and $\lambda_2 = n_1$, we have that $(a + n_1 n_2, b + n_1 c_2) \in S$, and setting $\lambda_1 = n_2$ and $\lambda_2 = 0$, we have that $(a + n_2 n_1, b + n_2 c_1) \in S$. Observe that $a + n_1 n_2 = a + n_2 n_1$, and since S is increasing, this implies $b + n_1 c_2 = b + n_2 c_1$, that is $n_1 c_2 = n_2 c_1$. It follows that $n_2(n_1, c_1) = n_1(n_2, c_2)$, but P_i is linearly independent, so it must hold that $(n_1, c_1) = (n_2, c_2)$, so $|P_i| \leq 1$.

Thus, we can in fact write $S = \bigcup_{i=1}^M \text{Lin}(\mathbf{b}_i, \{\mathbf{p}_i\})$ where $\mathbf{b}_i, \mathbf{p}_i \in \mathbb{N}^2$ (note that if $P_i = \emptyset$ we now take $\mathbf{p}_i = (0, 0)$). For every $1 \leq i \leq M$, denote $\mathbf{b}_i = (a_i, b_i)$ and $\mathbf{p}_i = (p_i, r_i)$.

All Periods have the same first component. We now claim that we can restrict all periods to have the same first component. That is, we can compute $\gamma \in \mathbb{N}$ and write $S = \bigcup_{j=1}^K \text{Lin}((\alpha_j, \beta_j), \{(\gamma, \eta_j)\})$.

Indeed, take $\gamma = \text{lcm}(\{p_i\}_{i=1}^M)$, we now “spread” each linear component $\text{Lin}((a_i, b_i), \{(p_i, r_i)\})$ by changing the period to $(\gamma, \frac{\gamma}{p_i} r_i)$, and compensating by adding additional linear sets with the same period and offset basis, to capture the “skipped” elements. In Appendix A.5 we describe the construction in general, and illustrate with an example.

All Periods are the same. Finally, we claim that we now have $\eta_i = \eta_j$ for every $1 \leq i, j \leq K$, so that in fact all the periods are the same vector (γ, η) . Indeed, Assume by way of contradiction that $\eta_j < \eta_i$ for some $1 \leq i, j \leq K$. Now, let $y \in \mathbb{N}$ be large enough so that $\alpha_i \leq \alpha_j + y \cdot \gamma$, and let $x \in \mathbb{N}$ be large enough so that (given y): $\beta_i + x \cdot \eta_i > \beta_j + y \cdot \eta_j + x \cdot \eta_j$.

18:12 Determinization of One-Counter Nets

We now have that $(\alpha_i, \beta_i) + x \cdot (\gamma, \eta_i) \in S$ and $(\alpha_j, \beta_j) + (x + y) \cdot (\gamma, \eta_j) \in S$, which contradicts S being increasing, since $\alpha_i \leq \alpha_j + y \cdot \gamma$ and therefore $\alpha_i + x \cdot \gamma \leq \alpha_j + (y + x) \cdot \gamma$, but also $\beta_i + x \cdot \eta_i > \beta_j + (y + x) \cdot \eta_j$.

Thus, we can now write $S = \bigcup_{j=1}^K \text{Lin}((\alpha_j, \beta_j), \{(\gamma, \eta)\})$

S is effectively ultimately periodic. Let $\alpha_{\max} = \max\{\alpha_j\}_{j=1}^K$, we claim that S is $(\alpha_{\max}, \gamma, \eta)$ -ultimately periodic. Let $n \geq \alpha_{\max}$, then $(n, c) \in S$ for some $c \in \mathbb{N}$ iff $(n, c) = (\alpha_i + \gamma \cdot m, \beta_i + \eta \cdot m)$ for some $1 \leq i \leq K$ and $m \in \mathbb{N}$. This happens iff $(n + \gamma, c + \eta) \in S$, since $(n + \gamma, c + \eta) = (\alpha_i + \gamma \cdot (m + 1), \beta_i + \eta \cdot (m + 1))$.

Finally, observe that all the constants in the proof are effectively computable. \blacktriangleleft

We now turn to the implication 2 \implies 3 of Theorem 13.

► Lemma 15. *Consider an OCN \mathcal{A} over $\Sigma = \{a\}$. If $\text{MCR}(\mathcal{A})$ is increasing and ultimately periodic, then \mathcal{A} is **Uniform-Det**, and we can effectively compute it.*

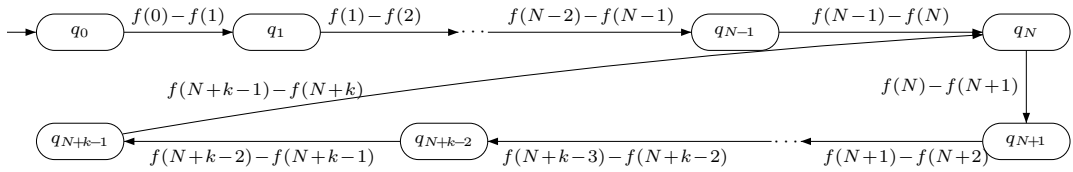
Proof. Assume $\text{MCR}(\mathcal{A})$ is (N, k, d) -ultimately periodic. We start by completing $\text{MCR}(\mathcal{A})$ to a (full) function $f : \mathbb{N} \rightarrow \mathbb{N}$ as follows: set $f(0) = 0$, and for $n > 0$ inductively define $f(n) = c$ if $(n, c) \in \text{MCR}(\mathcal{A})$, or $f(n) = f(n - 1)$ otherwise. That is, f matches $\text{MCR}(\mathcal{A})$ on its domain, and remains fixed between defined values. Observe that there is no violation in defining $f(0) = 0$, since if $(0, c) \in \text{MCR}(\mathcal{A})$, then $c = 0$, as the empty word requires a minimal counter of 0 to be accepted.

We now use f to obtain a DOCN \mathcal{D} as depicted in Figure 5. Formally, we construct $\mathcal{D} = \langle \{\sigma\}, Q, q_0, \delta, F \rangle$ as follows.

- $Q = \{q_i\}_{i=1}^{N+k-1}$.
- $\delta = \{(q_i, a, f(i) - f(i + 1), q_{i+1})\}_{i=1}^{N+k-2} \cup \{(q_{N+k-1}, a, f(N) + d - f(N + k - 1), q_N)\}$.
- $F = \{q_i : (i, f(i)) \in \text{MCR}(\mathcal{A}), 1 \leq i \leq N + k - 1\}$.

Observe that since f is increasing (as $\text{MCR}(\mathcal{A})$ is increasing), the weight of all transitions in \mathcal{D} is non-positive.

We claim that for every c , $\mathcal{L}(\mathcal{A}, c) = \mathcal{L}(\mathcal{D}, c)$. To show this, observe that for every $n \in \mathbb{N}$ we have that the sum of weights along n consecutive transitions of \mathcal{D} (ignoring the OCN semantics) is exactly $-f(n)$. In particular, if $\sigma^n \in \mathcal{L}(\mathcal{A}, c)$, then $(n, c') \in \text{MCR}(\mathcal{A})$ for some $c' \leq c$ and $f(n) = c'$. Indeed, this is trivial for $n \leq N + k - 1$, and for $n > N + k - 1$ this follows immediately from (N, k, d) -ultimate periodicity.



■ Figure 5 An illustration of the construction method for a uniform-deterministic-equivalent of an OCN \mathcal{A} , given f . Accepting states are not mentioned in the illustration.

Thus, if $\sigma^n \in \mathcal{L}(\mathcal{A}, c)$ then there exists $c' \leq c$ such that $(n, c') \in \text{MCR}(\mathcal{A})$ it follows that with initial counter c , \mathcal{D} can traverse n transitions. Moreover, the state reached is accepting, since $(n, c') \in \text{MCR}(\mathcal{A})$, so $\sigma^n \in \mathcal{L}(\mathcal{D}, c)$.

Conversely, if $\sigma^n \in \mathcal{L}(\mathcal{D}, c)$ then $c \geq f(n)$ and $(n, f(n)) \in \text{MCR}(\mathcal{A})$, thus, $\sigma^n \in \mathcal{L}(\mathcal{A}, c)$.

Finally, observe that the construction is computable given the parameters of ultimate periodicity. \blacktriangleleft

We now address the implication $3 \implies 1$.

► **Lemma 16.** *Consider an OCN \mathcal{A} over $\Sigma = \{\sigma\}$. If \mathcal{A} is *Uniform-Det*, then $\text{MCR}(\mathcal{A})$ is increasing.*

Proof. Let \mathcal{D} be a DOCN such that $\mathcal{L}(\mathcal{A}, c) = \mathcal{L}(\mathcal{D}, c)$ for every c , and let $(n_1, c_1), (n_2, c_2) \in \text{MCR}(\mathcal{A})$ with $n_1 \leq n_2$. Assume by way of contradiction that $c_1 > c_2$, then $\sigma^{n_2} \in \mathcal{L}(\mathcal{D}, c_2)$, but $\sigma^{n_1} \notin \mathcal{L}(\mathcal{D}, c_2)$. It follows that the run of \mathcal{D} on σ^{n_1} must end in a non-accepting state starting from counter value c_2 (i.e., the counter does not become negative). But then the same run is taken from counter value c_1 , so $\sigma^{n_1} \notin \mathcal{L}(\mathcal{D}, c_1)$, which is a contradiction. ◀

By Lemma 11, $\text{MCR}(\mathcal{A})$ is semilinear. Thus, if $\text{MCR}(\mathcal{A})$ is increasing, then by Lemma 14 it is also effectively ultimately periodic. This completes the implication $1 \implies 2$, and the implications $2 \implies 3$ and $3 \implies 1$ are immediate from Lemmas 15 and 16, respectively. This completes the proof of Theorem 13.

Finally, we can show the decidability of *Uniform-Det* by combining the characterization of Theorem 13 with the procedure of Lemma 11 and the decidability of PA [3].

► **Theorem 17.** *For OCNs over singleton alphabet, *Uniform-Det* is decidable. Moreover, it is in $3 - \text{EXPSpace}$.*

Proof. We start by showing the decidability of *Uniform-Det*. Consider an OCN \mathcal{A} . By Theorem 13, it suffices to show that it is decidable whether $\text{MCR}(\mathcal{A})$ is increasing. By Lemma 11, we can compute a PA formula $\theta(n, c)$ such that $\llbracket \theta \rrbracket = \text{MCR}(\mathcal{A})$. We now state the assertion that $\text{MCR}(\mathcal{A})$ is not increasing in PA as follows: $\chi = \exists n_1, n_2, c_1, c_2, n_1 < n_2 \wedge c_1 > c_2 \wedge \theta(n_1, c_1) \wedge \theta(n_2, c_2)$. Since PA is decidable, we can decide whether this sentence holds.

It remains to analyze the complexity of *Uniform-Det*. To this end, observe that in the proof of Lemma 11 we show that the length of $\theta(n, c)$ is single-exponential in $|\mathcal{A}|$ (and that we can obtain $\theta(n, c)$ from \mathcal{A} in polynomial space). Since PA is decidable in $2 - \text{EXPSpace}$ [3], we conclude that *Uniform-Det* is decidable in $3 - \text{EXPSpace}$. ◀

► **Remark 18 (On the $3 - \text{EXPSpace}$ upper bound).** It is easy to show that in fact $\theta(n, c)$ has at most 3 quantifier alternations. Therefore, the upper bound can be somewhat lowered using bounds for PA with fixed quantifier alternations [14]. However, applied to the exponential-length formula, these bounds do not get us as low as the next “major” complexity classes (e.g., $3 - \text{NEXPTIME}$, or $2 - \text{EXPSpace}$), so Theorem 17 is stated with $3 - \text{EXPSpace}$.

While we suspect this upper bound can be lowered, deciding whether $\text{MCR}(\mathcal{A})$ is increasing seems to be a hard problem. Indeed, $\text{MCR}(\mathcal{A})$ intuitively corresponds to the reachability relation of the OCN with two additional constraints: the length of the path is fixed, and the counter value is required to be minimal. The former constraint can be circumvented using 2D-VASS, as we do in Lemma 11, but the latter introduces a flavour of universal quantification. In particular, this poses a barrier to techniques attempting to reduce the behaviour of $\text{MCR}(\mathcal{A})$ to a reachability relation.

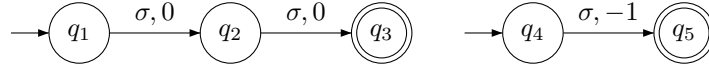
We proceed to give a lower bound on *Uniform-Det* (albeit far from the upper bound).

► **Theorem 19.** *For OCNs over singleton alphabet, *Uniform-Det* is *coNP-hard*.*

Proof. We show a reduction from the universality problem for NFAs over a singleton alphabet, which is *coNP-hard* [29].

We start by describing a gadget OCN \mathcal{B} as depicted in Figure 6. Note that \mathcal{B} is not *Uniform-Det*, since $\text{MCR}(\mathcal{B})$ is not increasing. Indeed, σ is only accepted with counter 1, whereas $\sigma\sigma$ is accepted with counter 0. Thus, $(1, 1), (2, 0) \in \text{MCR}(\mathcal{B})$.

18:14 Determinization of One-Counter Nets



■ **Figure 6** Gadget OCN \mathcal{B} for the reduction in Theorem 19.

We now describe the reduction. Consider an NFA $\mathcal{A} = \langle \{\sigma\}, Q, S_0, \delta, F \rangle$. We assume that \mathcal{A} is complete, i.e. that \mathcal{A} has a (not necessarily accepting) run on every word (if \mathcal{A} is not complete, we add a rejecting sink state as an initial state to \mathcal{A}).

We start by obtaining a new NFA $\mathcal{A}' = \langle \{\sigma\}, Q', S_0, \delta', F \rangle$ by “stretching” \mathcal{A} threefold: we define $Q' = \bigcup_{q \in Q} \{q, q', q''\}$ and the transition relation $\delta' = \{(q_1, \sigma, q'_1), (q'_1, \sigma, q''_1), (q''_1, \sigma, q_2) : (q_1, \sigma, q_2) \in \delta\}$. We then connect every non-accepting state q originally in \mathcal{A} to the initial states of the gadget OCN \mathcal{B} , and we connect every accepting state of \mathcal{A}' to a gadget NFA \mathcal{C} that accepts exactly $\{\sigma, \sigma\sigma\}$.

We now obtain from \mathcal{A}' an OCN \mathcal{A}'' by assigning counter updates of 0 on all transitions except those of \mathcal{B} .

It remains to prove that $L(\mathcal{A}) = \{\sigma\}^*$ iff \mathcal{A}'' is **Uniform-Det**.

Indeed, assume $L(\mathcal{A}) = \{\sigma\}^*$, then for every $n \in \mathbb{N}$, there is an accepting run of \mathcal{A}'' on σ^{3n} from counter 0. Since we have connected every accepting state in \mathcal{A} to the gadget \mathcal{C} that accepts both σ and $\sigma\sigma$, we have that σ^{3n+1} and σ^{3n+2} are accepted from counter 0 as well. Therefore, \mathcal{A}'' is universal for initial counter 0, hence it is universal for all initial counter values, and in particular \mathcal{A}'' is **Uniform-Det**.

Conversely, if \mathcal{A} is not universal, then there exists a word $w = \sigma^n$ such that all runs of \mathcal{A} on w end in non-accepting states (and at least one such run exists, by completeness). We then have that all successful runs of \mathcal{A}'' on σ^{3n} end in non-accepting states. The words $\sigma^{3n+1}, \sigma^{3n+2}$ can therefore only be accepted through \mathcal{B} . By the structure of \mathcal{B} we then have that $(3n+1, 1), (3n+2, 0) \in \text{MCR}(\mathcal{A}'')$, so $\text{MCR}(\mathcal{A}'')$ is not increasing, and \mathcal{A}'' is not **Uniform-Det**. ◀

5.2 Uniform-Det- Properties and Fragments

The wide complexity gap between the bounds of Theorems 17 and 19 suggest that **Uniform-Det** is an intricate problem. We now turn to present several results shed some light on the behaviour of **Uniform-Det**.

We start by showing that the first witness to the fact that $\text{MCR}(\mathcal{A})$ is non-increasing may be exponential in $|\mathcal{A}|$. This holds when \mathcal{A} has weights encoded in unary, and if the weights are encoded in binary this holds already for OCNs with 3 states.

► **Example 20.** Consider the OCN \mathcal{A} depicted in Figure 7, where k is encoded in binary. It is not hard to verify that for $0 \leq n \leq k$ it holds that $(n, \min(n, k-n+1)) \in \text{MCR}(\mathcal{A})$, but $(k+1, 0) \in \text{MCR}(\mathcal{A})$, since σ^{k+1} is accepted with counter value 0 in the left component. Thus, already for 3-state OCNs, the minimal witness for decreasing MCR can be exponential.



■ **Figure 7** Binary encoded OCN \mathcal{A} in Example 20.

► **Example 21.** We now describe a unary-encoded OCN whose minimal witness for decreasing MCR is exponential. Let p_1, \dots, p_m be the first m prime numbers. We construct an OCN \mathcal{A} as a disjoint union of m cycles of lengths p_1, \dots, p_m , where on each cycle all transitions have counter update -1 , and all states are accepting except the initial state on each cycle. In addition, \mathcal{A} has another initial and accepting self loop with counter update -2 .

Let $M = \prod_{i=1}^m p_i$, then for every $0 \leq n < M$ we have that $(n, n) \in \text{MCR}(\mathcal{A})$, since upon reading σ^n at least one cycle of length p_j does not divide n and is therefore not back at its initial state. Similarly, $(M+1, M+1) \in \text{MCR}(\mathcal{A})$. However, $(M, 2M) \in \text{MCR}(\mathcal{A})$ since σ^M is only accepted in the -2 self loop. Thus, the first witness for the non-increasing MCR is $M+1$, which is exponential in $|\mathcal{A}| = O(\sum_{i=1}^m p_i)$.

The next property shows that when all states are accepting, **Uniform-Det** becomes trivial.

► **Theorem 22.** *Consider an OCN \mathcal{A} over a singleton alphabet such that all states in \mathcal{A} are accepting. Then \mathcal{A} is **Uniform-Det**.*

Proof. We show that $\text{MCR}(\mathcal{A})$ is increasing, and therefore \mathcal{A} is **Uniform-Det**. Let $n_1, c_1 \in \mathbb{N}$ such that $(n_1, c_1) \in \text{MCR}(\mathcal{A})$, then initial counter c_1 is sufficient for \mathcal{A} to read (and hence accept) σ^{n_1} via some run ρ . Let $n_2 < n_1$, then \mathcal{A} reads σ^{n_2} along a prefix of ρ with initial counter value c_1 , and since all states are accepting, c_1 is sufficient to accept σ^{n_2} . Thus, if $(n_2, c_2) \in \text{MCR}(\mathcal{A})$, we have $c_2 \leq c_1$, so $\text{MCR}(\mathcal{A})$ is increasing. ◀

Our final property concerns unambiguous OCNs. An OCN \mathcal{A} over alphabet $\{\sigma\}$ is *unambiguous* if for every $n \in \mathbb{N}$ there exists at most one accepting run of \mathcal{A} on σ^n , for any counter value c . Technically, this means that the OCN is structurally unambiguous, in that its underlying NFA is unambiguous.

► **Theorem 23.** *For unambiguous OCNs over a singleton alphabet, deciding **Uniform-Det** is in PSPACE.*

Proof. Let \mathcal{A} be an unambiguous OCNs over a singleton alphabet. In Appendix A.6 we show that by careful analysis of the PA formula obtained as per Lemma 11, we can represent the notion of $\text{MCR}(\mathcal{A})$ being non-increasing using a PA formula ν that is a disjunction of exponentially many existential formulas – each polynomial in the size of \mathcal{A} . By traversing these fragments in polynomial space, and since existential PA is decidable in NP [8], we conclude the PSPACE bound. ◀

5.3 Triviality of 0-Det, \forall -Det, \exists -Det

We now turn to study the remaining notions of determinization for singleton alphabet.

► **Theorem 24.** *Consider an OCN \mathcal{A} over $\Sigma = \{\sigma\}$, then \mathcal{A} is \forall -Det, 0-Det, and \exists -Det.*

Proof. By Observation 1, it is enough to prove that \mathcal{A} is \forall -Det. To this end, recall that by Lemma 11, $\text{MCR}(\mathcal{A})$ is PA definable by a formula $\varphi(n, c)$.

For every initial counter value c , define $\varphi_{\leq c}(n) = \bigvee_{i=0}^c \varphi(n, i)$, then $\llbracket \varphi_{\leq c}(n) \rrbracket = \{n : \mathcal{A} \text{ accepts } \sigma^n \text{ with initial counter } c\}$. Then, we can write $\mathcal{L}(\mathcal{A}, c) = \{\sigma^m : m \in \llbracket \varphi_{\leq c}(n) \rrbracket\}$.

It is folklore that a singleton-alphabet language whose set of lengths is semilinear, is regular. We bring a short proof of this for completeness: Let $S = \bigcup_{i=1}^k \text{Lin}(c_i, p_i) \subseteq \mathbb{N}$ be a semilinear set (by assuming that the periods are linearly independent, it follows each has a single number), and let $L_S = \{\sigma^k : k \in S\}$. For every i , the language $\{a^k | k \in \text{Lin}(c_i, p_i)\}$ can be defined by the regular expression $r_i = \sigma^{c_i} (\sigma^{p_i})^*$. So L_S is defined by the regular expression $r = r_1 + \dots + r_k$.

Thus, for every $c \in \mathbb{N}$, we have that $\mathcal{L}(\mathcal{A}, c)$ is regular, and in particular is recognized by a DOCN, so \mathcal{A} is \forall -Det, and we are done. ◀

6 Discussion and Future Work

In this work, we introduce and study notions of determinization for OCNs. We demonstrate that the notions, while comparable in strictness, are distinct both from a conceptual perspective, having different motivations, as well as from a technical perspective: the mathematical tools needed to analyze them vary.

The most pressing direction for future work is resolving the decidability status of **Uniform-Det**. Note that **Uniform-Det** bears some similarities to the determinization problem for tropical automata, in that both models essentially follow the $(\min, +)$ semantics. The differences between the models are that (1) in OCNs we only care about Boolean acceptance, whereas in weighted automata we need to match the function exactly, and (2) in OCNs we have the restriction that the counter is nonnegative, unlike in weighted automata.

The determinization problem of weighted automata is famously open, and thus it could well be that **Uniform-Det** is similarly difficult. It is worth noting that techniques for handling the determinization of weighted automata in some fragments (namely unambiguous [25], or polynomially ambiguous [20]) can be easily shown not carry over to determinization of OCNs, meaning that besides the semantic differences, there are also technical differences in reasoning about these models.

Another important direction of future work is tightening the complexity gap of **Uniform-Det** over singleton alphabet. Our preliminary analysis in Lemma 11 suggests that this may require a more ad-hoc technique than using Presburger Arithmetic.

References

- 1 Parosh Aziz Abdulla and Karlis Čerāns. Simulation is decidable for one-counter nets. In *International Conference on Concurrency Theory*, pages 253–268. Springer, 1998.
- 2 Shaull Almagor, Udi Boker, Piotr Hofman, and Patrick Totzke. Parametrized universality problems for one-counter nets. In *31st International Conference on Concurrency Theory (CONCUR 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- 3 Leonard Berman. The complexity of logical theories. *Theoretical Computer Science*, 11(1):71–77, 1980.
- 4 Michael Blondin, Matthias Englert, Alain Finkel, Stefan Göller, Christoph Haase, Ranko Lazić, Pierre McKenzie, and Patrick Totzke. The reachability problem for two-dimensional vector addition systems with states. *Journal of the ACM (JACM)*, 68(5):1–43, 2021.
- 5 Stanislav Böhm, Stefan Göller, and Petr Jančár. Bisimilarity of one-counter processes is pspace-complete. In *International Conference on Concurrency Theory*, pages 177–191. Springer, 2010.
- 6 Stanislav Böhm, Stefan Göller, and Petr Jančár. Bisimulation equivalence and regularity for real-time one-counter automata. *Journal of Computer and System Sciences*, 80(4):720–743, 2014.
- 7 Udi Boker and Thomas A. Henzinger. Exact and approximate determinization of discounted-sum automata. *Log. Methods Comput. Sci.*, 10(1), 2014. doi:10.2168/LMCS-10(1:10)2014.
- 8 Itshak Borosh and Leon Bruce Treybig. Bounds on positive integral solutions of linear diophantine equations. *Proceedings of the American Mathematical Society*, 55(2):299–304, 1976.
- 9 Adam L Buchsbaum, Raffaele Giancarlo, and Jeffery R Westbrook. On the determinization of weighted finite automata. *SIAM Journal on Computing*, 30(5):1502–1531, 2000.
- 10 Wojciech Czerwiński and Slawomir Lasota. Regular separability of one counter automata. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE, 2017.
- 11 Alain Finkel, Gilles Geeraerts, J-F Raskin, and Laurent Van Begin. On the ω -language expressive power of extended petri nets. *Theoretical computer science*, 356(3):374–386, 2006.

- 12 Seymour Ginsburg and Edwin H Spanier. Bounded algol-like languages. *Transactions of the American Mathematical Society*, 113(2):333–368, 1964.
- 13 C. Haase. A survival guide to presburger arithmetic. *ACM SIGLOG News*, 5(3):67–82, 2018. URL: <https://dl.acm.org/citation.cfm?id=3242964>, doi:10.1145/3242953.3242964.
- 14 Christoph Haase. Subclasses of presburger arithmetic and the weak exp hierarchy. In *Proceedings of the 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)*, pages 1–10, 2014.
- 15 Piotr Hofman, Richard Mayr, and Patrick Totzke. Decidability of weak simulation on one-counter nets. In *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 203–212. IEEE, 2013.
- 16 Piotr Hofman and Patrick Totzke. Trace inclusion for one-counter nets revisited. *Theoretical Computer Science*, 735:50–63, 2018.
- 17 Petr Jančar, Antonín Kučera, and Faron Moller. Simulation and bisimulation over one-counter processes. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 334–345. Springer, 2000.
- 18 Petr Jančar and Faron Moller. Simulation of one-counter nets via colouring. In *Proceedings of Workshop Journées Systemes Infinis*, pages 1–6, 1999.
- 19 Petr Jančar, Faron Moller, et al. Simulation problems for one-counter machine. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 404–413. Springer, 1999.
- 20 Daniel Kirsten and Sylvain Lombardy. Deciding unambiguity and sequentiality of polynomially ambiguous min-plus automata. In *26th International Symposium on Theoretical Aspects of Computer Science*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2009.
- 21 Ines Klimann, Sylvain Lombardy, Jean Mairesse, and Christophe Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *Theoretical Computer Science*, 327(3):349–373, 2004.
- 22 Jérôme Leroux and Grégoire Sutre. On flatness for 2-dimensional vector addition systems with states. In *International Conference on Concurrency Theory*, pages 402–416. Springer, 2004.
- 23 Richard Mayr. Undecidable problems in unreliable computations. *Theoretical Computer Science*, 297(1-3):337–354, 2003.
- 24 Marvin Lee Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall Englewood Cliffs, 1967.
- 25 M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311, 1997.
- 26 Mojżesz Presburger. Über die vollständigkeit eines gewissen systems der arithmetik ganzer zahlen, in welchen die addition als einzige operation hervortritt. In *Comptes-Rendus du ler Congres des Mathematiciens des Pays Slavs*, 1929.
- 27 Philippe Schnoebelen. Lossy counter machines decidability cheat sheet. In *International Workshop on Reachability Problems*, pages 51–75. Springer, 2010.
- 28 Jiří Srba. Beyond language equivalence on visibly pushdown automata. *Logical Methods in Computer Science*, 5, 2009.
- 29 Larry J Stockmeyer and Albert R Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 1–9, 1973.
- 30 Patrick Totzke, Richard Mayr, Slawomir Lasota, and Piotr Hofman. Simulation problems over one-counter nets. *Logical Methods in Computer Science*, 12, 2016.
- 31 Leslie Valiant. *Decision procedures for families of deterministic pushdown automata*. PhD thesis, University of Warwick, 1973.
- 32 Rüdiger Valk and Guy Vidal-Naquet. Petri nets and regular languages. *Journal of Computer and system Sciences*, 23(3):299–325, 1981.

A Proofs

A.1 Proof of Lemma 2

A.1.1 \mathcal{A} is \exists -Det, but not 0-Det

We define formally $\mathcal{A} = \langle \{a, b, c, \#\}, \{q_0, q', q'', q_5\}, q_0, \delta_{\mathcal{A}}, \{q', q'', q_5\} \rangle$, for:

$$\delta_{\mathcal{A}} = \langle (q_0, \#. - 5, q_5), (q_0, \#.0, q'), (q_0, \#.0, q''), (q', a, 1, q'), (q', b, 0, q') \rangle \cup \{(q', c, -1, q'), (q'', a, 0, q''), (q'', b, 1, q''), (q'', c, -1, q''), (q_5, a, 0, q_5), (q_5, b, 0, q_5), (q_5, c, 0, q_5)\}.$$

\mathcal{A} is \exists -Det, since $\mathcal{L}(\mathcal{A}, k) = \Sigma^*$ for $k \geq 5$. Now, assume by way of contradiction that \mathcal{A} is 0-Det, and let \mathcal{D} be a deterministic OCN with $n \in \mathbb{N}$ states that satisfies $\mathcal{L}(\mathcal{A}, 0) = \mathcal{L}(\mathcal{D}, 0)$. We now define $w = \#c^{n+1}a^{n+1}b^{n+1}$. throughout the run of \mathcal{D} on w , \mathcal{D} travels through a cycle β_1 when reading a^{n+1} , and a cycle β_2 when reading b^{n+1} . If the cumulative costs of both β_1 and β_2 are non-negative, then \mathcal{D} accepts $w' = \#c^{n+1}a^N b^N$ for arbitrarily large $N \in \mathbb{N}$, which contradicts $\mathcal{L}(\mathcal{A}, 0) = \mathcal{L}(\mathcal{D}, 0)$. Otherwise, the cumulative cost of either β_1 or β_2 is negative, w.l.o.g β_1 . In this case, $w' = \#c^{n+1}a^N$ is not accepted by \mathcal{D} for sufficiently large $N \in \mathbb{N}$, which again contradicts $\mathcal{L}(\mathcal{A}, 0) = \mathcal{L}(\mathcal{D}, 0)$. \blacktriangleleft

A.1.2 \mathcal{B} is 0-Det, but not \forall -Det

We define formally $\mathcal{B} = \langle \{a, b, c, \#\}, \{q_0, q', q''\}, q_0, \delta_{\mathcal{B}}, \{q', q''\} \rangle$, for:

$$\delta_{\mathcal{B}} = \{(q_0, \#. - 1, q'), (q_0, \#. - 1, q''), (q', a, 1, q'), (q', b, 0, q'), (q', c, -1, q')\} \cup \{(q'', a, 0, q''), (q'', b, 1, q''), (q'', c, -1, q'')\}.$$

Since $\mathcal{L}(\mathcal{B}, 0) = \emptyset$, \mathcal{B} is 0-Det trivially. However, since with initial counter 0, both $(q_0, \#. - 1, q')$ and $(q_0, \#. - 1, q'')$ cannot be traversed, we have that $\mathcal{L}(\mathcal{B}, 1) = \mathcal{L}(\mathcal{A}, 0)$, therefore, as can be shown by an identical analysis to the one presented in Appendix A.1.1, there is no deterministic OCN \mathcal{D} that satisfies $\mathcal{L}(\mathcal{B}, 1) = \mathcal{L}(\mathcal{D}, 0)$, and \mathcal{B} is not \forall -Det. \blacktriangleleft

A.1.3 \mathcal{C} is \forall -Det, but not Uniform-Det

We define formally $\mathcal{C} = \langle \{a, b, \#\}, \{q_0, q_1, q_2\}, q_0, \delta_{\mathcal{C}}, \{q_1, q_2\} \rangle$, for:

$$\delta_{\mathcal{C}} = \{(q_0, \#.0, q_1), (q_0, \#. - 1, q_2), (q_1, a.1, q_1), (q_1, b, -1, q_1)\} \cup \{(q_2, a, 0, q_2), (q_2, b, 0, q_2)\}.$$

For initial counter 0, the transition $(q_0, \#. - 1, q_2)$ cannot be traversed, therefore \mathcal{C} is 0-Det, since $\mathcal{D} = \langle \{a, b, \#\}, \{q_0, q_1\}, q_0, \{(q_0, \#.0, q_1), (q_1, a.1, q_1), (q_1, b, -1, q_1)\}, \{q_1\} \rangle$ satisfies $\mathcal{L}(\mathcal{D}, 0) = \mathcal{L}(\mathcal{C}, 0)$. In addition, $\mathcal{L}(\mathcal{C}, k) = \#\{a, b\}^*$ for all $k \geq 1$. Hence \mathcal{C} is \forall -Det.

Now assume by way of contradiction that \mathcal{C} is Uniform-Det, and let \mathcal{D} be a deterministic OCN with $n \in \mathbb{N}$ states that satisfies $\mathcal{L}(\mathcal{D}, k) = \mathcal{L}(\mathcal{C}, k)$ for all $k \in \mathbb{N}$, and let $w = \#a^{n+1}b^{n+1} \in \mathcal{L}(\mathcal{D}, k)$ for all $k \in \mathbb{N}$. \mathcal{D} travels through a cycle β when reading b^{n+1} . If the cumulative weight of β is non-negative, then $w' = \#a^{n+1}b^N \in \mathcal{L}(\mathcal{D}, 0)$ for arbitrarily large $N \in \mathbb{N}$, which contradicts $\mathcal{L}(\mathcal{D}, 0) = \mathcal{L}(\mathcal{C}, 0)$. If, however, the cumulative weight of β is negative, then $w' = \#a^{n+1}b^N \notin \mathcal{L}(\mathcal{D}, 1)$ for large enough $N \in \mathbb{N}$, which in turn contradicts $\mathcal{L}(\mathcal{D}, 1) = \mathcal{L}(\mathcal{C}, 1)$. \blacktriangleleft

A.2 Proof of Lemma 3

We prove undecidability of 0-FINITE-REACH using a straightforward reduction from FINITE-REACH. Given an LCM $\mathcal{M} = \langle \text{Loc}, \mathcal{C}, \Delta \rangle$ and a configuration $\sigma_0 = \langle q, (a_1, a_2 \dots a_n) \rangle$, we define an LCM \mathcal{M}' with a new initial state q_0 that leads to q with a single path that increments z_1 a_1 times, z_2 a_2 times, etc.

Formally, If $a_i = 0$ for all $0 \leq i \leq n$, we define $\mathcal{M}' = \mathcal{M}$ and the reduction is trivial. Otherwise, we define $\mathcal{M}' = (\text{Loc}', \mathcal{C}, \Delta')$ such that:

- $\text{Loc}' = \text{Loc} \cup \{q_0\} \cup \left\{ \{q_i\}_{i=1}^{\sum_{j=1}^n a_j - 1} \right\}$. Note that if $\sum_{j=1}^n a_j = 1$, the only new state added is q_0 .
- $\Delta' = \Delta \cup \left\{ (q_{\sum_{j=1}^n a_j - 1}, (z_y, ++), q) \right\} \cup \left\{ (q_i, (z_x, ++), q_{i+1}) \right\}$ such that y is the largest integer $0 \leq y \leq n$ for which $a_y \neq 0$, and the parameter x varies such that throughout the $\sum_{j=1}^n a_j$ transitions, each counter z_i is incremented exactly a_i times.

The reachability set of $\sigma_0 = \langle q, (a_1, a_2 \dots a_n) \rangle$ under \mathcal{M} is finite iff the reachability sets of all configurations $\sigma'_0 = \langle q, (a'_1, a'_2 \dots a'_n) \rangle$ such that $a'_i \leq a_i$ for all i are finite, due to monotonicity of LCMs. This, in turn, is satisfied iff the reachability set of $\langle q_0, (0, 0 \dots 0) \rangle$ under \mathcal{M}' is finite. ◀

A.3 Proof of Lemma 4

We start by describing several gadgets used in the construction.

A.3.1 Gadgets

Let $\mathcal{M} = \langle \text{Loc}, \mathbb{Z}, \Delta \rangle$ be an LCM, let $z_i \in \mathbb{Z}$, and let $(\ell_1, \text{op}, \ell_2) \in \Delta$. Our goal is to construct an OCN \mathcal{A} that reads two consecutive configuration encodings - an encoding that corresponds to a visit in ℓ_2 and then an encoding that corresponds to a visit in ℓ_1 , such that $w \in \mathcal{L}(\mathcal{A}, 0)$ iff w admits a violation for counter z_i .

The structure of \mathcal{A} depends on the value of op , which can any of the following:

1. z_i++ , i.e., increment z_i ,
2. z_i-- , i.e., decrement z_i ,
3. z_j++ or z_j-- for $j \neq i$, which does not affect z_i ,
4. $z_i = 0?$, i.e., test z_i for 0.

In addition, we have a special gadget to capture violations in the initial configuration, namely if the counter values is not 0 (recall that the initial configuration is read last, since the encoding is reversed).

Thus, \mathcal{A} can be any of the gadgets presented in figure 8 (depending on op).

Formally, we define $\mathcal{A} = \langle \Sigma, \{q_0, q_1, q_2\}, q_0, \delta, \{q_2\} \rangle$ such that:

- $\Sigma = \text{Loc} \cup \left\{ \{a_i\}_{z_i \in \mathbb{Z}} \right\}$.
- $\delta = \left\{ (q_0, a_j, 0, q_0) \right\}_{j \neq i} \cup \left\{ (q_0, a_i, 1, q_0) \right\} \cup \left\{ (q_0, \ell_2, \nu, q_1) \right\} \cup \left\{ (q_1, a_j, 0, q_1) \right\}_{j \neq i} \cup \left\{ (q_1, a_i, -1, q_1) \right\} \cup \left\{ (q_1, \ell_1, 0, q_2) \right\} \cup \left\{ (q_2, \sigma, 0, q_2) \right\}_{\sigma \in \Sigma}$.

For the initial configuration checker, we define $\mathcal{A} = \langle \Sigma, \{q_0, q_1\}, q_0, \delta, \{q_1\} \rangle$ such that:

- $\Sigma = \text{Loc} \cup \left\{ \{a_i\}_{z_i \in \mathbb{Z}} \right\}$.
- $\delta = \left\{ (q_0, a_j, 1, q_0) \right\}_{z_j \in \mathbb{Z}} \cup \left\{ (q_0, \ell_0, -1, q_1) \right\}$.

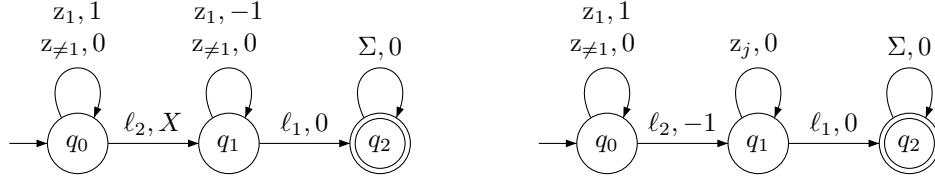
Our last gadget captures ill-formed words, regardless of counter values.

Let $\mathcal{M} = \langle \text{Loc}, \mathbb{Z}, \Delta \rangle$ be an LCM. we say that a word w is *well formed* if the following conditions are satisfied:

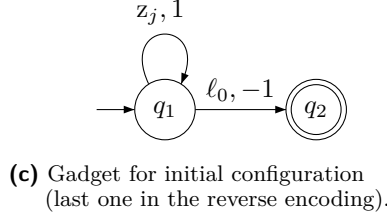
1. w is of the form $w = a_n^* \dots a_1^* \ell_{iN} \dots a_n^* \dots a_1^* \ell_{i0}$ for $\{\ell_{ij} \in \text{Loc}\}_{0 \leq j \leq N}$.
2. $\ell_{i0} = \ell_0$.
3. for every $0 \leq j \leq N - 1$, there is at least one transition in \mathcal{M} that leads from ℓ_{ij} to $\ell_{i,j+1}$.

It is easy to see that well formed words are a regular language, and in particular its complement is the desired OCN.

18:20 Determinization of One-Counter Nets



(a) Gadgets for scenarios 1, 2, and 3, by setting X to be $-2, 0$, and -1 , respectively. (b) Gadget for scenario 4.



(c) Gadget for initial configuration (last one in the reverse encoding).

■ **Figure 8** The violation-check gadgets for z_1 . By $z_{\neq 1}$ we mean z_j for all $j \neq 1$, and by z_j we mean every counter.

A.3.2 The Main Construction

Let $\mathcal{M} = \langle \text{Loc}, \mathbb{Z}, \Delta \rangle$. We wish to construct an OCN \mathcal{A} such that $\mathcal{L}(\mathcal{A}, 0)$ is the set of all words that do not represent legal runs of \mathcal{M} .

Intuitively, we construct \mathcal{A} through the following process:

1. Construct a flow violation checker (with regards to \mathcal{M}), which will be part of \mathcal{A} as a separate component.
2. for every location $\ell \in \text{Loc}$, add a corresponding state ℓ' in \mathcal{A} . all such ℓ' 's are initial states in \mathcal{A} , and they all have self loops with weight 0 when reading all counter accumulators $\{a_i\}_{z_i \in \mathbb{Z}}$. Intuitively, when \mathcal{A} visits a state ℓ' , it means that \mathcal{A} is currently in the process of reading a configuration in which \mathcal{M} is in location ℓ .
3. for every transition $(\ell_1, \text{op}, \ell_2) \in \Delta$, add the transition $(\ell'_2, \ell_2, 0, \ell'_1)$ to \mathcal{A} . Intuitively, traveling this transition means that \mathcal{A} has finished reading a configuration of location ℓ_2 , and is now starting to read a configuration of location ℓ_1 .
4. connect an initial configuration violation checker to ℓ'_0 .
5. for every transition $(\ell_1, \text{op}, \ell_2) \in \Delta$, add from ℓ'_2 transitions to all relevant violation checkers for all counters $\{z_i\}_{1 \leq i \leq n}$.

Now let us define the construction formally. Let $V(\ell_i \rightarrow \ell_j, z_m)$ be the violation checker that matches the transition $(\ell_i, \text{op}, \ell_j)$ for counter z_m , as detailed in Appendix A.3.1. Let $Q(\ell_i \rightarrow \ell_j, z_m)$ be its states, let $F(\ell_i \rightarrow \ell_j, z_m)$ be its accepting states, let $\delta(\ell_i \rightarrow \ell_j, z_m)$ be its transitions, and $\lambda(\ell_i \rightarrow \ell_j, z_m) \subseteq \delta(\ell_i \rightarrow \ell_j, z_m)$ be the transitions from its initial state. In that spirit we also define, with regards to the flow control violation checker, and the initial configuration violation checker: $Q(\text{initial}), \delta(\text{initial}), \lambda(\text{initial}), Q(\text{flow}), \delta(\text{flow}), \lambda(\text{flow})$. Lastly, for convenience' sake alone we define \mathcal{A} as having multiple initial states. this has been done for readability, and can easily be formally circumvented by defining a single initial state α_0 , along with an outgoing transition (α_0, σ, z, q) for each $(s_0, \sigma, z, q) \in \delta$.

We now define $\mathcal{A} = \langle \Sigma, Q, S_0, \delta, F \rangle$ such that:

- $\Sigma = \text{Loc} \cup \{a_i\}_{z_i \in \mathbb{Z}}$
- $Q = \{\ell'_i\}_{\ell_i \in \text{Loc}} \cup Q(\text{initial}) \cup Q(\text{flow}) \cup \{Q(\ell_i \rightarrow \ell_j, z_m)\}$ for all $\ell_i, \ell_j \in \text{Loc}$ such that there is a transition from ℓ_i to ℓ_j in Δ , and for all $1 \leq i \leq m$.

- $S_0 = \{\ell'_i\}_{\ell_i \in \text{Loc}} \cup \{s_{0,\text{flow}}\}$ such that $s_{0,\text{flow}}$ is the initial state of the flow violation checker.
- $\delta_1 = \{(\ell'_i, a_j, 0, \ell'_i)\}$ for all $\ell_i \in \text{Loc}$ and for all $1 \leq j \leq n$.
- $\delta_2 = \{(\ell'_i, \ell_i, 0, \ell'_j)\}$ for all $\ell_i, \ell_j \in \text{Loc}$ such that there is a transition from ℓ_j to ℓ_i in Δ .
- $\delta_3 = \{(\ell'_i, \sigma, \nu, q')\}$ for all $\ell_i, \ell_j \in \text{Loc}$ such there is a transition from ℓ_j to ℓ_i in Δ , and $(q, \sigma, \nu, q') \in \lambda(\ell_j \rightarrow \ell_i, z_m)$ for some $1 \leq i \leq m$, or otherwise $(q, \sigma, \nu, q') \in \lambda(\text{initial})$.
- $\delta_V = \bigcup_{\text{all violations}} \delta(\text{violation})$.
- $\delta = \delta_1 \cup \delta_2 \cup \delta_3 \cup \delta_V$
- $F = \bigcup_{\text{all violations}} F(\text{violation})$.

We turn to prove the correctness of the construction. Consider a word w that represents a legal run of \mathcal{M} . Then, first of all, w is well formed, and therefore not accepted by the flow violation checker. second, there is no transition from one configuration to the next that involves a violation, and therefore w cannot be accepted through any of the violation checkers in \mathcal{A} . Since all accepting states of \mathcal{A} are inside violation checkers, $w \notin \mathcal{L}(\mathcal{A}, 0)$.

Conversely, assume a word w does not represent a legal run of \mathcal{M} . If w is not well formed, then it is accepted through the flow violation checker. Otherwise - a transition from a state $\ell_i \in \text{Loc}$ to a state $\ell_j \in \text{Loc}$ represents a violation for counter z_m such that $1 \leq m \leq n$. \mathcal{A} then accepts w by branching from ℓ'_j to $V(\ell_i \rightarrow \ell_j, z_m)$ at the right moment. It is also possible that the violation occurs in the first configuration (last one to be read), and in this case w will be accepted through the initial configuration violation checker.

A.4 Details for the proof of Lemma 5

The following is a formal construction of DFA $\mathcal{D} = \langle \Sigma, Q', s'_0, \delta', F' \rangle$:

- $Q' = \{ \langle \ell, a_1 \dots a_k, b_1 \dots b_k \rangle \mid \ell \in \text{Loc}, 0 \leq a_i, b_i \leq m \text{ for all } 1 \leq i \leq k \} \cup \{ \langle \perp, \perp \dots \perp, b_1 \dots b_k \rangle \mid 0 \leq b_i \leq m \text{ for all } 1 \leq i \leq k \}$.
- $s'_0 = \langle \perp, \perp \dots \perp, 0 \dots 0 \rangle$.
- $\delta'(\langle \ell, a_1 \dots a_k, b_1 \dots b_k \rangle, \ell') = \langle \ell', b_1 \dots b_k, 0 \dots 0 \rangle$ if the configuration $\langle \ell, a_1 \dots a_k \rangle$ can be obtained from the configuration $\langle \ell', b_1 \dots b_k \rangle$ through a single transition in \mathcal{M} .
- $\delta'(\langle \perp, \perp \dots \perp, b_1 \dots b_k \rangle, \ell) = \langle \ell, b_1 \dots b_k, 0 \dots 0 \rangle$ for all $\ell \in \text{Loc}, 0 \leq b_1 \dots b_k \leq m$.
- $\delta'(\langle \ell, a_1 \dots a_k, 0 \dots 0, b_j \dots b_k \rangle, z_j) = \langle \ell, a_1 \dots a_k, 0 \dots 0, b_j + 1 \dots b_k \rangle$ for all $0 \leq j \leq k$, $b_j < m$.
- $\delta'(\langle \ell, a_1 \dots a_k, 0 \dots 0, b_j \dots b_k \rangle, z_{j-x}) = \langle \ell, a_1 \dots a_k, 0 \dots 1, 0 \dots b_j \dots b_k \rangle$ for all $1 \leq j \leq k$, $1 \leq x \leq j$.
- $F = \{ \langle \ell_0, 0 \dots 0, 0 \dots 0 \rangle \}$.

Correctness stems directly from the construction.

As for the other direction, assume the reachability set of $\langle \ell_0, (0 \dots 0) \rangle$ under \mathcal{M} is infinite, and assume by way of contradiction that \mathcal{A} has a deterministic equivalent \mathcal{D} with d states. Observe that for every word $u \in \Sigma^*$, the run of \mathcal{D} does not end due to the counter becoming negative. Indeed, we can always concatenate some $\lambda \in \Sigma^*$ such that $u\lambda$ does not correspond to a run, and is hence accepted by \mathcal{D} , so the run on u must be able to continue reading λ . We call this property of \mathcal{D} *positivity*.

Since the reachability set of $\langle \ell_0, (0 \dots 0) \rangle$ is infinite, there exists a counter of \mathcal{M} , w.l.o.g z_1 , that can take unbounded values (in different runs). Let w be a word corresponding to a run of \mathcal{M} that ends with the value of z_1 being N for some $N > d$. We can then write $w = a_k^* \dots a_1^N \ell a_k^* \dots a_1^{N'} \ell' \rho$ such that ρ represents the reverse of a legal prefix of a run of \mathcal{M} , and N' satisfies $N' \geq N - 1$, since no single transition of \mathcal{M} can increase a counter by more than one (but N' can be arbitrarily large).

18:22 Determinization of One-Counter Nets

Since w corresponds to a legal run of \mathcal{M} , \mathcal{A} (and therefore \mathcal{D}) does not accept w . By the positivity of \mathcal{D} , its run on w ends in a non-accepting state.

Since $N > d$, \mathcal{D} goes through a cycle β when reading a_1^N . We pump the cycle β to obtain a run of \mathcal{D} on a word $w'' = a_k^* \cdots a_1^{N+t} q a_k^* \cdots a_1^{N'} q' \rho$ for some $t \in \mathbb{N}$ that satisfies $N + t > N' + 1$. Again, by the positivity of \mathcal{D} , the run cannot end due to the counter becoming negative, so it ends in the same non-accepting state as the run on w . However, w'' does not represent a legal run of M , since $N + t > N' + 1$, therefore $w'' \in \mathcal{L}(\mathcal{A}, 0)$, which contradicts $\mathcal{L}(\mathcal{A}, 0) = \mathcal{L}(\mathcal{D}, 0)$.

A.5 Details for the Proof of Lemma 14

We start by demonstrating our method, followed by the general construction. Consider, for example, $S = \text{Lin}((1, 0), (4, 8)) \cup \text{Lin}((2, 1), (6, 12))$. In this case $\gamma = 12$. We split $\text{Lin}((1, 0), (4, 8))$ to $\text{Lin}((1, 0), (12, 24)) \cup \text{Lin}((5, 8), (12, 24)) \cup \text{Lin}((9, 16), (12, 24))$, the intuition being that instead of a (4, 8) period, we have a (12, 24) period, and we add different basis vectors to fill the gaps, so the new basis vectors are (5, 8) and (9, 16), where the next basis vector (13, 24) is already captured by $(1, 0) + (12, 24)$. Similarly, we split $\text{Lin}((2, 1), (6, 12))$ to $\text{Lin}((2, 1), (12, 24)) \cup \text{Lin}((8, 13), (12, 24))$. Overall we get $S = \bigcup_{v \in V} \text{Lin}(v, (12, 24))$ for $V = \{(1, 0), (5, 8), (9, 16), (2, 1), (8, 13)\}$.

Generally, let $\gamma = \text{lcm}(\{p_i\}_{i=1}^M)$. We split each linear component $\text{Lin}((a_i, b_i), \{(p_i, r_i)\})$ to $\frac{\gamma}{p_i}$ parts, by defining the γ -split of $\text{Lin}((a_i, b_i), (p_i, r_i))$ (defined only for $p_i | \gamma$) to be $\bigcup_{i=0}^{\frac{\gamma}{p_i}-1} \text{Lin}((a_i, b_i) + i \cdot (p_i, r_i), (\gamma, r_i) \cdot \frac{\gamma}{r_i})$. Each such split is semilinear by definition, and it is straightforward to show that $S = \bigcup_{i=1}^k \text{l-split}(\text{Lin}((a_i, b_i), (p_i, r_i)))$.

A.6 Unambiguous OCNs

We now consider the case where \mathcal{A} is unambiguous. Observe that in order to construct $\theta(n, c)$ above, we explicitly placed the requirement that the counter is minimal. As we now show, if \mathcal{A} is unambiguous, we can modify the formula such that no universal quantification is required.

Recall that in the construction of the formula $\varphi_\rho(n, c)$, we define the subformula $\text{SUFFICIENT-COUNTER}_\rho(c, e_1, \dots, e_k)$, stating that the counter c is sufficient for traversing the run $\alpha_0 \beta_1^{e_1} \alpha_1 \beta_2^{e_2} \alpha_2 \cdots \alpha_k$. The structure of $\text{SUFFICIENT-COUNTER}_\rho(c, e_1, \dots, e_k)$ can be viewed as a conjunction of inequalities $\bigwedge_j \tau_j \geq 0$ where each τ_j is a linear expression containing c . We observe that c is a minimal counter that satisfies these equations iff one of them is satisfied as an equality.

In addition, for unambiguous OCNs, if $\text{SUFFICIENT-COUNTER}_\rho(c, e_1, \dots, e_k)$ is satisfied, then all alternative values e'_1, \dots, e'_k for which this formula is satisfied represent the same run. Therefore, if an initial counter value c is minimal for words of length n and certain e_1, \dots, e_k , then it is minimal for all alternative e'_1, \dots, e'_k . We can then construct the following formula

$$\begin{aligned} \psi_\rho(n, c) := & \exists e_1 \cdots e_k, \text{CORRECT-LENGTH}_\rho(n, e_1 \cdots e_k) \\ & \wedge \text{SUFFICIENT-COUNTER}_\rho(c, e_1 \cdots e_k) \\ & \wedge \text{MINIMAL-COUNTER}_\rho(c, e_1 \cdots e_k) \end{aligned}$$

where $\text{MINIMAL-COUNTER}_\rho(c, e_1 \cdots e_k) := \bigvee_j \tau_j = 0$ where τ_j are the inequalities that appear in $\text{SUFFICIENT-COUNTER}_\rho(c, e_1 \cdots e_k)$.

By the above, we have that $\psi_\rho(n, c)$ is satisfied iff c is the minimal counter value such that there exists a run of length n that is of the shape ρ starting from counter value c .

Defining $P \subseteq S$ to be the set of linear path schemes from the initial state to an accepting state, as above, we can rewrite θ more compactly, as follows: $\theta(n, c) = \bigvee_{\rho \in P} \varphi_\rho(n, c)$.

As for the bigger picture, we remind the reader that **Uniform-Det** can be decided using $\nu = \exists n_1, n_2, c_1, c_2, n_1 < n_2 \wedge c_1 > c_2 \wedge \theta(n_1, c_1) \wedge \theta(n_2, c_2)$. In the unambiguous case, we can rewrite ν as follows:

$$\begin{aligned} \nu = & \bigvee_{\rho_1, \rho_2 \in P} \exists n_1, c_1, n_2, c_2, e_{11}, \dots, e_{1k_1}, e_{21}, \dots, e_{2k_2}, n_1 < n_2 \wedge c_1 > c_2 \wedge \\ & \text{CORRECT-LENGTH}_{\rho_1}(n_1, e_{11} \dots e_{1k_1}) \wedge \text{SUFFICIENT-COUNTER}_{\rho_1}(c_1, e_{11} \dots e_{1k_1}) \\ & \wedge \text{MINIMAL-COUNTER}_{\rho_1}(c_1, e_{11} \dots e_{1k_1}) \wedge \text{CORRECT-LENGTH}_{\rho_2}(n_2, e_{21} \dots e_{2k_2}) \\ & \wedge \text{SUFFICIENT-COUNTER}_{\rho_2}(c_2, e_{21} \dots e_{2k_2}) \wedge \text{MINIMAL-COUNTER}_{\rho_2}(c_2, e_{21} \dots e_{2k_2}). \end{aligned}$$

This representation of ν is a disjunction of existential fragments, all of which are polynomial in the size of \mathcal{A} .

Energy Games with Resource-Bounded Environments

Orna Kupferman ✉

School of Computer Science and Engineering, Hebrew University of Jerusalem, Israel

Naama Shamash Halevy ✉

School of Computer Science and Engineering, Hebrew University of Jerusalem, Israel

Abstract

An *energy game* is played between two players, modeling a resource-bounded system and its environment. The players take turns moving a token along a finite graph. Each edge of the graph is labeled by an integer, describing an update to the energy level of the system that occurs whenever the edge is traversed. The system wins the game if it never runs out of energy. Different applications have led to extensions of the above basic setting. For example, addressing a combination of the energy requirement with behavioral specifications, researchers have studied richer winning conditions, and addressing systems with several bounded resources, researchers have studied games with multi-dimensional energy updates. All extensions, however, assume that the environment has no bounded resources.

We introduce and study *both-bounded energy games* (BBEGs), in which both the system and the environment have multi-dimensional energy bounds. In BBEGs, each edge in the game graph is labeled by two integer vectors, describing updates to the multi-dimensional energy levels of the system and the environment. A system wins a BBEG if it never runs out of energy or if its environment runs out of energy. We show that BBEGs are determined, and that the problem of determining the winner in a given BBEG is decidable iff both the system and the environment have energy vectors of dimension 1. We also study how restrictions on the memory of the system and/or the environment as well as upper bounds on their energy levels influence the winner and the complexity of the problem.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Energy Games, Infinite-State Systems, Decidability

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.19

Related Version *Full Version*: <https://www.cs.huji.ac.il/~ornak/publications/concur22.pdf>

Funding Supported by the Israel Science Foundation, grant No. 2357/19.

1 Introduction

A reactive system interacts with its environment and should behave correctly in all environments. Synthesis of a reactive system thus corresponds to finding a winning strategy in a *two-player game* between the system and the environment. The game is played on a graph whose vertices are partitioned between the players. Starting from some initial vertex, the players move a token along the graph: whenever the token is in a vertex owned by the system, the system decides to which successor to move the token, and similarly for the environment. Together, the players generate a path in the graph. The choices of the players correspond to actions that the system and the environment may take, and so the generated path corresponds to a possible outcome of an interaction between the system and its environment.

The winning condition in the game is induced by the correctness criteria for the system. Early work on synthesis focuses on qualitative criteria, typically described by a temporal logic formula that specifies the allowed interactions [26, 3]. There, the essence of the actions that the system and the environment take is the way they modify the truth assignment to input



© Orna Kupferman and Naama Shamash Halevy;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 19; pp. 19:1–19:23

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and output signals. Accordingly, the edges of the graph are labeled by such assignments, and the generated path is an infinite word over the alphabet of assignment. The system wins if this word satisfies the specification. Recent work studies also games with quantitative objectives. There, the essence of the actions that the system and the environment take is the way they modify some quantitative measure, such as a budget or an energy level. Accordingly, the edges of the graph are labeled by updates to the quantitative measure, and the winning condition refers to properties like its limit sum or average [17].

Energy games belong to the second class of games: the two players model a *resource-bounded* system and its environment. Accordingly, each edge of the game graph is labeled by an integer, describing an update to the energy level of the system that occurs whenever the edge is traversed. The system wins the game if it never runs out of energy. The term “energy” may refer to a wide range of applications: an actual energy level, where actions involve consumption or charging of energy; storage, where actions involve storing or freeing disc space; money ones, where actions involve costs and rewards to a budget of some economic entity, and more [11].

Different applications have led to extensions of the above basic setting. For example, addressing a combination of the energy requirement with behavioral specifications, researchers have studied *energy parity games*, whose winning conditions combine quantitative and qualitative conditions [9, 2]. Then, addressing systems with several bounded resources, researchers have studied *generalized energy games*, in which the system player has a multi-dimensional energy level, the updates along the edges are vectors of integers, and the system wins if it does not run out of energy in any of its resources.

Two main questions regarding energy games have been studied. The first, called the *unknown initial-credit problem*, is the problem of deciding the existence of an initial energy level that is sufficient for the system to win the game. The second, called the *given initial-credit problem*, is the problem of deciding whether a given initial energy level is sufficient for the system to win. It is shown in [6, 8] that *memoryless strategies*, namely strategies that decide how to direct the token based on its current location, are sufficient to win energy games, and that consequently, both the unknown and the given initial-credit problems are decidable in $\text{NP} \cap \text{coNP}$. For multi-dimensional energy games, the unknown initial-credit problem is coNP -complete [10], whereas the given initial-credit problem (a.k.a. *Z-reachability VASS game*) is 2EXPTIME -complete [7, 12, 19].

We introduce and study *both-bounded energy games* (BBEGs), in which both the system and the environment have (multi-dimensional) energy bounds. In BBEGs, each edge in the game graph is labeled by two integer vectors, describing updates to the multi-dimensional energy levels of the system and the environment. A system wins a BBEG if it never runs out of energy or if its environment runs out of energy.

Bounded environments are of interest in several paradigms in computer science. For example, in cryptography, one studies the security of a given cryptosystem with respect to attackers with bounded (typically polynomial) computational power [24]. In the analysis of on-line algorithms, one sometimes cares for the competitive ratio of a given on-line algorithm with respect to requests issued by a bounded adversary [5]. Likewise, studying bounded rationality in games, bounds are placed on the power of the players. Closer to the work here is the extension of *bounded synthesis* [27] to settings where both the system and the environment have bounds on their size [21]. In addition to better modeling the studied setting, the bounds are sometimes used in order to obtain decidability or better complexity, and they can also serve in heuristics, as in SAT-based algorithms for bounded synthesis [13]. Finally, a setting in which the system and the environment have similar properties (in particular,

both are bounded) enjoys *duality* between the players. Adding budget constraints to the environment makes the players in energy games dual up to the player that moves first and the definition of who wins when the game continues forever. From a practical point of view, in many of the scenarios modeled by energy games, the environment is another system, hence with its own bounds. This includes, for example, a robot that interacts with another robot, both having bounded batteries, or a consumer that interacts with a company, both having bounded budgets.

We show that BBEGs are determined, and that the problem of determining the winner in a given BBEG is decidable iff both the system and the environment have energy vectors of dimension 1. This is both bad news, as traditional energy games are decidable for all dimensions [7], and good news, as adding an (unbounded) energy level to the environment causes even the setting with energy vectors of dimension 1 to include two unbounded components, as in two-counter machines [25]. In order to show decidability, we relate the energy level of the environment with the value of a counter in *one-counter energy games* [1], which augment energy games with a counter. Once, however, the system or the environment has an energy vector of dimension 2, we can use the energy level of the other player to store the sum of the counters, which enable us to simulate a two-counter machine by a BBEG in which the dimension of the energy vector of one of the players is strictly bigger than 1.

We continue and study how restrictions on the memory of the system and/or the environment influence the winner and the complexity of the problem. We show that unlike the case of energy games, where memoryless strategies suffice [6, 8], here the situation is more complicated, and is also not symmetric: while infinite memory may be needed for the system, finite-memory strategies are sufficient for the environment. Essentially, this follows from the different winning criteria for the system and the environment, in particular the fact that wins of the environment happen in finite prefixes of the interaction. The memory required for the environment, however, cannot be a-priori bounded. We study the problem of deciding a winner in BBEGs in which the players are restricted to memoryless or finite-memory strategies. We show that such games are not determined, and that when both players are restricted, the problem is Σ_2^P -complete. Also, when only the system is restricted, the problem is strongly related to reachability problems in *vector addition systems with states* (VASS) [18], is decidable, and is in PSPACE for BBEGs in which both the system and the environment have energy vectors of dimension 1.

Finally, we consider settings in which there is an upper bound on the capacity of the bounded resources. Such bounds exist in resources like batteries or disc space. In standard energy games, researchers have extensively studied settings in which the energy level of the system does not exceed a given maximum capacity [6, 15]. This includes both a semantics in which an overflow leads to losing the game and a semantics in which an overflow is truncated. We study this setting in BBEGs, in particular the problem of determining the winner in a BBEG with energy bounds for one of the players. We show that the problem is reducible to deciding standard multi-dimensional energy games, and is thus decidable.

Due to the lack of space, some proofs are omitted and can be found in the full version, in the authors' URLs.

2 Preliminaries

Both-bounded energy game. A *both-bounded energy game* (BBEG, for short) is a game played by two players, Player 1 and Player 2, on a weighted game graph. Each of the players has an energy vector, and the edges of the graph are labeled with updates to those vectors,

applied when the edge is traversed. The vertices of the graph are partitioned into positions that are owned by Player 1 and positions that are owned by Player 2. The game proceeds as follows. A token is placed on the initial position of the game graph. The players move the token along the graph in rounds. In each round, the player that owns the position the token is placed on chooses an edge from this position, and moves the token along it. Each of the players has an initial energy vector, which is updated according to the updates along the edges. The goal of Player 1 is not to run out of energy. The goal of Player 2 is to make Player 1 run out of energy, without running out of energy herself.

Formally, a BBEG is a tuple $G = \langle S_1, S_2, s_{init}, E, d_1, d_2, x_0^1, x_0^2, \tau \rangle$, where S_1 and S_2 are disjoint finite sets of positions, owned by Player 1 and Player 2, respectively. We use S to denote $S_1 \cup S_2$. Position $s_{init} \in S$ is the initial position; $E \subseteq S \times S$ is a set of edges; for $j \in \{1, 2\}$, we have that $d_j \geq 1$ is the *dimension* of Player j and $x_0^j \in \mathbb{N}^{d_j}$ is the *initial energy vector* of Player j . Finally, $\tau : E \rightarrow \mathbb{Z}^{d_1} \times \mathbb{Z}^{d_2}$ is a cost function. Traversing an edge e with $\tau(e) = (x_1, x_2)$, updates to the energy vectors of Player 1 and Player 2 by x_1 and x_2 , respectively. We use $\tau(e)[1]$ and $\tau(e)[2]$ to denote x_1 and x_2 , respectively. We consider non-blocking games, i.e., for every position $s \in S$, there is at least one edge leaving s , thus $\langle s, s' \rangle \in E$, for some $s' \in S$. We call a BBEG with dimensions d_1 for Player 1 and d_2 for Player 2 a (d_1, d_2) -BBEG.

For an integer $n \geq 1$, we denote by $[n]$ the set $\{1, \dots, n\}$. For a vector u in \mathbb{Z}^n and $i \in [n]$, we denote by $u[i]$ the i -th component of u . We define the *size* of G to be the size required for storing the cost function τ , that is $|G| = |E| \cdot (d_1 + d_2) \cdot \log(m)$, where m is the largest integer appearing in some energy update vector. Note that since G is non-blocking, the definition takes the position space into account. Note also the definition assumes that the updates are given in binary.

Given a BBEG G , we define a *run* in G to be an infinite sequence $r = s_1, s_2, \dots \in S^\omega$ such that $s_1 = s_{init}$ and $\langle s_i, s_{i+1} \rangle \in E$ for all $i \geq 1$. For a run $r = s_1, s_2, \dots$ and $n \geq 0$, we denote by r_n the prefix of r up to its n -th position. That is, $r_n = s_1, s_2, \dots, s_n$. We say that n is the *length* of r_n . For $j \in \{1, 2\}$, we say that a prefix r_n *belongs* to Player j if $s_n \in S_j$. We define the *energy level of Player j up to the n -th position in r* to be $e_j(r_n) = x_0^j + \sum_{i=0}^{n-1} \tau(\langle s_i, s_{i+1} \rangle)[j]$. Note that $e_j(r_n)$ is a vector in \mathbb{Z}^{d_j} . For a vector u in \mathbb{Z}^n , We use $u \geq 0$ to indicate that $u[i] \geq 0$ for all $i \in [n]$, and, dually, use $u < 0$ to indicate that $u[i] < 0$ for some $i \in [n]$.

We say that a sequence $c \in S^* + S^\omega$ is a *computation* in G if one of the following holds:

1. c is an infinite run in G , and for every $n \geq 1$, we have that $e_1(c_n) \geq 0$ and $e_2(c_n) \geq 0$.
2. There is $n \geq 1$ such that c is a finite prefix of length n of a run in G , $e_1(c) < 0$ or $e_2(c) < 0$, and for every $m < n$, it holds that $e_1(c_m) \geq 0$ and $e_2(c_m) \geq 0$.

We denote by $comp(G)$ the set of computations in G . For a finite computation $c \in comp(G)$ of length $m \in \mathbb{N}$ and $0 \leq n \leq m$, we denote by c_n the prefix of c up to its n -th position. We denote by $comp(G)$ the set of computations in G , by $pref(G)$ the set of prefixes of $comp(G)$, and by $pref_j(G)$, for $j \in \{1, 2\}$, the set of prefixes that belong to Player j .

Strategies. A strategy for Player j is a function $\gamma_j : pref_j(G) \rightarrow S$, such that for all $p \cdot s \in pref_j(G)$ with $p \in S^*$ and $s \in S_j$, we have that $\langle s, \gamma_j(p \cdot s) \rangle \in E$. That is, a strategy for Player j maps each prefix $p \cdot s$ with $s \in S_j$ to a position that has an incoming edge from s . We say that a computation $c = s_1, s_2, \dots \in comp(G)$ is *consistent* with a strategy γ_j for Player j , if for every $i \geq 1$ such that $c_i \in pref_j(G)$, it holds that $s_{i+1} = \gamma_j(c_i)$. Given two strategies γ_1 for Player 1 and γ_2 for Player 2, we define the *outcome* of γ_1 and γ_2 , denoted $outcome(\gamma_1, \gamma_2)$, to be the single computation that is consistent with both γ_1 and γ_2 . Note that indeed there is exactly one such computation. Note also that since the domain of a strategy may be infinite, a general strategy may require infinite memory.

Winning Conditions. A computation c is *winning for Player 1* if one of the following holds:

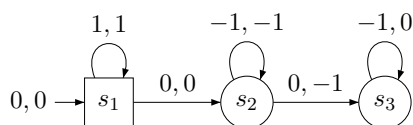
1. Player 1 never runs out of energy. That is, c is infinite. Note that if c is infinite, then for all $n \geq 1$, we have that $e_1(c_n) \geq 0$. Thus, Player 1 manages to keep her energy level non-negative during the infinite computation c .
2. Player 2 runs out of energy before Player 1. That is, there is $n \geq 1$ such that $c = s_1, s_2, \dots, s_n$, it holds that $e_2(c) < 0$, and either $e_1(c) \geq 0$ or $s_{n-1} \in S_2$. We can think of the energy updates along the edges as if traversing an edge leaving position in S_j , for $j \in \{1, 2\}$, updates first the energy vector of Player j , and then updates the energy vector of the other player. Thus, Player 2 runs out of energy before Player 1 if the energy level of Player 2 becomes negative while the energy level of Player 1 is non-negative, or both energy levels become negative together, but as a consequence of a move made by Player 2.

If none of the two conditions above hold, then c is *winning for Player 2*. In other words, c is winning for Player 2 if Player 1 runs out of energy before Player 2. That is, there is $n \geq 1$ such that $c = s_1, s_2, \dots, s_n$, $e_1(c) < 0$, and either $e_2(c) \geq 0$ or $s_{n-1} \in S_1$. Note that while a computation winning for Player 2 is always finite, a computation winning for Player 1 may be either finite or infinite.

A strategy γ_1 is winning for Player 1 if for every strategy γ_2 for Player 2, the computation $\text{outcome}(\gamma_1, \gamma_2)$ is winning for Player 1. Dually, a strategy γ_2 is winning for Player 2 if for every strategy γ_1 for Player 1, the computation $\text{outcome}(\gamma_1, \gamma_2)$ is winning for Player 2. For $j \in \{1, 2\}$, we say that Player j wins in G if she has a winning strategy.

► **Example 1.** Consider the BBEG G in Figure 1. Drawing BBEGs, we describe positions in S_1 and S_2 by circles and squares, respectively. The initial position is marked by an incoming arrow from the initial energy vectors, and edges are labeled with the energy vectors assigned by the cost function. For example, in G both players start with energy level 0, and the transition from s_2 to s_3 does not change the energy level of Player 1, and decreases by 1 the energy level of Player 2.

We show that Player 1 wins in G . Indeed, if Player 2 always takes the loop on s_1 , then Player 1 wins, as the outcome is an infinite computation in which the energy level of Player 1 is always non-negative. Otherwise, Player 2 loops n times in s_1 , for some $n \in \mathbb{N}$, and then moves to s_2 . At this point, the energy level of both players is n . Player 1 can then take the loop on s_2 exactly n times, setting both energy levels back to 0. At this point, Player 1 can take the transition to s_3 and make Player 2 lose, since her energy level drops below 0. ◀



■ **Figure 1** The game graph G .

Determinacy. A game is *determined* if in all instances G of the game, either Player 1 wins in G , or Player 2 wins in G . Since the set of computations that are winning for Player 1 is closed, we have from [23] that BBEGs are determined. Indeed, if Player 2 does not have a winning strategy, one can construct a strategy for Player 1 such that every finite-computation consistent with it is not losing for Player 1. Since the set of winning computations for Player 1 is closed (in the topological sense), this strategy must be winning.

► **Remark 2** (Adding structural assumptions). For simplicity of describing computations and strategies, we define BBEGs without parallel edges. For convenience, we sometimes describe BBEGs with parallel edges (that is, the graph G may have several, yet finitely many, edges between two positions, each with a different update). We sometimes also assume that each transition in the BBEG updates the energy to one player only, or assume that the costs on the transitions are all in $\{-1, 0, 1\}$. As explained in Appendix A.1, these assumptions do not restrict the generality of our results. In particular, while a translation to BBEGs with updates in $\{-1, 0, 1\}$ may involve an exponential blow-up (this is since we define the costs to be given in binary), we consider such BBEGs only in the context of decidability. ◀

3 Deciding BBEGs

In this section we study the problem of determining the winner in a given BBEG. We give a clear border for their decidability: determining the winner in $(1, 1)$ -BBEGs is decidable, yet determining the winner in (d_1, d_2) -BBEGs is undecidable when $d_1 \geq 1$ and $d_2 \geq 2$ or when $d_2 \geq 1$ and $d_1 \geq 2$.

► **Theorem 3.** *The problem of determining the winner in $(1, 1)$ -BBEGs is decidable.*

Proof. We reduce $(1, 1)$ -BBEGs to *one-counter energy games* of dimension 1.

A one-counter energy game of dimension 1 is $A = \langle Q_1, Q_2, \delta, \delta_0 \rangle$, where Q_1 and Q_2 are distinct finite sets of positions owned by Player 1 and Player 2, respectively. We use Q to denote $Q_1 \cup Q_2$. The game A has two transition relations, $\delta \subseteq Q \times \{-1, 0, 1\}^2 \times Q$ and $\delta_0 \subseteq Q \times \{-1, 0, 1\} \times \{0, 1\} \times Q$. A configuration in A is a triple $\langle p, e, c \rangle \in Q \times \mathbb{Z} \times \mathbb{N}$, which describes a position, energy level, and a counter value. The transition relations δ and δ_0 define a relation between successor configurations as follows. A configuration $\langle p', e', c' \rangle$ is successor of configuration $\langle p, e, c \rangle$ iff one of the following holds:

1. $c' \geq 0$ and $\langle p, e' - e, c' - c, p' \rangle \in \delta$.
2. $c = 0$ and $\langle p, e' - e, c', p' \rangle \in \delta_0$.

Note that δ_0 -transitions can be taken only when the value of the counter is 0, and they can not decrease the value. Also, δ -transitions can be taken whenever they do not reduce the value of the counter below 0.

The game proceeds as follows. At each round, the player who owns the current position chooses a transition, and the new configuration is a successor of the current one. Note that during the game, the value of the counter is always non-negative. The game terminates and Player 2 wins if a configuration $\langle p, e, r \rangle$ with $e < 0$ is reached. Player 1 wins every infinite game. It is shown in [1], that given an initial configuration $c = \langle p, e, r \rangle$, determining the winner in A from c is decidable.

Given a $(1, 1)$ -BBEG G , we construct a one-counter energy game A with dimension 1, such that Player 1 wins in G iff Player 1 wins in A . Since determining the winner of one-counter energy games with dimension 1 is decidable [1], we get decidability for $(1, 1)$ -BBEGs.

Let $G = \langle S_1, S_2, s_{init}, E, 1, 1, x_0^1, x_0^2, \tau \rangle$. For simplicity, we assume that each transition in G updates the energy level of only one player, and that the costs on the transitions are numbers in $\{-1, 0, 1\}$ (see Remark 2).

We define $A = \langle Q_1, Q_2, \delta, \delta_0 \rangle$ so that the energy level in A represents the energy of Player 1 in G , and the counter value represents the energy level of Player 2 in G . For that, we define $Q_1 = S_1 \cup \{sink\}$, and $Q_2 = S_2$. Now, let $Q'_1 = \{s \in S_1 : \text{there is } s' \in S \text{ such that } \langle s, s' \rangle \in E \text{ and } \tau(\langle s, s' \rangle) = (0, -1)\}$, and $Q'_2 = \{s \in S_2 : \text{for all } s' \in S \text{ such that } \langle s, s' \rangle \in E, \text{ we have that } \tau(\langle s, s' \rangle) = (0, -1)\}$. That is, Q'_1 is the set of positions from which Player 1 can decrease the energy level of Player 2, and Q'_2 is the set of positions from which Player 2 must decrease her own energy level.

We define $\delta = \{\langle s, \tau(\langle s, s' \rangle)[1], \tau(\langle s, s' \rangle)[2], s' \rangle : \langle s, s' \rangle \in E\} \cup \{\langle sink, 0, 0, sink \rangle\}$ and $\delta_0 = (Q'_1 \cup Q'_2) \times \{0\}^2 \times \{sink\}$. In Appendix A.2, we prove that Player 1 wins in A from $\langle s_{init}, x_0^1, x_0^2 \rangle$ iff Player 1 wins in G . Essentially, this follows from the fact we let Player 1 reach a winning sink whenever she can make Player 2 lose her energy, and we force Player 2 to the sink whenever she runs out of energy. ◀

We now show that the positive result in Theorem 3 is tight.

► **Theorem 4.** *The problem of determining the winner of BBEGs is undecidable. Undecidability holds already for (1,2)-BBEGs or (2,1)-BBEGs, and when the weights on the transitions are all vectors over $\{-1, 0, 1\}$.*

Proof. We start with (1,2)-BBEGs, and show a reduction from *the halting problem of two-counter machines* to our problem. A two-counter machine is a sequence $M = (l_1, \dots, l_n)$ of commands involving two counters x and y . We refer to $\{1, \dots, n\}$ as the *locations* of the machine. The command l_n is the halting command, and each command l_i , for $i < n$, is of one of the following forms, where $c \in \{x, y\}$ is a counter and $1 \leq i, j \leq n$ are locations:

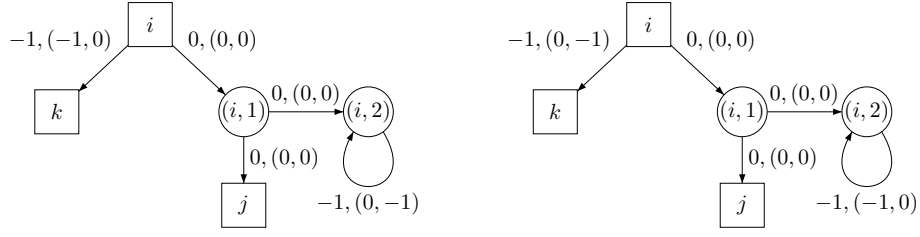
- INC : $c := c + 1$
- GOTO : goto i
- TEST-DEC : if $c = 0$ then goto i else ($c := c - 1$; goto j)

For the TEST-DEC command, we refer to i as the *positive successor* of the command, and refer to j as the *negative successor* of the command. Since we always check whether $c = 0$ before decreasing it, the counters never have negative values. For a two-counter machine M , the question whether M halts is known to be undecidable [25].

Given a machine M , we construct a game G such that M halts iff Player 2 wins in G . The reduction idea is as follows: the dimension of Player 1 is one, and the dimension of Player 2 is two. During a computation in G , the energy level of Player 1 is $x + y$, and the energy level of Player 2 is (x, y) , where x and y are the two counters of M . If M never halts, then both energy levels remain non-negative during the infinite computation, and thus Player 1 wins. If M reaches the halting command, then we reach a losing position for Player 1, so Player 2 wins. We now describe the reduction in detail. Given $M = (l_1, \dots, l_n)$, we construct $G = \langle S_1, S_2, s_{init}, E, 1, 2, 0, 0^2, \tau \rangle$, such that $S_2 = \{1, \dots, n\}$, and $S_1 = L_{td} \times \{1, 2\}$, where $L_{td} \subseteq \{1, \dots, n\}$ is the set of all locations of the TEST-DEC commands in M . The initial energy levels are 0 for Player 1 and $(0, 0)$ for Player 2, reflecting the fact that the counters are initiated to 0. Now, we introduce a gadget for each command l_i as follows.

1. if l_i is $x := x + 1$, then G includes an edge $e = \langle i, i + 1 \rangle$ with $\tau(e) = (1, (1, 0))$.
2. if l_i is $y := y + 1$, then G includes an edge $e = \langle i, i + 1 \rangle$ with $\tau(e) = (1, (0, 1))$.
3. if l_i is goto j , then G includes an edge $e = \langle i, j \rangle$ with $\tau(e) = (0, (0, 0))$.
4. if l_i is if $x = 0$ then goto j else ($x := x - 1$; goto k), then G includes the gadget described in Figure 2 (left).
5. if l_i is if $y = 0$ then goto j else ($y := y - 1$; goto k), then G includes the gadget described in Figure 2 (right).
6. for the halting command, l_n , the game G includes an edge $e = \langle n, n \rangle$ with $\tau(e) = (-1, (0, 0))$.

These transitions are the only transitions G has. We also define s_{init} to be 1; that is, the state corresponding to l_1 .



■ **Figure 2** The gadgets for x -TEST-DEC (left) and y -TEST-DEC (right) commands.

In Appendix A.3 we prove that the reduction is correct, thus M halts iff Player 2 wins in G . For this, we first prove that if a player has a winning strategy, then she also has a winning strategy that follows the instructions. That is, at every step of the computation, the best move for the current player is the one that leads to the state corresponding to the next command to be read according to M . Then, we show that the outcome of strategies that follow the instruction, is such that the energy level of Player 1 stores $x + y$, and the energy level of Player 2 stores (x, y) . Then, as the value of the counters is always non-negative and the position that corresponds to the halting command is losing for Player 1, we get that M halts iff Player 2 wins in G .

The challenging part in the construction and its proof is to construct the TEST-DEC gadgets so that a strategy that follows the instruction is indeed dominating, and that the energy levels indeed maintain the values of the the counters and their sum. Note that excluding positions induced by the TEST-DEC gadgets, all positions in G belong to Player 2. In order to understand the idea behind the gadget, consider for example the x -TEST-DEC gadget, associated with the command if $x = 0$ then goto j else ($x := x - 1$; goto k). As the energy level of Player 2 is (x, y) , taking the transition from position i to position k when $x = 0$ is a losing action for Player 2, as it updates the x -component of her energy level to -1 . Thus, when $x = 0$, a dominating strategy for Player 2 takes the transition from position i to position $(i, 1)$. Then, as the energy level of Player 1 is $x + y$, taking the transition from $(i, 1)$ to $(i, 2)$ when $x = 0$ is a losing action for Player 1. Indeed, after y traversals in the loop in position $(i, 2)$, the energy levels of the players become 0 and $(0, 0)$, causing Player 1 to lose in the next round. Thus, when $x = 0$, a dominating strategy for Player 1 takes the transition from position $(i, 1)$ to position j . In addition, the energy levels of the players does not change when the token moves from position i to j . Similar considerations show that when $x \neq 0$, a dominating strategy for Player 2 takes the transition from position i to position k , which involves an update to the energy levels that corresponds to the decrement of x by 1.

We continue and prove undecidability for $(2, 1)$ -BBEGs. We show a similar reduction from the halting problem of two-counter machines. Take $G = \langle S_1, S_2, s_{init}, E, 1, 2, 0, (0, 0), \tau \rangle$ the BBEG used above, and consider the BBEG $G' = \langle S_2, S_1, s_{init}, E, 2, 1, (0, 0), 0, \tau' \rangle$, where $\tau'(\langle s, s' \rangle) = (\tau(\langle s, s' \rangle)[2], \tau(\langle s, s' \rangle)[1])$ for all $\langle s, s' \rangle \in E$, $s \neq n$, and $\tau'(n, n) = ((-1, 0), 0)$. That is, G' obtained from G by switching the dimensions of the players, their initial energy vectors, the updates on the edges and the sets of positions. Consequently, also in G' , a dominating strategy for the players is consistent with the commands, it implies that the energy level of Player 1 is (x, y) , the energy level of Player 2 is $x + y$, and since the sink n is losing for Player 1, we get that M halts if and only if Player 2 wins in G' . ◀

It is easy to extend Theorem 4 to bigger dimensions, by adding to the energy vectors components whose energy values are not updated during the computation. Thus, by Theorems 3 and 4, determining the winner of (d_1, d_2) -BBEGs is decidable iff $d_1 = d_2 = 1$.

4 BBEGs with finite-memory strategies

In this section we study BBEGs in which the memory used in the strategies of the players is bounded. Following [13], we consider two types of finite-memory strategies. The first type bounds the number of states of a *transducer* that induces the strategy. The second type is *position-based*, and bounds the number of memory states with which we can refine each position of the BBEG. In particular, a *memoryless* strategy is a position-based strategy in which no refinement is allowed. Below we describe the two types formally.

An *I/O-transducer* is a tuple $\mathcal{M} = \langle I, O, Q, q_0, \delta, L \rangle$, for an input alphabet I , an output alphabet O , a finite set of states Q , an initial state $q_0 \in Q$, a transition function $\delta : Q \times I \rightarrow Q$, and a labelling function $L : Q \rightarrow O$. We extend the transition function δ to words in I^* in the expected way, thus $\delta^* : Q \times I^* \rightarrow Q$ is such that for all $q \in Q$, $p \in I^*$, and $i \in I$, we have that $\delta^*(q, \epsilon) = q$, and $\delta^*(q, p \cdot i) = \delta(\delta^*(q, p), i)$. The transducer \mathcal{M} induces a strategy $\gamma_{\mathcal{M}} : I^* \rightarrow O$, where for all $p \in I^*$, we have that $\gamma_{\mathcal{M}}(p) = L(\delta^*(q_0, p))$.

Consider a BBEG $G = \langle S_1, S_2, s_{init}, E, d_1, d_2, x_0^1, x_0^2, \tau \rangle$. Let $S = S_1 \cup S_2$. We say that a strategy γ_j for Player j in G has *finite-memory* if it can be defined by an S/S -transducer (or transducer, when S is clear from the context). The strategy corresponding to \mathcal{M} is defined by $\gamma_j(p) = L(\delta^*(q_0, p))$, for all $p \in \text{pref}_j(G)$. We say that an S/S -transducer $\mathcal{M} = \langle S, S, Q, q_0, \delta, L \rangle$ *refines* G , if the states of \mathcal{M} refine the positions of G . Formally, $Q = S \times M$ for some finite set of *memory states* M , $q_0 = \langle s_{init}, m_0 \rangle$ for some $m_0 \in M$, and for all $s_1, s_2 \in S$ and $m_1 \in M$, it holds that $\delta(\langle s_1, m_1 \rangle, s_2) = \langle s_2, m_2 \rangle$ for some $m_2 \in M$. We say that a strategy for Player j is *memoryless*, if it is induced by a transducer that refines G with $|M| = 1$, thus, $Q = S$. Note that one can refer to a memoryless strategy for Player j as a function $\gamma_j : S_j \rightarrow S$.

For $m_1, m_2 \geq 1$, we say that Player 1 (m_1, m_2) -wins in G , if she has a strategy induced by a transducer with m_1 states, that is winning against all strategies for Player 2 that are induced by a transducer with m_2 states. The definition for Player 2 (m_1, m_2) -winning is similar. All our results on (m_1, m_2) -winning apply also to transducers that refine G (see Remark 15). Note that a general BBEG corresponds to $m_1 = m_2 = \infty$. Of special interest are also settings in which only one of m_1 or m_2 is ∞ , corresponding to BBEGs where only one player has a memory bound.

4.1 Properties of BBEGs with finite-memory strategies

Recall that in energy games with no resource-bounds on the environment, it is sufficient to consider memoryless strategies. We first show that the situation in BBEGs is more complicated, and is also not symmetric: while infinite memory may be needed for Player 1, finite-memory strategies are sufficient for Player 2. Essentially, this follows from the fact that a win of Player 2 is a *co-safety* property: when Player 2 wins, she does so in a finite computation.

► **Theorem 5.** *There is a game G such that Player 1 (∞, ∞) -wins G , but for all $m_1 \geq 1$, Player 2 (m_1, ∞) -wins G . On the other hand, for every BBEG G , if Player 2 (∞, ∞) -wins G , then there is $m_2 \in \mathbb{N}$ such that Player 2 (∞, m_2) -wins G .*

Proof. For the first claim, consider the game G described in Example 1. We saw that Player 1 has a (general) winning strategy. On the other hand, for every strategy γ_1 for Player 1 that is based on a transducer with m_1 states, the (finite-memory) strategy γ_2 for Player 2 that loops $m_1 + 1$ times in s_1 and then moves to s_2 is winning for Player 2 (see proof in the full version). We continue to the second claim. Intuitively, it follows from the

19:10 Energy Games with Resource-Bounded Environments

fact that all the computations in which Player 2 wins are finite. Formally, let G be a BBEG in which Player 2 wins, and let γ_2 be a winning strategy. Consider the unfolding of the game G in which Player 2 plays γ_2 . The unfolding is a tree $T_G^{\gamma_2}$ in which each node is a prefix of a computation that is consistent with γ_2 . Since Player 2 wins, every such a computation is finite, thus every path in $T_G^{\gamma_2}$ is finite. Since the degree of $T_G^{\gamma_2}$ is bounded, we get that $T_G^{\gamma_2}$ is a finite tree, which induces a finite-memory winning strategy for Player 2. ◀

Since finite-memory strategies are sufficient for Player 2 to win, a natural question is whether there is a “bounded-size property” for Player 2’s strategy, in particular whether she can win with a memoryless strategies. Such properties exist in several other settings. For example, in synthesis of an LTL formula ψ , we know that if there is an infinite system that realizes ψ , then there is also a system with at most $2^{2^{|\psi|}}$ states that does it, and the same for the environment [21, 26, 14]. Thus, (∞, ∞) -realizability coincides with $(\infty, 2^{2^{|\psi|}})$ -realizability, $(2^{2^{|\psi|}}, \infty)$ -realizability, and $(2^{2^{|\psi|}}, 2^{2^{|\psi|}})$ -realizability. As we now show, in the case of BBEGs, no bounded-size property exists.

► **Theorem 6.** *There is no computable function $f : \text{BBEGs} \rightarrow \mathbb{N}$ such that for every BBEG G , we have that Player 2 (∞, ∞) -wins G iff Player 2 $(\infty, f(G))$ -wins G .*

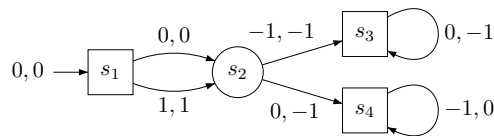
Proof. In Section 4.2, we are going to show that the problem of deciding whether Player 2 (∞, m_2) -wins a BBEG G is decidable for all given BBEGs and bounds $m_2 \in \mathbb{N}$. Hence, the existence of a computable function f would lead to decidability of BBEGs of all dimensions, contradicting Theorem 4. ◀

Recall that BBEGs are determined. As finite-state and memoryless strategies need not be sufficient to winning a BBEG, we now study determinacy of BBEGs when both players have bounds on their memory. Formally, we say that a game is *determined under finite-memory strategies* or *determined under memoryless strategies*, if in all instances G of the game, either Player 1 wins in G , or Player 2 wins in G , when the strategies of both players are restricted to finite-memory or memoryless strategies, respectively. Note that since the restriction applies to both players, the two types of determinacy need not imply each other.

► **Theorem 7.** *BBEGs are not determined under finite-memory or memoryless strategies.*

Proof. We start with finite-memory strategies. Consider the game G described in Example 1. In the full version, we show that when both players are restricted to finite-memory strategies, there is no winning player in G .

We continue to memoryless strategies. Consider the $(1, 1)$ -BBEG G described in Figure 3. In Appendix A.4, we show that there is no winning strategy in G when both players are restricted to play memoryless strategies. ◀



■ **Figure 3** No player has a memoryless winning strategy.

4.2 Deciding BBEGs with finite-memory strategies

In this section we study the problem of deciding the winner in a given BBEG in which at least one player is restricted to finite-memory strategies. We show that the problem is decidable for BBEGs of all dimensions. We start with BBEGs with memoryless strategies and show that deciding whether Player 1 has a memoryless strategy that is winning against every memoryless strategy for Player 2 is Σ_2^P -complete. We first prove the following lemma, about deciding the winner given strategies for the players. The proof, in the full version, is based on the fact that $outcome(\gamma_1, \gamma_2)$ is a simple lasso, and one can determine the winner by analyzing the updates to the energy levels along the prefix and the cycle of the lasso.

► **Lemma 8.** *Given a BBEG and memoryless strategies γ_1 and γ_2 for Player 1 and Player 2, respectively, deciding the winner in $outcome(\gamma_1, \gamma_2)$ can be done in polynomial time.*

Lemma 8 suggests that deciding whether Player 1 has a memoryless strategy that is winning against every memoryless strategy for Player 2 can proceed by guessing a Player 1 strategy and challenging it against a guessed Player 2 strategy. Thus, the problem can be solved by a nondeterministic polynomial-time Turing machine with an oracle to a nondeterministic polynomial-time Turing machine. Below we formalize this intuition and provide also a matching lower bound.

► **Theorem 9.** *Deciding whether Player 1 has a memoryless strategy that is winning against every memoryless strategy for Player 2 is Σ_2^P -complete.*

Proof. The upper bound follows directly from Lemma 8 (see details in Appendix A.5). For the lower bound, we describe a reduction from QBF_2 , the problem of determining the truth of quantified Boolean formulas with two alternations of quantifiers, where the external quantifier is “exists”. Let ψ be a Boolean propositional formula over the variables $x_1, \dots, x_l, y_1, \dots, y_m$, and let $\theta = \exists x_1, \dots, x_l \forall y_1, \dots, y_m \psi$. Also, let $X = \{x_1, \dots, x_l\}, Y = \{y_1, \dots, y_m\}, \bar{X} = \{\bar{x}_1, \dots, \bar{x}_l\}, \bar{Y} = \{\bar{y}_1, \dots, \bar{y}_m\}$, and $Z = X \cup \bar{X} \cup Y \cup \bar{Y}$. By [28], we may assume that ψ is given in 3DNF. That is, $\psi = (z_1^1 \wedge z_1^2 \wedge z_1^3) \vee \dots \vee (z_n^1 \wedge z_n^2 \wedge z_n^3)$, where for all $1 \leq i \leq 3$ and $1 \leq j \leq n$, we have that $z_j^i \in Z$. For $1 \leq j \leq n$, we denote the clause $(z_j^1 \wedge z_j^2 \wedge z_j^3)$ by c_j .

Given a formula $\theta = \exists x_1, \dots, x_l \forall y_1, \dots, y_m \psi$, we construct a (1,1)-BBEG G such that θ is true iff Player 1 wins G with a memoryless strategy. In the game G , we describe the energy levels of the players and updates to the energy levels by bit-vectors in $\{-2, -1, 0, 1, 2, 3\}^n$. Updates to the bit-vectors are done in a bit-wise manner, thus $\langle b_n, b_{n-1}, \dots, b_1 \rangle + \langle b'_n, b'_{n-1}, \dots, b'_1 \rangle = \langle b_n + b'_n, b_{n-1} + b'_{n-1}, \dots, b_1 + b'_1 \rangle$. Our games are defined so that all reachable energy levels are in $\{-2, -1, 0, 1, 2, 3\}^n$. Each bit vector $v = \langle b_n, b_{n-1}, \dots, b_1 \rangle$ represents a single value in \mathbb{Z} , namely $\sum_{i=1}^n b_i \cdot (10)^{i-1}$. For example, the value of $\langle 1, -2, 0, 3 \rangle$ is $3 \cdot 1 + 0 \cdot 10 + (-2) \cdot 100 + 1 \cdot 1000 = 803$. We say that v is positive (negative) iff the value it represents is positive (negative), respectively.

The idea behind the reduction is as follows. Each assignment $g : X \cup Y \rightarrow \{T, F\}$ induces a bit-vector $v_g = \langle b_n, b_{n-1}, \dots, b_1 \rangle \in \{0, 1, 2, 3\}^n$, such that for all $1 \leq i \leq n$, the bit b_i indicates how many literals in c_i are satisfied by the assignment g . Note that this number is indeed in $\{0, 1, 2, 3\}$. For example, take $\psi = (x_1 \wedge x_2 \wedge y_1) \vee (x_1 \wedge x_2 \wedge \bar{y}_1)$, with the assignment g in which $g(x_1) = g(x_2) = T$, and $g(y_1) = F$. Since g satisfies two literals in c_1 and three literals in c_2 , we have that $v_g = \langle 3, 2 \rangle$.

The game G consists of two parts: an *assignment* part, and a *check* part. In the assignment part, Player 1 assigns values to the variables in X , and then Player 2 assigns values to the variables in Y . Together, the players generate an assignment $g : X \cup Y \rightarrow \{T, F\}$, and the

19:12 Energy Games with Resource-Bounded Environments

energy level of both players is updated in the same way, so that by the end of this part, it is v_g . Note that the assignment g satisfies ψ iff the vector v_g contains the bit 3; thus there is $1 \leq i \leq n$ with $b_i = 3$. At the check part, we let Player 2 win if v_g does not contain such a bit. We do this by allowing Player 2 to decrease each bit (in the energy level of both players) by 0, 1 or 2. Accordingly, if no bit in v_g is 3, then Player 2 has a strategy so that by the end of this process, the energy level of the players is represented by the bit-vector 0^n , in which case Player 2 can force a win. On the other hand, if some bit in v_g is 3, then for all strategies of Player 2, at least one bit is not 0 at the end of this process. In this case, Player 2 loses.

In Appendix A.5, we describe the two parts in detail and prove the correctness of the reduction. \blacktriangleleft

Note that since under memoryless strategies, BBEGs are not determined, Π_2^P -completeness for the dual problem does not follow from Theorem 9. In fact, as we show below, the dual problem is also Σ_2^P -complete. The proof, in the full version, is similar to the proof of Theorem 9. In particular, for the lower bound, the game we construct here is obtained from the game constructed there by switching the ownership of positions, switching between the cost functions of the players, and by changing the sink to be a winning position for Player 2.

► **Theorem 10.** *Deciding whether Player 2 has a memoryless strategy that is winning against every memoryless strategy for Player 1 is Σ_2^P -complete.*

We now show that Σ_2^P -completeness holds also when both players are restricted to finite-state strategies. Note that while the considerations are similar to these in the proof of Theorem 9, the lower bound for the memoryless case implies only a lower bound for the finite-memory case with transducers that refine the game G . There, we can use the reduction from the proof of Theorem 9 as is, with $m_1 = |S_1|$ and $m_2 = |S_2|$. For general finite-state strategies, a transducer with $|S_j|$ states, for $j \in \{1, 2\}$, does not necessarily induce a memoryless strategy for Player j . In the proof of the theorem, in the full version, we show that for the specific game G described in the reduction in Theorem 9, Player 1 ($|S_1|, |S_2|$)-wins G iff she wins with a memoryless strategy, and similarly for Player 2 and the game described in the reduction in Theorem 10. Hence, the same reduction can be used.

► **Theorem 11.** *Given a BBEG G and $m_1, m_2 \in \mathbb{N}$ (given in unary), the problems of deciding whether Player 1 (m_1, m_2)-wins and deciding whether Player 2 (m_1, m_2)-wins in G are Σ_2^P -complete.*

Note that the reductions used in Theorems 9, 10, and 11 generate a (1, 1)-BBEG, thus Σ_2^P -hardness holds already for them.

We continue and consider BBEGs in which only Player 1 has a memory bound. We show that the setting is strongly related to *vector addition systems with states* (VASS), defined below.

For $d \geq 1$, a d -VASS is a finite \mathbb{Z}^d -labeled directed graph $V = \langle Q, T \rangle$, where Q is a finite set of *states*, and $T \subseteq Q \times \mathbb{Z}^d \times Q$ is a finite set of *transitions*. The set of *configurations* of V is $C = Q \times \mathbb{N}^d$. For a pair of configurations $\langle p_1, v_1 \rangle, \langle p_2, v_2 \rangle \in C$ and $t = \langle p_1, z, p_2 \rangle \in T$ such that $v_2 = v_1 + z$, we write $\langle p_1, v_1 \rangle \xrightarrow{t} \langle p_2, v_2 \rangle$. For $c, c' \in C$ we write $c \xrightarrow{*} c'$ if $c = c'$, or if there is $m \geq 1$ such that $c_0 \xrightarrow{t_1} c_1 \xrightarrow{t_2} \dots \xrightarrow{t_m} c_m$, for some $t_1, \dots, t_m \in T$ and $c_0, \dots, c_m \in C$, with $c_0 = c$ and $c_m = c'$. That is, $c \xrightarrow{*} c'$ indicates that there is a sequence of successive configurations from c to c' in V , and the vector is non-negative in all the configurations along the sequence. The d -VASS *reachability problem* is to decide, given a d -VASS V and configurations $c, c' \in C$, whether $c \xrightarrow{*} c'$.

We are going to reduce questions about (m_1, ∞) -winning in BBEGs to questions about VASSs. The underlying idea is as follows. First, once we bound the memory of Player 1, we can guess a transducer that generates her strategy. The product of the BBEG with such a transducer results in a *one-player BBEG*, in which all positions belong to Player 2. As the evolution of a one-player BBEG does not involve alternation between players, we can model it by a VASS. Essentially, the configurations of the VASS correspond to positions in the game along with energy vectors of the players. The winning condition in the BBEG induces requirement on the VASS, as formalized in the following lemma (see proof in Appendix A.6).

► **Lemma 12.** *Given a (d_1, d_2) -BBEG G in which all the positions are owned by Player 2, the winner in G can be decided by solving at most d_1 instances of $(d_2 + 1)$ -VASS reachability.*

We now use Lemma 12 in order to decide whether Player 1 (m_1, ∞) -wins a given BBEG.

► **Theorem 13.** *Given a BBEG G and $m_1 \in \mathbb{N}$, determining whether Player 1 (m_1, ∞) -wins G is decidable.*

Proof. Let G be a (d_1, d_2) -BBEG, for some $d_1, d_2 \geq 1$, and consider a transducer T with state space Q of size m_1 that maintains a strategy for Player 1. Let $S = S_1 \cup S_2$ be the state space of G . When Player 1 follows T , the possible outcomes of the game are embedded in the product $G \times T$. The product has state space $S \times Q$. Each positions in $S_1 \times Q$ has a single successor: its S -component is determined by the output function of T and its Q -component is determined by the transition function of T . Therefore, we can refer to the product $G \times T$ as a BBEG all whose positions belong to Player 2. The updates on the edges of the product BBEG are induced by these in G , and so it is a (d_1, d_2) -BBEG. By Lemma 12, determining the winner in $G \times T$ can be reduced to solving d_1 instances of $(d_2 + 1)$ -VASS-reachability, which is decidable [22].

It follows that determining whether Player 1 (m_1, ∞) -wins G can be decided by going over the finitely many candidates transducers T of size m_1 , and applying the above check to each of them. ◀

► **Remark 14 (Complexity).** While Theorems 13 only refer to decidability, known complexity results on VASS can be used in order to give complexity upper bounds in some cases. Specifically, as 2-VASS reachability is PSPACE-complete [4], and the candidate transducers T are polynomial in m_1 , we get that determining whether Player 1 (m_1, ∞) -wins G is decidable in PSPACE for $(1, 1)$ -BBEGs with m_1 given in unary. ◀

We note that while similar considerations can be used in order to decide whether Player 2 (∞, m_2) -wins a given BBEG, for $m_2 \in \mathbb{N}$ (see proof in Appendix A.7), the latter does not provide a solution to the problem of deciding whether Player 1 (∞, m_2) -wins a given BBEG, which we leave open. Indeed, BBEGs are not (∞, m_2) -determined, in the sense that there is a BBEG G and $m_2 \in \mathbb{N}$ such that neither Player 1 (∞, m_2) -wins nor Player 2 (∞, m_2) -wins G . For example, by switching the vertices owned by Player 1 and Player 2 in the BBEG appearing in Figure 3, we get a BBEG such that Player 1 does not (∞, m_2) -wins for all $m_2 \in \mathbb{N}$, and Player 2 does not wins with a memoryless strategy, and in particular does not $(\infty, 1)$ -wins.

Finally, we note that, unsurprisingly, even when we fix the size of the strategy of Player 2, the size of the strategy required for Player 1 to win depends on both the number of positions in the game and the updates in its transitions, inducing a strict hierarchy. Specifically, in the full version, we show that for all $m_1 \in \mathbb{N}$, there is a BBEG G_{m_1} with 3 states as well as a BBEG G'_{m_1} in which all updates are in $\{-1, 0, 1\}$, such that Player 1 $(m_1 + 2, 0)$ -wins G_{m_1} and G'_{m_1} , yet Player 2 $(m_1 + 1, 0)$ -wins G_{m_1} and G'_{m_1} . Similar results can be shown for the size of the strategy for Player 2.

► **Remark 15** (From general to position-based strategies). Our positive decidability and complexity results are based on going over candidate strategies for the players. By restricting attention to strategies that refine the BBEG, these results apply also to position-based finite-state strategies. In addition, our lower bounds apply already for memoryless strategies, and so apply also for position-based finite-state strategies. ◀

5 BBEG with Bounded Energy Capacities

So far we studied BBEGs in which the players must keep their energy level non-negative, but there is no upper bound on the energy they may accumulate. This corresponds to systems in which there is no bound on the capacity of the energy resource. In many cases (c.f., battery, disc space), such a bound exists. In this section we study the problem of determining the winner in BBEGs in which one of the players has a bounded energy capacity. We consider both a semantics in which an overflow leads to losing the game (losing semantics, for short) and a semantics in which an overflow is truncated (truncated semantics, for short).

Formally, a *one-player-bounded BBEG* is $G = \langle S_1, S_2, s_{init}, E, d_1, d_2, x_0^1, x_0^2, \tau, j, b \rangle$, which extends a BBEG by specifying a player $j \in \{1, 2\}$ and a bound vector $b \in \mathbb{Z}^{d_j}$. In the losing semantics, the definition of a winning computation in a one-player-bounded BBEG is similar to the definition in the case of a BBEG, except that the requirement for the energy to stay non-negative is replaced, for Player j , by a requirement to stay both non-negative and below the bound b . Formally, a computation c that is winning for Player j has to satisfy, in addition to the winning condition of a BBEG, the requirement $e_j(c_n)[i] \leq b[i]$ for all $n \geq 1$ and $i \in [d_j]$. In the truncated semantics, the winning condition is as in the underlying BBEG, yet the energy level of Player j up to the n -th position in a run $r = s_1, s_2, \dots$ is defined inductively for all $i \in [d_j]$ as follows: $e_j(r_n)[i] = \min\{b[i], e_j(r_{n-1})[i] + \tau(\langle s_i, s_{i+1} \rangle)[j][i]\}$, where $e_j(r_0)[i] = x_0^j[i]$.

In Theorem 16 below we show that the problem of deciding whether Player 1 wins a one-player-bounded BBEG is decidable for BBEGs of all dimensions. Essentially, our solution is based on expanding the position space of the game to maintain the energy level of Player j . Consequently, the cost function in the transitions updates the energy level of the other player only. When $j = 2$, thus the energy of Player 2 is bounded, we are left with updates to the energy level of Player 1. Thus, we obtain a standard multi-dimensional energy game, except that we add a sink that is winning for Player 1 and corresponds to positions in which the energy level of Player 2 is negative or, in the losing semantics, is above the bound b .

When $j = 1$, thus the energy of Player 1 is bounded, we obtain a multi-dimensional energy game in which transitions update the energy level Player 2 only. The game contains a sink, which is losing for Player 1, and Player 2 wins the game if she can reach the sink without her energy becoming negative. Thus, the setting is similar to that of multi-dimensional reachability energy games. By [16], one-dimensional energy-reachability games can be decided in $\text{NP} \cap \text{coNP}$, and so our proof boils down to extending their algorithm to the multi-dimensional case. The full details can be found in Appendix A.8.

► **Theorem 16.** *The problem of determining whether Player 1 wins a one-player-bounded BBEG is decidable.*

► **Remark 17** (Bounding only some of the energy components). In the multi-dimensional setting, we can consider games in which each player has energy bounds for some of the components in her energy vector. It is easy to see for $d_1, d_2 \geq 1$ determining the winner of a (d_1, d_2) -BBEG is decidable iff each player has at most one unbounded component. Indeed, one can extend the position space of a BBEG to remember the value of the $(d-1) + (d-1)$ bounded components, and then deciding $(1, 1)$ -BBEG. ◀

References

- 1 P.A. Abdulla, M.F. Atig, P. Hofman, R. Mayr, K.N. Kumar, and P. Totzke. Infinite-state energy games. In *Proc. 29th IEEE Symp. on Logic in Computer Science*, pages 7:1–7:10. ACM, 2014.
- 2 G. Amram, S. Maoz, O. Pistiner, and J. O. Ringert. Efficient algorithms for omega-regular energy games. In *24th International Symposium on Formal Methods*, volume 13047 of *Lecture Notes in Computer Science*, pages 163–181. Springer, 2021.
- 3 R. Bloem, K. Chatterjee, and B. Jobstmann. Graph games and reactive synthesis. In *Handbook of Model Checking.*, pages 921–962. Springer, 2018.
- 4 M. Blondin, A. Finkel, S. Göller, C. Haase, and P. McKenzie. Reachability in two-dimensional vector addition systems with states is pspace-complete. In *Proc. 30th IEEE Symp. on Logic in Computer Science*, pages 32–43. IEEE, 2015.
- 5 A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
- 6 P. Bouyer, U. Fahrenberg, K.G. Larsen, N. Markey, and J. Srba. Infinite runs in weighted timed automata with energy constraints. In *6th International Conference on Formal Modeling and Analysis of Timed Systems*, volume 5215 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2008.
- 7 T. Brázdil, P. Jančar, and A. Kučera. Reachability games on extended vector addition systems with states. In *Proc. 37th Int. Colloq. on Automata, Languages, and Programming*, pages 478–489. Springer, 2010.
- 8 A. Chakrabarti, L. de Alfaro, T.A. Henzinger, and M. Stoelinga. Resource interfaces. In *International Workshop on Embedded Software*, pages 117–133. Springer, 2003.
- 9 K. Chatterjee and L. Doyen. Energy parity games. *Theoretical Computer Science*, 458:49–60, 2012.
- 10 K. Chatterjee, L. Doyen, T.A. Henzinger, and J-F. Raskin. Generalized mean-payoff and energy games. In *Proc. 30th Conf. on Foundations of Software Technology and Theoretical Computer Science*, volume 8 of *LIPICs*, pages 505–516. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010.
- 11 K. Chatterjee, A.K. Goharshady, and Y. Velner. Quantitative analysis of smart contracts. In *27th European Symposium on Programming Languages and Systems*, volume 10801 of *Lecture Notes in Computer Science*, pages 739–767. Springer, 2018.
- 12 J-B. Courtois and S. Schmitz. Alternating vector addition systems with states. In *39th Int. Symp. on Mathematical Foundations of Computer Science*, pages 220–231. Springer, 2014.
- 13 R. Ehlers. Symbolic bounded synthesis. In *Proc. 22nd Int. Conf. on Computer Aided Verification*, volume 6174 of *Lecture Notes in Computer Science*, pages 365–379. Springer, 2010.
- 14 E.A. Emerson. Automata, tableaux, and temporal logics. In *Proc. Workshop on Logic of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 79–87. Springer, 1985.
- 15 U. Fahrenberg, L. Juhl, K. G. Larsen, and J. Srba. Energy games in multiweighted automata. In *8th International Colloquium on Theoretical Aspects of Computing*, volume 6916 of *Lecture Notes in Computer Science*, pages 95–115. Springer, 2011.
- 16 L. Hélouët, N. Markey, and R. Raha. Reachability games with relaxed energy constraints. *Information and Computation*, page 104806, 2021.
- 17 T.A. Henzinger. From Boolean to quantitative notions of correctness. In *Proc. 37th ACM Symp. on Principles of Programming Languages*, pages 157–158, 2010.
- 18 J.E. Hopcroft and J-J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8:135–159, 1979.
- 19 M. Jurdzinski, R. Lazic, and S. Schmitz. Fixed-dimensional energy games are in pseudo-polynomial time. In *Proc. 42nd Int. Colloq. on Automata, Languages, and Programming*, volume 9135 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 2015.

- 20 M. Jurdziński, R. Lazić, and S. Schmitz. Fixed-dimensional energy games are in pseudo-polynomial time. In *Proc. 42nd Int. Colloq. on Automata, Languages, and Programming*, pages 260–272. Springer, 2015.
- 21 O. Kupferman, Y. Lustig, M.Y. Vardi, and M. Yannakakis. Temporal synthesis for bounded systems and environments. In *Proc. 28th Symp. on Theoretical Aspects of Computer Science*, pages 615–626, 2011.
- 22 J. Leroux and S. Schmitz. Reachability in vector addition systems demystified. In *Proc. 35th IEEE Symp. on Logic in Computer Science*, 2015.
- 23 D.A. Martin. Borel determinacy. *Annals of Mathematics*, 65:363–371, 1975.
- 24 A. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.
- 25 M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1 edition, 1967.
- 26 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.
- 27 S. Schewe and B. Finkbeiner. Bounded synthesis. In *5th Int. Symp. on Automated Technology for Verification and Analysis*, volume 4762 of *Lecture Notes in Computer Science*, pages 474–488. Springer, 2007.
- 28 C. Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.

A Proofs

A.1 Proof of the assumptions in Remark 2

It is easy to see that every BBEG with parallel edges has an equivalent BBEG of linear size without parallel edges. Indeed, let $s, t \in S$ be two positions and let A be the set of edges from s to t , with updates $l_1, \dots, l_{|A|}$. We can add new positions $s_1^{(s,t)}, \dots, s_{|A|}^{(s,t)}$, and edges $\{(s, s_i^{(s,t)}) : 1 \leq i \leq |A|\} \cup \{(s_i^{(s,t)}, t) : 1 \leq i \leq |A|\}$ instead of the parallel edges, with updates $\tau(\langle s, s_i^{(s,t)} \rangle) = l_i$ and $\tau(\langle s_i^{(s,t)}, t \rangle) = (0^{d_1}, 0^{d_2})$, for all $1 \leq i \leq |A|$.

It is also easy to see that every BBEG with has an equivalent BBEG of linear size in which each transition updates the energy to one player only. The only nontrivial issue in the decomposition of a transition is that we should first update the energy of the player that owns the source position. Thus, an edge leaving $s \in S_1$, labeled with (x_1, x_2) and leading to $t \in S$, can be replaced the two edges $\langle s, u_{s,t} \rangle$ with $\tau(\langle s, u_{s,t} \rangle) = (x_1, 0^{d_2})$, and $\langle u_{s,t}, t \rangle$ with $\tau(\langle u_{s,t}, t \rangle) = (0^{d_1}, x_2)$, for a new position $u_{s,t}$. For the case $s \in S_2$, the new edges update first the energy of Player 2.

Finally, we can translate a BBEG to a BBEG in which the updates on the transitions are all in $\{-1, 0, 1\}$. We describe the translation for $(1, 1)$ -BBEGs. A similar translation works for BBEGs of higher dimensions. Indeed, one can first convert a BBEG to one in which every transition updates the energy to one player only, as described above, and then replace an edge labeled with $(x_1, 0^{d_2})$ with $|x_1|$ edges that update x_1 to the energy of Player 1, while not affecting the energy of Player 2. Similarly, we can handle edges labeled with $(0^{d_1}, x_2)$. Note, however, that since we define the size of a BBEG with the costs on the edges of given in binary, the resulting BBEG is of size exponential in the size of the original BBEG. Since we consider BBEGs with updates in $\{-1, 0, 1\}$ only in the context of decidability, this does not affect our results.

A.2 Correctness of the upper-bound reduction in Theorem 3

We prove that Player 1 wins in A from $\langle s_{init}, x_0^1, x_0^2 \rangle$ iff Player 1 wins in G . First, an infinite computation in G induces an infinite game in A that never reaches the sink. Also, a finite computation in G in which Player 1 runs out of energy before Player 2, induces a finite game in A that is losing for Player 1. Finally, a finite computation in G that reaches a configuration in which Player 1 can make Player 2 lose, or Player 2 has no choice but to lose her energy, reaches a position in $Q'_1 \cup Q'_2$ with the energy level of Player 2 being 0. The corresponding game in A reaches $Q'_1 \cup Q'_2$ with the counter being 0. If the current position is in Q'_1 , Player 1 can use the δ_0 -transition to the sink and stay there forever. If the current position is in Q'_2 , Player 2 has no choice but to use the δ_0 -transition and reach the sink. Thus, Player 1 wins in G iff Player 1 can force an infinite game in A .

A.3 Correctness of the lower-bound reduction in Theorem 4

We prove that the reduction is correct, i.e., the machine M halts iff Player 2 wins in G . We describe a computation of M by an infinite sequence $f = f_0, f_1, f_2, \dots \in (\{1, \dots, n\} \times \mathbb{N} \times \mathbb{N})^\omega$, such that $f_0 = (1, 0, 0)$ and for all $i \geq 1$, we have that $f_i[1]$ is the location of the i -th command in the computation, and $f_i[2]$ and $f_i[3]$ are the values of the counters x and y , respectively, after reading that command. If for some $i \geq 0$ we have that $f_i[1] = n$, then $f_{i+1} = f_i$. Consider a computation $\pi \in \text{comp}(G)$, and let $v = v_0, v_1, \dots$ be the projection of π on S_2 . We say that π is *consistent* if for all $i \in \mathbb{N}$, we have that $e_1(v_i) = f_i[2] + f_i[3]$ and $e_2(v_i) = (f_i[2], f_i[3])$. That is, π is consistent if the energy level of Player 1 stores $x + y$, and the energy level of Player 2 stores $\langle x, y \rangle$.

First, we show that if a player has a winning strategy, then she also has a winning strategy that follows the instructions. That is, at every step of the computation, the best move for the current player is the one that leads to the state corresponding to the next command to be read according to M . For $c \in \{x, y\}$, denote by $L_{td}^c \subseteq L_{td}$ the set of locations of TEST-DEC commands that examine counter c . Note that excluding positions induced by the TEST-DEC gadgets, all positions in G belong to Player 2, and that the position corresponding to the halting command is losing for Player 1. Also note that all positions except some positions in the TEST-DEC gadgets are deterministic, that is, have a single transition leaving them.

Recall that for a consistent prefix p , the energy level $e_2(p)$ stores $\langle x, y \rangle$. Accordingly, for $c \in \{x, y\}$, we use $e_2^c(p)$ to refer to $e_2(p)[1]$ when $c = x$, and to refer to $e_2(p)[2]$ when $c = y$. Also, we use \bar{c} to refer to y when $c = x$, and to refer to x when $c = y$.

We say that a strategy γ_1 for Player 1 is *consistent* if for every $p \in \text{pref}_1(G)$ ending in position $(i, 1)$ for $i \in L_{td}^c$, if $e_1(p) > e_2^{\bar{c}}(p)$, then $\gamma_1(p) = (i, 2)$, and if $e_1(p) \leq e_2^{\bar{c}}(p)$, then $\gamma_1(p) = j$, for j that is the positive successor of l_i . Similarly, we say that a strategy γ_2 for Player 2 is *consistent* if for every $p \in \text{pref}_2(G)$ ending in position $i \in L_{td}^c$, if $e_2^c(p) = 0$, then $\gamma_2(p) = (i, 1)$, and if $e_2^c(p) > 0$, then $\gamma_2(p) = k$, for k that is the negative successor of l_i .

Note that every player has a unique consistent strategy. Let γ_1 and γ_2 be the consistent strategies for Player 1 and Player 2, respectively. Let $r = \text{outcome}(\gamma_1, \gamma_2)$. We argue that r is consistent. Let $v = v_0, v_1, \dots$ be the projection of r on S_2 . We prove that for all $i \in \mathbb{N}$, it holds that $e_1(v_i) = f_i[2] + f_i[3]$ and $e_2(v_i) = (f_i[2], f_i[3])$. The proof proceeds by an induction on i . Initially, $f_0 = (1, 0, 0)$, and indeed for all runs in G , the initial position is 1 and the initial energy levels are 0 for Player 1 and $(0, 0)$ for Player 2.

Let $m \geq 1$, and assume that the claim holds for all $0 \leq i < m$. If $v_m \notin L_{td}$, then Player 2 has a single successor, which corresponds to $f_{m+1}[1]$, and the energy levels are updated correctly. We now consider the case $v_m \in L_{td}^x$. Denote $f_{m-1}[1] = i$, $f_{m-1}[2] = x$, and $f_{m-1}[3] = y$. By the induction hypothesis, we have that $e_1(v_{m-1}) = x + y$ and $e_2(v_{m-1}) = (x, y)$. We distinguish between two cases:

19:18 Energy Games with Resource-Bounded Environments

1. If $x = 0$, then following γ_2 , Player 2 chooses to go to position $(i, 1)$. This move does not affect the energy level. Since $x = 0$, then $x + y = y$, and following γ_1 , Player 1 chooses to go to position j that is the positive successor of l_i . This transition does not affect the energy levels either. So, we have that $v_m = j$, $e_1(v_m) = x + y$, and $e_2(v_m) = (x, y)$, as required.
2. If $x > 0$, then, following γ_2 , Player 2 chooses to go to position k that is the negative successor of l_i . This transition decreases by one the the energy level of Player 1 and the first component in the energy level of Player 2. So, $v_m = k$, $e_1(v_m) = x + y - 1$, and $e_2(v_m) = (x - 1, y)$, as required.

The case where $i \in L_{td}^y$ is similar.

Let γ_1, γ_2 be the consistent strategies for Player 1 and Player 2, respectively, and denote $r = \text{outcome}(\gamma_1, \gamma_2)$. We show that if Player 2 plays a strategy δ_2 that is not consistent, then she loses against the consistent strategy γ_1 of Player 1.

Assume that Player 1 plays γ_1 and Player 2 plays δ_2 , which is not consistent. Let m be the minimal index in $\text{outcome}(\gamma_1, \delta_2)$ that deviates from r . That is, m is the minimal index t such that $\delta_2(r_t) \neq \gamma_2(r_t)$. Let i be the last position in r_m . Since all positions in $S_2 \setminus L_{td}$ are deterministic, it must be that $i \in L_{td}$. Assume that $i \in L_{td}^x$. Then, either $e_2(r_m)[0] = 0$ and $\delta_2(r_m) = k$, for k that is the negative successor of l_i , or $e_2(r_m)[0] > 0$ and $\delta_2(r_m) = (i, 1)$. Since m is minimal and r is consistent, we get that $e_1(r_m) = x + y$ and $e_2(r_m) = (x, y)$ for some $x, y \in \mathbb{N}$. If $x = 0$ and $\delta_2(r_m) = k$, then the first component in the energy level of Player 2 is decreased below 0, so she loses. If $x > 0$ and $\delta_2(r_m) = (i, 1)$, then according to γ_1 , Player 1 chooses from $(i, 1)$ to go to $(i, 2)$. Since $x + y > y$, Player 1 wins at the sink $(i, 2)$. Hence, $\text{outcome}(\gamma_1, \delta_2)$ is winning for Player 1. The case where $i \in L_{td}^y$ is similar.

Since δ_2 is not winning for every $\delta_2 \neq \gamma_2$, we get that if Player 2 wins, her winning strategy must be consistent.

Now, we show that if Player 1 wins, then she can win with γ_1 . Assume that Player 1 has a winning strategy $\delta_1 \neq \gamma_1$. We show that γ_1 is winning for Player 1 too. We already showed that $\text{outcome}(\gamma_1, \delta_2)$ is winning for Player 1 for every $\delta_2 \neq \gamma_2$. It is left to show that $\text{outcome}(\gamma_1, \gamma_2)$ is winning for Player 1. Let m be the minimal index t in $\text{outcome}(\delta_1, \gamma_2)$ such that $\delta_1(r_t) \neq \gamma_1(r_t)$. Since all positions in $S_1 \setminus (L_{td} \times \{1\})$ are deterministic, it must be that r_m ends in position $i \in L_{td} \times \{1\}$. Assume that $i \in L_{td}^x \times \{1\}$. Then, either $e_1(r_m) > e_2(r_m)[2]$ and $\delta_1(r_m) = j$ for j that is the positive successor of l_i , or $e_1(r_m) \leq e_2(r_m)[2]$ and $\delta_1(r_m) = (i, 2)$. Since m is minimal and r is consistent, we get that $e_1(r_m) = x + y$ and $e_2(r_m) = (x, y)$ for some $x, y \in \mathbb{N}$. If it is the case that $e_1(r_m) > e_2(r_m)[2]$, we have that $\delta_1(r_m) = j$ and $\gamma_1(r_m) = (i, 2)$. By going to $(i, 2)$, since $x + y > y$, we get that Player 2 loses at $(i, 2)$. Hence, $\text{outcome}(\gamma_1, \gamma_2)$ is winning for Player 1. Also, it cannot be the case that $e_1(r_m) \leq e_2(r_m)[2]$ and $\delta_1(r_m) = (i, 2)$: since $x + y \leq y$, we get that Player 1 loses at $(i, 2)$, in contradiction to the fact that δ_1 is winning. The case where $i \in L_{td}^y \times \{1\}$ is similar.

By the above, if Player 2 has a winning strategy, it must be consistent, and if Player 1 wins, her consistent strategy is winning. Therefore, the question of determining the winner in G is reduced to determining the winner of $\text{outcome}(\gamma_1, \gamma_2)$. When both players play their consistent strategies, we have that the energy levels are updated according to the values of the counters in f . Since the value of every counter is non-negative during the run, so are the energy levels of the players during the computation. Since the state corresponding to the HALT command is a rejecting sink for Player 1, we have that if M halts, then Player 2 wins in G . Otherwise, the energy levels of both players, in particular Player 1, remain non-negative during the infinite computation, and Player 1 wins.

A.4 Proof of Theorem 7 – memoryless strategies

We prove that when both players are restricted to memoryless strategies, there is no winning player in the BBEG G described in Figure 3.

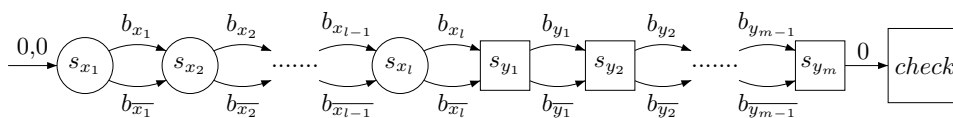
First, we show that for every memoryless strategy γ_1 for Player 1, there is a memoryless strategy γ_2 for Player 2 such that $outcome(\gamma_1, \gamma_2)$ is winning for Player 2. Note that Player 1 has to choose an outgoing edge only from s_2 . Let us consider a memoryless strategy γ_1 for Player 1. If $\gamma_1(s_2) = s_3$, then for the strategy γ_2 for Player 2 that chooses to go from s_1 to s_2 by the edge labeled $(0, 0)$, it holds that $outcome(\gamma_1, \gamma_2)$ is winning for Player 2: when the computation reaches s_2 , the energy level of Player 1 is 0, so the transition to s_3 makes her lose. If $\gamma_1(s_2) = s_4$, then the strategy γ_2 for Player 2 that chooses to go from s_1 to s_2 by the edge labeled $(1, 1)$ is such that $outcome(\gamma_1, \gamma_2)$ is winning for Player 2: when the computation reaches s_4 , the energy level of Player 2 is 1, so she can pay 1 to reach s_5 , which is a rejecting sink for Player 1.

We continue and show that for every strategy γ_2 for Player 2 (in particular a memoryless strategy), there is a memoryless strategy γ_1 for Player 1 such that $outcome(\gamma_1, \gamma_2)$ is winning for Player 1. Consider a strategy γ_2 for Player 2. If by following γ_2 Player 2 goes from s_1 to s_2 by the edge labeled $(0, 0)$, then a memoryless strategy γ_1 for Player 1 with $\gamma_1(s_2) = s_4$ is such that $outcome(\gamma_1, \gamma_2)$ is winning for Player 1: the energy level of Player 2 becomes negative at the transition to s_4 . If by following γ_2 Player 2 goes from s_1 to s_2 by the edge labeled $(1, 1)$, then the strategy γ_1 for Player 1 with $\gamma_1(s_2) = s_3$ is such that $outcome(\gamma_1, \gamma_2)$ is also winning for Player 1: until the computation reaches s_3 , the energy level of Player 1 remains non-negative, and s_3 is a winning sink for Player 1.

A.5 Missing details in the proof of Theorem 9

For the upper bound, consider a BBEG $G = \langle S_1, S_2, s_{init}, E, d_1, d_2, x_0^1, x_0^2, \tau \rangle$. Memoryless strategies for the players can be represented by polynomial-length strings. Then, given a memoryless strategy γ_1 for Player 1, the problem of checking whether there is a memoryless strategy γ_2 for Player 2 such that $outcome(\gamma_1, \gamma_2)$ is winning for Player 2 is in NP. Indeed, given a memoryless strategy γ_1 for Player 1, we can decide by a non-deterministic Turing Machine whether there is a memoryless strategy γ_2 for Player 2 such that $outcome(\gamma_1, \gamma_2)$ is winning for Player 2, by guessing γ_2 and applying Lemma 8. So, deciding whether there is a memoryless strategy γ_1 for Player 1 such that for every memoryless strategy γ_2 for Player 2 it holds that $outcome(\gamma_1, \gamma_2)$ is winning for Player 1, can be done by a nondeterministic polynomial-time Turing machine with an oracle to a nondeterministic polynomial-time Turing machine, and we are done.

We continue to the lower bound and describe the two parts of the BBEG in detail. For convenience, we describe the BBEG with parallel edges (see Remark 2). Both players start with the initial energy level 0, which is represented by the bit-vector 0^n . The assignment part is described in Figure 4.

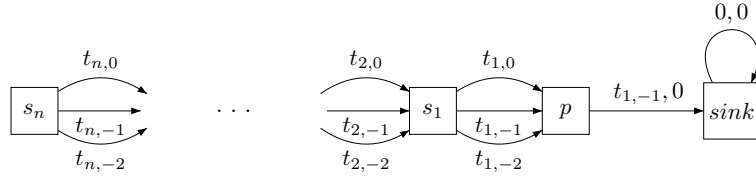


■ **Figure 4** The assignment part.

19:20 Energy Games with Resource-Bounded Environments

For every literal $z \in Z$, let $b_z = \langle b_z^0, \dots, b_z^1 \rangle \in \{0, 1\}^n$ describe how the bit-vector v_g should be updated when z is assigned T . That is, for all $1 \leq i \leq n$, if the literal z appears in the clause c_i , then $b_z^i = 1$, and otherwise $b_z^i = 0$. For our example formula $(x_1 \wedge x_2 \wedge y_1) \vee (x_1 \wedge x_2 \wedge \bar{y}_1)$, we have $b_{\bar{x}_1} = \langle 0, 0 \rangle$ and $b_{y_1} = \langle 0, 1 \rangle$. Since in this part, the energy levels of both players are updated in the same way, we label each transition in the figure by a single update. As described in the figure, first Player 1 assigns values to the variables in X and then Player 2 assigns values to the variable in Y . An assignment is reflected in the energy levels of both players being updated according to the literal that is chosen. In our example, if from s_{y_1} Player 2 chooses the transition that corresponds to assigning T to y_1 , then the energy level of both players is increased by $\langle 0, 1 \rangle$.

We continue to the check part, where all the positions belong to Player 2. The check part is described in Figure 5. Here too, except for the transition to the sink, the updates to the energy levels of Player 1 and Player 2 coincide, and we label the transitions in the figure by a single update.



■ **Figure 5** The check part.

For every $1 \leq i \leq n$ and $d \in \{0, -1, -2\}$, let $t_{i,d} = 0^{i-1} \cdot \{d\} \cdot 0^{n-(i+1)}$. That is, all the bits in $t_{i,d}$ are 0, except for the i -th bit, which is d . As described in Figure 5, the check part consists of a chain of positions s_i , for $n \geq i \geq 1$, where from s_{i+1} Player 2 proceeds to s_i while updating the energy levels by $t_{i,0}$, $t_{i,-1}$, or $t_{i,-2}$. Then, from position p , there is a single transition with updates $t_{1,-1}, 0$ to the energy levels. Thus, the least significant bit of the energy level of Player 1 is decreased by 1, and the energy level of Player 2 is not changed.

We now prove that θ is true iff Player 1 wins in G with a memoryless strategy.

Assume first that θ is true. Then, there is an assignment f_X for X such that for every assignment f_Y for Y , we have that ψ is true under $f_X \cup f_Y$. We show that there is a memoryless strategy for Player 1 that is winning against every (not necessarily memoryless) strategy for Player 2. An assignment f_X for X induces a memoryless strategy γ_{f_X} for Player 1 in which for every variable x_i such that $f_X(x_i) = T$, Player 1 chooses from s_{x_i} the transition labeled b_{x_i} , and for every variable x_i such that $f_X(x_i) = F$, Player 1 chooses from s_{x_i} the transition labeled $b_{\bar{x}_i}$. We show that γ_{f_X} is winning for Player 1. Let γ be a strategy for Player 2, and let f_Y be the assignment for Y induced by γ_{f_X} and γ . That is, $f_Y(y_i) = T$ if γ proceeds from s_{y_i} with the transition labeled b_{y_i} in the computation in which Player 1 follows γ_{f_X} , and $f_Y(y_i) = F$ if γ proceeds from s_{y_i} with the transition labeled $b_{\bar{y}_i}$. When the computation that is consistent with γ_{f_X} and γ reaches the check part, the energy level of both players is $v_{f_X \cup f_Y}$. Since $f_X \cup f_Y$ satisfies ψ , we have that there is $1 \leq i \leq n$ such that the i -th bit of $v_{f_X \cup f_Y}$ is 3. Let v^p be the bit-vector the players own when reaching p . It is easy to verify that v^p is not all-zero. Let j be the most significant bit in v^p that is not 0. We distinguish between two cases. If the j -th bit of v^p is positive, then v^p is positive. In this case, $v^p + t_{1,-1}$ is not negative, and Player 1 can loop in the sink forever and win the game. Otherwise, the j -th bit of v^p is negative, so v^p is negative. So, at some point at the check part, the current bit-vector of the players becomes negative, as a consequence of step made by Player 2. So Player 2 loses.

For the second direction, assume that θ is false, and consider a strategy γ for Player 1. Note that every strategy for Player 1 in G is memoryless. Let f_X be the assignment for X induced by γ . Then, there is an assignment f_Y for Y such that ψ is false under $f_X \cup f_Y$. Let γ_{f_Y} be the following memoryless strategy for Player 2. First, at the assignment part, the strategy γ_{f_Y} is consistent with f_Y . That is, as detailed above, for a position s_{y_i} the strategy γ_{f_Y} proceeds with the transition labeled with the update that corresponds to $f_Y(y_i)$. Let $v = \langle b_n, b_{n-1}, \dots, b_1 \rangle$ be the energy level of both players at the end of the assignment part. Since ψ is false under $f_X \cup f_Y$, then $b_i \in \{0, 1, 2\}$ for all $n \geq i \geq 1$. Accordingly, in the check part, the strategy γ_{f_Y} can choose from s_i a transition labeled $t_{i,-b_i}$, namely a transition that decreases the i -th bit of the energy levels of both players to 0. Consequently, the computation of G that is consistent with γ and γ_{f_Y} reaches the state p with energy level 0, and reaches the sink with a negative energy level for Player 1, which loses.

A.6 Proof of Lemma 12

Let $G = \langle \emptyset, S_2, s_{init}, E, d_1, d_2, x_0^1, x_0^2, \tau \rangle$ be a BBEG. We construct a VASS V with configurations that represent a position and energy vectors in G , with target configuration that represents a position and energy vectors from which Player 2 can win in one move. The idea is that Player 2 wins in G iff she can force the game to an edge in which the energy level of Player 1 is low enough at some component to drop below 0, and her own energy level is high enough to stay non-negative after taking this edge.

Formally, for all $k \in [d_1]$, we construct the $(d_2 + 1)$ -VASS $V_k = \langle Q_k, T_k \rangle$ as follows. Let $Q_k = S \cup \{s_{sink}\}$ for some $s_{sink} \notin S$, and $T'_k = \{\langle u, z, v \rangle : \langle u, v \rangle \in E, \text{ for all } i \in [d_2] \text{ we have that } z[i] = \tau(\langle u, v \rangle)[2][i], \text{ and } z[d_2 + 1] = \tau(\langle u, v \rangle)[1][k]\}$. That is, the vectors on the transitions in T'_k represent the update to the energy vector of Player 2 in their first d_2 components, and the update of the k -th component of Player 1 in their last component. We define the set of transitions $T''_k = \{\langle u, z, s_{sink} \rangle : \text{there is } v \in S \text{ such that } \langle u, z, v \rangle \in T'_k\}$. That is, for every transition in T'_k leaving a state u , there is a transition in T''_k leaving u with the same update and entering s_{sink} . For $i \in [d_2 + 1]$ and $z \in Z$, let b_i^z to be the vector of dimension $d_2 + 1$ with z in the i -th component, and 0 in all other components. We define the set of transitions $T'''_k = \{\langle s_{sink}, b_i^{-1}, s_{sink} \rangle : i \in [d_2]\} \cup \{\langle u, b_{d_2+1}^1, u \rangle : u \in V \setminus \{s_{sink}\}\}$. That is, s_{sink} has self loops that can decrease the components that belong to Player 2. Also, every state but the sink has a self loop that increases the component that belongs to Player 1. We define the set of transitions of V to be $T_k = T'_k \cup T''_k \cup T'''_k \cup \{\langle s_{sink}, 0^{d_2+1}, s_{sink} \rangle\}$. Let $v_{init}^k \in \mathbb{Z}^{d_2+1}$ be the vector with $v_{init}^k[i] = x_0^2[i]$ for all $i \in [d_2]$, and $v_{init}^k[d_2 + 1] = x_0^1[k] + 1$. That is, v_{init}^k represents x_0^2 in its first d_2 components, and $x_0^1[k] + 1$ in its last component. Note that we added 1 to $x_0^1[k]$. That is because in V_k we want to let the last component reach 0, if in the corresponding computation in G it becomes negative.

In the full version, we prove that Player 2 wins G iff there is $k \in [d_1]$ such that $\langle s_{init}, v_{init}^k \rangle \rightarrow^* \langle s_{sink}, 0^{d_2+1} \rangle$ in V_k .

A.7 Deciding whether Player 2 (∞, m_2) -wins

► **Theorem 18.** *Given a BBEG G and $m_2 \in \mathbb{N}$, determining whether Player 2 (∞, m_2) -wins G is decidable.*

Proof. Assume that G is a (d_1, d_2) -BBEG. As in (m_1, ∞) -winning for Player 1, we can consider the product of G with a transducer T for Player 2 with m_2 states. This product is a BBEG all whose positions are owned by Player 1. It is easy to see that Player 1 wins in this product iff it contains infinite computation in which her energy level is always non-negative,

or a finite prefix of a computation that leads to a position in which the energy level of Player 2 is negative in some component while the energy vector of Player 1 along this prefix is always non-negative. Checking the second condition can be done by a reduction to VASS, with a construction similar to the one in the proof of Lemma 12. Checking the first condition can also be reduced to VASS, but is more complicated. So, for the sake of decidability, it is sufficient to note that the first condition can also be solved by solving a d_1 -dimensional energy game, in which we ignore the components that belong to Player 2. From [20, 7], the given initial-credit problem of d_1 -dimensional energy game can be solved in $(d_1 - 1)$ -EXPTIME, and is thus decidable.

It follows that for every transducer with m_2 states for Player 2, we can check whether Player 1 wins when Player 2 follows this transducer. Moreover, if Player 1 does not win, Player 2 does, and so the transducer T induces a winning strategy for her. Thus, Player 2 (∞, m_2) -wins G iff she wins with some transducer with m_2 states, that is, iff she wins in at least one of these products, which is decidable. ◀

A.8 Proof of Theorem 16

Let $G = \langle S_1, S_2, s_{init}, E, d_1, d_2, x_0^1, x_0^2, \tau, j, b \rangle$ be a one-player-bounded BBEG. Assume first that $j = 2$, thus $b \in \mathbb{Z}^{d_2}$ is a bound vector for Player 2. We start with the losing semantics and define the d_1 -dimensional energy game $G' = \langle S'_1, S'_2, \langle s_{init}, x_0^2 \rangle, E', \tau' \rangle$ as follows. Let V be the set of all non-negative vectors in \mathbb{Z}^{d_2} that are bounded by b . That is, $V = \{v \in \mathbb{Z}^{d_2} : 0 \leq v[i] \leq b[i] \text{ for all } i \in [d_2]\}$. Let $S'_1 = S_1 \times V$ and $S'_2 = S_2 \times V$. Also, let $S = S'_1 \cup S'_2 \cup \{s_{sink}\}$, for some $s_{sink} \notin S_1 \cup S_2$. We now define a set of edges $E' \subseteq S' \times S'$ and a cost function $\tau' : E' \rightarrow \mathbb{Z}^{d_1}$. For all $e = \langle s, s' \rangle \in E$ and $v, v' \in V$ such that $v' = v + \tau(e)[2]$, we have the edge $e' = \langle \langle s, v \rangle, \langle s', v' \rangle \rangle$ in E' , with $\tau'(e') = \tau(e)[1]$. For all $e = \langle s, s' \rangle \in E$ and $v \in V$ such that $v + \tau(e)[2] \notin V$, we have the edge $e' = \langle \langle s, v \rangle, s_{sink} \rangle$ in E' , with $\tau'(e') = \tau(e)[1]$. We also have an edge $\langle s_{sink}, s_{sink} \rangle$ in E' , with $\tau'(\langle s_{sink}, s_{sink} \rangle) = 0^{d_1}$. Note that the cost function τ' defines the cost for Player 1 only, while S' maintains the energy level of Player 2.

We claim Player 1 wins in G iff Player 1 wins in G' with initial energy x_0^1 . Indeed, every computation c in G induces a computation c' in G' , such that the current energy level of Player 2 in c' is maintained at the second component of the current position in c' , and the energy level of Player 1 in c is the same as in c' . Thus, if c is infinite, so is c' . Also, if at some point during c , Player 2 exceeds her boundaries (by going below 0 or above b at some component), then c' reaches s_{sink} , which is a winning position for Player 1. Finally, if at some point during c , the energy level of Player 1 drops below 0, then so it does in c' . Hence, in order to decide the winner in G , we can determine the winner in G' . Since the given initial-credit problem for d_1 -dimensional energy game is decidable in $(d_1 - 1)$ -EXPTIME [20, 7], we can decide the winner of a one-player-bounded BBEG with $j = 2$.

Now, in the truncated semantics, since there are finitely-many possible energy vectors for Player 2, we can also expand the position space to maintain them. The only difference is that when an overflow in the energy of Player 2 occurs in some components, the computation stays in positions that correspond to the maximum bound of those components.

We continue to the case $j = 1$, thus $b \in \mathbb{Z}^{d_1}$ is a bound vector for Player 1. We describe the construction for the losing semantics. The extension to the truncated semantics is as in the $j = 2$ case.

We define the d_2 -dimensional energy-reachability game $G' = \langle S'_1, S'_2, \langle s_{init}, x_0 \rangle, E', \tau' \rangle$ as follows. Let V be the set of all non-negative vectors in \mathbb{Z}^{d_1} that are bounded by b . That is, $V = \{v \in \mathbb{Z}^{d_1} : 0 \leq v[i] \leq b[i] \text{ for all } i \in [d_1]\}$. Let $S'_1 = S_1 \times V$ and $S'_2 = S_2 \times V$.

Also, let $S = S'_1 \cup S'_2 \cup \{s_{sink}\}$, for some $s_{sink} \notin S_1 \cup S_2$. We now define a set of edges $E' \subseteq S' \times S'$ and a cost function $\tau' : E' \rightarrow \mathbb{Z}^{d_2}$. For all $e = \langle s, s' \rangle \in E$ and $v, v' \in V$ such that $v' = v + \tau(e)[1]$, we have the edge $e' = \langle \langle s, v \rangle, \langle s', v' \rangle \rangle$ in E' with $\tau'(e') = \tau(e)[2]$. For all $e = \langle s, s' \rangle \in E$ and $v \in V$ such that $v + \tau(e)[1] \notin V$, we have the edge $e' = \langle \langle s, v \rangle, s_{sink} \rangle$ in E' with $\tau'(e') = \tau(e)[2]$. We also have an edge $\langle s_{sink}, s_{sink} \rangle$ in E' with $\tau'(\langle s_{sink}, s_{sink} \rangle) = 0^{d_2}$. Note that the cost function τ' defines the cost for Player 2 only, while S' maintains the energy level of Player 1. In G' , Player 2 wins if she can reach s_{sink} , while keeping her own energy vector non-negative. Otherwise, Player 1 wins.

By [16], one-dimensional energy-reachability games can be decided in $\text{NP} \cap \text{coNP}$. Since we are interested in the multi-dimensional case, we give here a brief description of an algorithm that determines the winner in multi-dimensional energy-reachability games: First, note that without the energy constraints, thus in a plain reachability game played on the game graph G' with objective s_{sink} , one can compute in polynomial time the set $Attr$ of winning positions for the reacher, namely for Player 2. From every position in $Attr$, Player 2 has a memoryless winning strategy, called the *attractor strategy*. Since the strategy is winning a memoryless, it includes no cycles, and so we can assume that every play that is consistent with this strategy is a simple path in the graph. Now, adding the energy constraint to the picture, we get that if Player 2 reaches a position in $Attr$ with energy level that is sufficient for traversing a simple path in G' she can win by using her attractor strategy. Moreover, such a sufficient energy level can be computed, for example $|E| \cdot |W|^{d_2}$, where $|W|$ is the largest absolute value of an update, is sufficient. Hence, we can extend the position-space of G' to maintain the energy level of Player 2 (with the bound of $|E| \cdot |W|^{d_2}$), and then determine the winner of a plain reachability game on this extended graph.

Half-Positional Objectives Recognized by Deterministic Büchi Automata

Patricia Bouyer 


Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, 91190, Gif-sur-Yvette, France

Antonio Casares 

LaBRI, Université de Bordeaux, France

Mickael Randour 

F.R.S.-FNRS & UMONS – Université de Mons, Belgium

Pierre Vandenhove 

F.R.S.-FNRS & UMONS – Université de Mons, Belgium

Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, 91190, Gif-sur-Yvette, France

Abstract

A central question in the theory of two-player games over graphs is to understand which objectives are *half-positional*, that is, which are the objectives for which the protagonist does not need memory to implement winning strategies. Objectives for which *both* players do not need memory have already been characterized (both in finite and infinite graphs); however, less is known about half-positional objectives. In particular, no characterization of half-positionality is known for the central class of ω -regular objectives.

In this paper, we characterize objectives recognizable by deterministic Büchi automata (a class of ω -regular objectives) that are half-positional, in both finite and infinite graphs. Our characterization consists of three natural conditions linked to the language-theoretic notion of *right congruence*. Furthermore, this characterization yields a polynomial-time algorithm to decide half-positionality of an objective recognized by a given deterministic Büchi automaton.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases two-player games on graphs, half-positionality, memoryless optimal strategies, Büchi automata, ω -regularity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.20

Related Version *Full Version*: <https://arxiv.org/abs/2205.01365> [8]

Funding This work has been partially supported by the ANR Project MAVeriQ (ANR-ANR-20-CE25-0012). Mickael Randour is an F.R.S.-FNRS Research Associate and a member of the TRAIL Institute. Pierre Vandenhove is an F.R.S.-FNRS Research Fellow.

Acknowledgements We would like to thank Igor Walukiewicz for suggesting a simplification of the proof of Lemma 17 and Pierre Ohlmann for interesting discussions on the subject.

1 Introduction

Graph games and reactive synthesis. We study *zero-sum turn-based games on graphs* confronting two players (a protagonist and its opponent). They interact by moving a pebble in turns through the edges of a graph for an infinite amount of time. Each vertex belongs to a player, and the player controlling the current vertex decides on the next state of the game. Edges of the graph are labeled with *colors*, and the interaction of the two players therefore produces an infinite sequence of them. The objective of the game is specified by a subset of infinite sequences of colors, and the protagonist wins if the produced sequence



© Patricia Bouyer, Antonio Casares, Mickael Randour, and Pierre Vandenhove; licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 20; pp. 20:1–20:18



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

belongs to this set. We are interested in finding a *winning strategy* for the protagonist, that is, a function indicating how the protagonist should move in any situation, guaranteeing the achievement of the objective.

This game-theoretic model is particularly fitted to study the *reactive synthesis problem* [7]: a system (the protagonist) wants to satisfy a specification (the objective) while interacting continuously with its environment (the opponent). The goal is to build a controller for the system satisfying the specification, whenever possible. This comes down to finding a winning strategy for the protagonist in the derived game.

Half-positionality. In order to obtain a controller for the system that is simple to implement, we are interested in finding the simplest possible winning strategy. Here, we focus on the amount of information that winning strategies have to remember. The simplest strategies are then arguably *positional* (also called *memoryless*) strategies, which do not remember anything about the past and base their decisions solely on the current state of the game. We intend to understand for which objectives positional strategies suffice for the protagonist to play optimally (i.e., to win whenever it is possible) – we call these objectives *half-positional*. We distinguish half-positionality from *bipositionality* (or *memoryless-determinacy*), which refers to objectives for which positional strategies suffice to play optimally for *both* players.

Many natural objectives have been shown to be bipositional over games on finite and sometimes infinite graphs: e.g., discounted sum [53], mean-payoff [28], parity [29], total payoff [31], energy [9], or average-energy games [11]. Bipositionality can be established using general criteria and characterizations, over games on both finite [31, 32, 3] and infinite [26] graphs. Yet, there exist many objectives and combinations thereof for which one player, but not both, has positional optimal strategies (Rabin conditions [35, 34], mean-payoff parity [22], energy parity [20], some window objectives [21, 14], energy mean-payoff [15]...), and to which these results do not apply.

Various attempts have been made to understand common underlying properties of half-positional objectives and provide sufficient conditions [36, 37, 38, 6], but little more was known until the recent work of Ohlmann [48] (discussed below). These conditions are not general enough to prove half-positionality of some very simple objectives, even in the well-studied class of ω -regular objectives [6, Lemma 13]. Furthermore, multiple questions concerning half-positionality remain open. For instance, in [38], Kopczyński conjectured that *prefix-independent* half-positional objectives are closed under finite union (this conjecture was recently refuted for games on finite graphs [39], but is still unsolved for games on infinite graphs). Also, Kopczyński showed that given a deterministic parity automaton recognizing a prefix-independent objective W , we can decide if W is half-positional [37]. However, the time complexity of his algorithm is $\mathcal{O}(n^{\mathcal{O}(n^2)})$, where n is the number of states of the automaton. It is unknown whether this can be done in polynomial time, and no algorithm exists in the non-prefix-independent case.

ω -regular objectives and deterministic Büchi automata. A central class of objectives, whose half-positionality is not yet completely understood, is the class of ω -regular objectives. There are multiple equivalent definitions for them: they are the objectives defined, e.g., by ω -regular expressions, by non-deterministic Büchi automata [45], and by deterministic parity automata [46]. These objectives coincide with the class of objectives defined by monadic second-order formulas [17], and they encompass linear-time temporal logic (LTL) specifications [50]. Part of their interest is due to the landmark result that finite-state machines are sufficient to implement optimal strategies in ω -regular games [16, 33], implying the decidability of the monadic second-order theory of natural numbers with the successor relation [17] and the decidability of the synthesis problem under LTL specifications [51].

In this paper, we focus on the subclass of ω -regular objectives recognized by *deterministic Büchi automata* (DBA), that we call *DBA-recognizable*. DBA-recognizable objectives correspond to the ω -regular objectives that can be written as a countable intersection of open objectives (for the Cantor topology, that is, that are G_δ -sets of the Borel hierarchy); or equivalently, that are the limit of a regular language of finite words [42, 49]. Deciding the winner of a game with a DBA-recognizable objective is doable in polynomial time in the size of the arena and the DBA (by solving a Büchi game on the product of the arena and the DBA [7]).

We now discuss two technical tools at the core of our approach: *universal graphs* and *right congruences*.

Universal graphs. One recent breakthrough in the study of half-positionality is the introduction of *well-monotonic universal graphs*, combinatorial structures that can be used to provide a witness of winning strategies in games with a half-positional objective. Recently, Ohlmann [48] has shown that the existence of a *well-monotonic universal graph* for an objective W exactly characterizes half-positionality (under minor technical assumptions on W). Moreover, under these assumptions, a wide class of algorithms, called *value iteration algorithms*, can be applied to solve any game with a half-positional objective [24, 48].

Although it brings insight on the structure of half-positional objectives, showing half-positionality through the use of universal graphs is not always straightforward, and has not yet been applied in a systematic way to ω -regular objectives.

Right congruence. Given an objective W , the *right congruence* \sim_W of W is an equivalence relation on finite words: two finite words w_1 and w_2 are equivalent for \sim_W if for all infinite continuations w , $w_1w \in W$ if and only if $w_2w \in W$. There is a natural automaton classifying the equivalence classes of the right congruence, which we refer to as the *prefix-classifier* [54, 44].

In the case of languages of *finite* words, a straightforward adaptation of the right congruence recovers the known Myhill-Nerode congruence. This equivalence relation characterizes the regular languages (a language is regular if and only if its congruence has finitely many equivalence classes), and the prefix-classifier is exactly the smallest deterministic finite automaton recognizing a language – this is the celebrated Myhill-Nerode theorem [47].

Objectives are languages of *infinite* words, for which the situation is not so clear-cut. In particular, an ω -regular objective may not always be recognized by its prefix-classifier along with a natural acceptance condition (Büchi, coBüchi, parity, Muller. . .) [44, 4].

Contributions. Our main contribution is a *characterization* of half-positionality for DBA-recognizable objectives through a conjunction of three easy-to-check conditions (Theorem 10).

- (1) The equivalence classes of the right congruence are *totally* ordered w.r.t. inclusion of their winning continuations.
- (2) Whenever the set of winning continuations of a finite word w_1 is a proper subset of the set of winning continuations of a concatenation w_1w_2 , the word $w_1(w_2)^\omega$ produced by repeating infinitely often w_2 is winning.
- (3) The objective has to be recognizable by a DBA using the structure of its prefix-classifier.

A few examples of simple DBA-recognizable objectives that were not encompassed by previous half-positionality criteria [36, 6] are, e.g., reaching a color twice [6, Lemma 13] and weak parity [55]. We also refer to Example 7, which is half-positional but not bipositional, and whose half-positionality is straightforward using our characterization.

Various corollaries with practical and theoretical interest follow from our characterization.

- We obtain a painless path to show (by checking each of the three conditions) that given a deterministic Büchi automaton, the half-positionality of the objective it recognizes is decidable in time $\mathcal{O}(k^2 \cdot n^4)$, where k is the number of colors and n is the number of states of the DBA (Section 3.3).
- Prefix-independent DBA-recognizable half-positional objectives are exactly the very simple *Büchi conditions*, which consist of all the infinite words seeing infinitely many times some subset of the colors (Proposition 11). In particular, Kopczyński’s conjecture trivializes for DBA-recognizable objectives (the union of Büchi conditions is a Büchi condition).
- We obtain a *finite-to-infinite* and *one-to-two-player* lift result (Proposition 14): in order to check that a DBA-recognizable objective is half-positional over arbitrary – possibly two-player and infinite – graphs, it suffices to check the existence of positional optimal strategies over *finite* graphs where all the vertices are controlled by the protagonist.

Other related works. We have discussed the relevant literature on half-positionality [36, 37, 6, 48] and bipositionality [31, 32, 26, 3]. A more general quest is to understand *memory requirements* when positional strategies are not powerful enough: e.g., [43, 10, 12, 13].

Memory requirements have been precisely characterized for some classes of ω -regular objectives (not encompassing the class of DBA-recognizable objectives), such as Muller conditions [27, 57, 18, 19] and safety specifications, i.e., objectives that are closed for the Cantor topology [25]. The latter also uses the order of the equivalence classes of the right congruence as part of its characterization.

Recently, a link between the prefix-classifier, the memory requirements, and the recognizability of ω -regular objectives was established [13]. However, this result does not provide optimal bounds on the strategy complexity, and is thereby insufficient to study half-positionality.

Structure of the paper. Notations and definitions are introduced in Section 2. Our main contributions are presented in Section 3: we introduce and discuss the three conditions used in our results, then we state our main characterization (Theorem 10) and some corollaries, and we end with an explanation on how to use the characterization to decide half-positionality of DBA-recognizable objectives in polynomial time. Due to space constraints, we only provide high-level details about proofs in this version of the article: a proof sketch for Theorem 10 is provided in Section 4. Complete proofs, as well as additional details and examples, can be found in the extended version of the article [8].

2 Preliminaries

In the whole article, letter C refers to a (finite or infinite) non-empty set of *colors*. Given a set A , we write respectively A^* , A^+ , and A^ω for the set of finite, non-empty finite, and infinite sequences of elements of A . We denote by ε the empty word.

2.1 Games and positionality

Graphs. An (*edge-colored*) graph $\mathcal{G} = (V, E)$ is given by a non-empty set of *vertices* V (of any cardinality) and a set of *edges* $E \subseteq V \times C \times V$. We write $v \xrightarrow{c} v'$ if $(v, c, v') \in E$. We assume graphs to be *non-blocking*: for all $v \in V$, there exists $(v', c, v'') \in E$ such that $v = v'$. We allow graphs with infinite branching. For $v \in V$, an *infinite path of \mathcal{G} from v* is an infinite sequence of edges $\pi = (v_0, c_1, v'_1)(v_1, c_2, v'_2) \dots \in E^\omega$ such that $v_0 = v$ and for all $i \geq 1$, $v'_i = v_i$. A *finite path of \mathcal{G} from v* is a finite prefix in E^* of an infinite path of \mathcal{G} from

v . For convenience, we assume that there is a distinct *empty path* λ_v for every $v \in V$. If $\gamma = (v_0, c_1, v_1) \dots (v_{n-1}, c_n, v_n)$ is a non-empty finite path of \mathcal{G} , we define $\text{last}(\gamma) = v_n$. For an empty path λ_v , we define $\text{last}(\lambda_v) = v$. An infinite (resp. finite) path $(v_0, c_1, v_1)(v_1, c_2, v_2) \dots$ is sometimes represented as $v_0 \xrightarrow{c_1} v_1 \xrightarrow{c_2} \dots$. A graph $\mathcal{G} = (V, E)$ is *finite* if both V and E are finite. A graph is *strongly connected* if for every pair of vertices $(v, v') \in V \times V$ there is a path from v to v' . A *strongly connected component* of \mathcal{G} is a maximal strongly connected subgraph.

Arenas and strategies. We consider two players \mathcal{P}_1 and \mathcal{P}_2 . An *arena* is a tuple $\mathcal{A} = (V, V_1, V_2, E)$ such that (V, E) is a graph and V is the disjoint union of V_1 and V_2 . Intuitively, vertices in V_1 are controlled by \mathcal{P}_1 and vertices in V_2 are controlled by \mathcal{P}_2 . An arena $\mathcal{A} = (V, V_1, V_2, E)$ is a *one-player arena* of \mathcal{P}_1 (resp. of \mathcal{P}_2) if $V_2 = \emptyset$ (resp. $V_1 = \emptyset$). Finite paths of (V, E) are called *histories* of \mathcal{A} . For $i \in \{1, 2\}$, we denote by $\text{Hists}_i(\mathcal{A})$ the set of histories γ of \mathcal{A} such that $\text{last}(\gamma) \in V_i$.

Let $i \in \{1, 2\}$. A *strategy* of \mathcal{P}_i on \mathcal{A} is a function $\sigma_i: \text{Hists}_i(\mathcal{A}) \rightarrow E$ such that for all $\gamma \in \text{Hists}_i(\mathcal{A})$, the first component of $\sigma_i(\gamma)$ coincides with $\text{last}(\gamma)$. Given a strategy σ_i of \mathcal{P}_i , we say that an infinite path $\pi = e_1 e_2 \dots$ is *consistent with* σ_i if for all finite prefixes $\gamma = e_1 \dots e_i$ of π such that $\text{last}(\gamma) \in V_i$, $\sigma_i(\gamma) = e_{i+1}$. A strategy σ_i is *positional* (also called *memoryless* in the literature) if its outputs only depend on the current vertex and not on the whole history, i.e., if there exists a function $f: V_i \rightarrow E$ such that for $\gamma \in \text{Hists}_i(\mathcal{A})$, $\sigma_i(\gamma) = f(\text{last}(\gamma))$.

Objectives. An *objective* is a set $W \subseteq C^\omega$ (subsets of C^ω are sometimes also called *languages of infinite words*, ω -*languages*, or *winning conditions* in the literature). When an objective W is clear in the context, we say that an infinite word $w \in C^\omega$ is *winning* if $w \in W$, and *losing* if $w \notin W$. We write \overline{W} for the complement $C^\omega \setminus W$ of an objective W . An objective W is *prefix-independent* if for all $w \in C^*$ and $w' \in C^\omega$, $w' \in W$ if and only if $ww' \in W$. An objective that we will often consider is the *Büchi condition*: given a subset $F \subseteq C$, we denote by $\text{Büchi}(F)$ the set of infinite words seeing infinitely many times a color in F . Such an objective is prefix-independent. A *game* is a tuple (\mathcal{A}, W) of an arena \mathcal{A} and an objective W .

Optimality and half-positionality. Let $\mathcal{A} = (V, V_1, V_2, E)$ be an arena, (\mathcal{A}, W) be a game, and $v \in V$. We say that a strategy σ_1 of \mathcal{P}_1 is *winning from* v if for all infinite paths $v_0 \xrightarrow{c_1} v_1 \xrightarrow{c_2} \dots$ from v consistent with σ_1 , $c_1 c_2 \dots \in W$.

A strategy of \mathcal{P}_1 is *optimal for* \mathcal{P}_1 in (\mathcal{A}, W) if it is winning from all the vertices from which \mathcal{P}_1 has a winning strategy. We often write *optimal for* \mathcal{P}_1 in \mathcal{A} if the objective W is clear from the context. We stress that this notion of optimality requires a *single* strategy to be winning from *all* the winning vertices (a property sometimes called *uniformity*).

An objective W is *half-positional* if for all arenas \mathcal{A} , there exists a positional strategy of \mathcal{P}_1 on \mathcal{A} that is optimal for \mathcal{P}_1 in \mathcal{A} . We sometimes only consider half-positionality on a restricted set of arenas (typically, finite and/or one-player arenas). For a class of arenas \mathcal{X} , an objective W is *half-positional over* \mathcal{X} if for all arenas $\mathcal{A} \in \mathcal{X}$, there exists a positional strategy of \mathcal{P}_1 on \mathcal{A} that is optimal for \mathcal{P}_1 in \mathcal{A} .

2.2 Büchi automata

Automaton structures and Büchi automata. A *non-deterministic automaton structure* (on C) is a tuple $\mathcal{S} = (Q, C, Q_{\text{init}}, \Delta)$ such that Q is a finite set of *states*, $Q_{\text{init}} \subseteq Q$ is a non-empty set of *initial states* and $\Delta \subseteq Q \times C \times Q$ is a set of *transitions*. We assume that all states of automaton structures are reachable from an initial state in Q_{init} by taking transitions in Δ .

A (*transition-based*) *non-deterministic Büchi automaton* (NBA) is an automaton structure \mathcal{S} together with a set of transitions $\alpha \subseteq \Delta$. The transitions in α are called *Büchi transitions*.

Given an NBA $\mathcal{B} = (Q, C, Q_{\text{init}}, \Delta, \alpha)$, a (*finite or infinite*) *run of \mathcal{B} on a (*finite or infinite*) word $w = c_1 c_2 \dots \in C^* \cup C^\omega$* is a sequence $(q_0, c_1, q_1)(q_1, c_2, q_2) \dots \in \Delta^* \cup \Delta^\omega$ such that $q_0 \in Q_{\text{init}}$. An infinite run $(q_0, c_1, q_1)(q_1, c_2, q_2) \dots \in \Delta^\omega$ of \mathcal{B} is *accepting* if for infinitely many $i \geq 0$, $(q_i, c_{i+1}, q_{i+1}) \in \alpha$. A word $w \in C^\omega$ is *accepted* by \mathcal{B} if there exists an accepting run of \mathcal{B} on w – if not, it is *rejected*. We denote the set of infinite words accepted by \mathcal{B} by $\mathcal{L}(\mathcal{B})$, and we then say that $\mathcal{L}(\mathcal{B})$ is the objective *recognized by \mathcal{B}* . Here, we take the definition of an ω -regular objective as an objective that can be recognized by an NBA. Given an automaton structure $\mathcal{S} = (Q, C, Q_{\text{init}}, \Delta)$, we say that an NBA \mathcal{B} is *built on top of \mathcal{S}* if there exists $\alpha \subseteq \Delta$ such that $\mathcal{B} = (Q, C, Q_{\text{init}}, \Delta, \alpha)$.

Deterministic automata. An automaton structure $\mathcal{S} = (Q, C, Q_{\text{init}}, \Delta)$ is *deterministic* if $|Q_{\text{init}}| = 1$ and, for each $q \in Q$ and $c \in C$, there is exactly one $q' \in Q$ such that $(q, c, q') \in \Delta$ (we remark that in this paper deterministic automata are *complete*). A *deterministic Büchi automaton* (DBA) is an NBA whose underlying automaton structure is deterministic. For a DBA $\mathcal{B} = (Q, C, \{q_{\text{init}}\}, \Delta, \alpha)$, we denote by q_{init} the unique initial state (and we will drop the braces around q_{init} in the tuple), and by $\delta: Q \times C \rightarrow Q$ the *update function* that associates to $(q, c) \in Q \times C$ the only $q' \in Q$ such that $(q, c, q') \in \Delta$. We denote by δ^* the natural extension of δ to finite words. As transitions are uniquely determined by their first two components, we also assume for brevity that $\alpha \subseteq Q \times C$.

For a DBA \mathcal{B} , a state $q \in Q$ and a word $w = c_1 c_2 \dots \in C^* \cup C^\omega$, we denote by $\mathcal{B}(q, w) = (q, c_1, q_1)(q_1, c_2, q_2) \dots \in \Delta^* \cup \Delta^\omega$ the only run on w starting from q .

An objective W is *DBA-recognizable* if there exists a DBA \mathcal{B} such that $W = \mathcal{L}(\mathcal{B})$. For $F \subseteq C$, notice that $\text{Büchi}(F)$ is DBA-recognizable: it is recognized by the DBA $(\{q_{\text{init}}\}, C, q_{\text{init}}, \Delta, \alpha)$ with a *single* state such that $(q_{\text{init}}, c) \in \alpha$ if and only if $c \in F$.

► **Remark 1.** The fact that a single state suffices for recognizing $\text{Büchi}(F)$ relies on the assumption that our DBA are *transition-based* and not *state-based* (α is a set of transitions, not of states). Indeed, apart from the trivial cases $F = \emptyset$ and $F = C$, a state-based DBA recognizing $\text{Büchi}(F)$ requires two states. The third condition of our upcoming characterization (Theorem 10) would therefore not apply to this simple example if we only considered state-based DBA. ┘

► **Remark 2.** DBA recognize a proper subset of the ω -regular objectives [56]. ┘

Let $\mathcal{B} = (Q, C, q_{\text{init}}, \Delta, \alpha)$ be a DBA. We say that a finite run $\rho \in \Delta^*$ of \mathcal{B} is α -free if it does not contain any transition from α . For $q \in Q$, we define

$$\begin{aligned} \alpha\text{-Free}_{\mathcal{B}}(q) &= \{w \in C^* \mid \mathcal{B}(q, w) \text{ is } \alpha\text{-free}\}, \\ \alpha\text{-FreeCycles}_{\mathcal{B}}(q) &= \{w \in C^* \mid w \in \alpha\text{-Free}_{\mathcal{B}}(q) \text{ and } \delta^*(q, w) = q\}. \end{aligned}$$

We call the words in the first set the α -free words from q , and the words in the second set the α -free cycles from q .

Right congruence. Let $W \subseteq C^\omega$ be an objective. For a finite word $w \in C^*$, we write $w^{-1}W = \{w' \in C^\omega \mid ww' \in W\}$ for the set of *winning continuations of w* . We define the *right congruence* $\sim_W \subseteq C^* \times C^*$ of W as $w_1 \sim_W w_2$ if $w_1^{-1}W = w_2^{-1}W$. Relation \sim_W is an equivalence relation. When W is clear from the context, we write \sim for \sim_W . For $w \in C^*$, we denote by $[w] \subseteq C^*$ its equivalence class for \sim .

When \sim has finitely many equivalence classes, we can associate a natural deterministic automaton structure $\mathcal{S}_\sim = (Q_\sim, C, \tilde{q}_{\text{init}}, \Delta_\sim)$ to \sim such that Q_\sim is the set of equivalence classes of \sim , $\tilde{q}_{\text{init}} = [\varepsilon]$, and $\delta_\sim([w], c) = [wc]$ [54, 44]. The transition function δ_\sim is well-defined since if $w_1 \sim w_2$, then for all $c \in C$, $w_1c \sim w_2c$. We call the automaton structure \mathcal{S}_\sim the *prefix-classifier of W* .

► **Remark 3.** Equivalence relation \sim_W has only one equivalence class if and only if W is prefix-independent. In particular, an objective has a prefix-classifier with a single state if and only if it is prefix-independent. ◻

We define the *prefix preorder* \preceq_W of W : for $w_1, w_2 \in C^*$, we write $w_1 \preceq_W w_2$ if $w_1^{-1}W \subseteq w_2^{-1}W$ (meaning that any continuation that is winning after w_1 is also winning after w_2). Intuitively, $w_1 \preceq_W w_2$ means that a game starting with w_2 is always preferable to a game starting with w_1 for \mathcal{P}_1 , as there are more ways to win after w_2 . When W is clear from the context, we write \preceq for \preceq_W . Relation $\preceq \subseteq C^* \times C^*$ is a preorder. Notice that \sim is equal to $\preceq \cap \succeq$. We also define the strict preorder $\prec = \preceq \setminus \sim$.

Given a DBA $\mathcal{B} = (Q, C, q_{\text{init}}, \Delta, \alpha)$ recognizing the objective W , observe that for $w, w' \in C^*$ such that $\delta^*(q_{\text{init}}, w) = \delta^*(q_{\text{init}}, w')$, we have $w \sim w'$. In this case, equivalence relation \sim has at most $|Q|$ equivalence classes. For $q \in Q$, we write abusively $q^{-1}W$ for the objective recognized by the DBA $(Q, C, q, \Delta, \alpha)$. Objective $q^{-1}W$ equals $w^{-1}W$ for any word $w \in C^*$ such that $\delta^*(q_{\text{init}}, w) = q$. We extend the equivalence relation \sim and preorder \preceq to elements of Q (we sometimes write $\sim_{\mathcal{B}}$ and $\preceq_{\mathcal{B}}$ to avoid any ambiguity).

3 Half-positionality characterization for DBA-recognizable objectives

In this section, we present our main contribution in Theorem 10, by giving three conditions that exactly characterize half-positional DBA-recognizable objectives. These conditions are presented in Section 3.1. Theorem 10 and several consequences of it are stated in Section 3.2. In Section 3.3, we use this characterization to show that we can decide the half-positionality of a DBA in polynomial time. Missing proofs for this section are in [8, Section 3], except for the proof of Theorem 10, which is in [8, Sections 4 & 5].

3.1 Three conditions for half-positionality

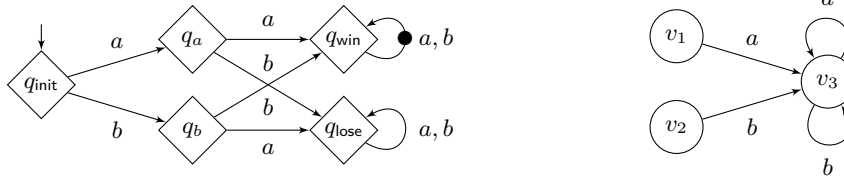
We define the three conditions on objectives at the core of our characterization.

► **Condition 1 (Total prefix preorder).** We say that an objective $W \subseteq C^\omega$ has a *total prefix preorder* if for all $w_1, w_2 \in C^*$, $w_1 \preceq_W w_2$ or $w_2 \preceq_W w_1$.

An objective W recognized by a DBA \mathcal{B} has a total prefix preorder if and only if the (reachable) states of \mathcal{B} are totally ordered for $\preceq_{\mathcal{B}}$.

► **Example 4 (Not total prefix preorder).** Let $C = \{a, b\}$. We consider the objective W recognized by the DBA \mathcal{B} depicted in Figure 1 (left). It consists of the infinite words starting with aa or bb . This objective does not have a total prefix preorder: words a and b are incomparable for \preceq_W . Indeed, a^ω is winning after a but not after b , and b^ω is winning after b but not after a . In terms of automaton states, we have that q_a and q_b are incomparable for $\preceq_{\mathcal{B}}$. This objective is not half-positional, as witnessed by the arena on the right of Figure 1. In this arena, \mathcal{P}_1 is able to win when the game starts in v_1 by playing a in v_3 , and when the game starts in v_2 by playing b . However, no positional strategy wins from both v_1 and v_2 . ◻

► **Remark 5.** The prefix preorder of an objective W is total if and only if the prefix preorder of its complement \overline{W} is total. ◻



■ **Figure 1** DBA \mathcal{B} recognizing objective $W = (aa + bb)C^\omega$ (left), and an arena in which positional strategies do not suffice for \mathcal{P}_1 to play optimally for this objective (right). Transitions labeled with a \bullet symbol are the Büchi transitions. In figures, diamonds represent automaton states and circles represent arena vertices controlled by \mathcal{P}_1 .

► **Remark 6.** Having a total prefix preorder is equivalent to the *strong monotony* notion [6] in general, and equivalent to *monotony* [32] for ω -regular objectives. We discuss in more depth the relation between the conditions appearing in the characterization and other properties from the literature studying half-positionality in [8, Appendix A]. ┘

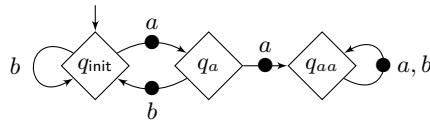
► **Condition 2 (Progress-consistency).** We say that an objective W is *progress-consistent* if for all $w_1 \in C^*$ and $w_2 \in C^+$ such that $w_1 \prec w_1w_2$, we have $w_1(w_2)^\omega \in W$.

Intuitively, this means that whenever a word w_2 can be used to make progress after seeing a word w_1 (in the sense of getting to a position in which more continuations are winning), then repeating this word has to be winning.

► **Example 7 (Progress-consistent objective).** We consider the DBA in Figure 2. This DBA recognizes the objective $W = \text{Büchi}(\{a\}) \cup C^*aaC^\omega$: W contains the words seeing a infinitely often, or that see a twice in a row at some point. The equivalence classes for \sim_W are $q_{\text{init}}^{-1}W = W$, $q_a^{-1}W = aC^\omega \cup W$ and $q_{aa}^{-1}W = C^\omega$. This objective is progress-consistent: any word reaching q_{aa} is straightforwardly accepted when repeated infinitely often, and any word w such that $\delta^*(q_{\text{init}}, w) = q_a$ necessarily contains at least one a , and thus is accepted when repeated infinitely often. Objective W is half-positional, which will be readily shown with our upcoming characterization (Theorem 10).

Here, notice that the complement \bar{W} of W is not progress-consistent. Indeed, $a \prec_{\bar{W}} a(bab)$, but $a(bab)^\omega \notin \bar{W}$. Unlike having a total prefix preorder, progress-consistency can hold for an objective but not its complement.

Note that half-positionality of W cannot be shown using existing half-positionality criteria [36, 6] (it is neither prefix-independent nor *concave*) nor bipositionality criteria, as it is simply not bipositional. ┘



■ **Figure 2** A DBA recognizing the set of words seeing a infinitely many times, or aa at some point.

► **Condition 3 (Recognizability by the prefix-classifier).** Being recognized by a Büchi automaton built on top of the prefix-classifier is our third condition. In other words, for a DBA-recognizable objective $W \subseteq C^\omega$ and its prefix-classifier $\mathcal{S} = (Q_\sim, C, \tilde{q}_{\text{init}}, \Delta_\sim)$, this condition requires that there exists $\alpha_\sim \subseteq Q_\sim \times C$ such that W is recognized by DBA $(Q_\sim, C, \tilde{q}_{\text{init}}, \Delta_\sim, \alpha_\sim)$.

We show an example of a DBA-recognizable objective that satisfies the first two conditions (having a total prefix preorder and progress-consistency), but not this third condition, and which is not half-positional.

► **Example 8** (Not recognizable by the prefix-classifier). Let $C = \{a, b\}$. We consider the objective $W = \text{Büchi}(\{a\}) \cap \text{Büchi}(\{b\})$ recognized by the DBA in Figure 3. This objective is prefix-independent: as such (Remark 3), there is only one equivalence class for \sim . This implies that the prefix preorder is total, and that W is progress-consistent (the premise of the progress-consistency property can never be true). This objective is not half-positional, as witnessed by the arena in Figure 3 (right): \mathcal{P}_1 has a winning strategy from v , but it needs to take infinitely often both a and b .

Any DBA recognizing this objective has at least two states, but all their (reachable) states are equivalent for \sim – no matter the state we choose as an initial state, the recognized objective is the same (by prefix-independence). As it is prefix-independent, its prefix-classifier \mathcal{S}_{\sim} has only one state. \lrcorner



■ **Figure 3** DBA recognizing the objective $\text{Büchi}(\{a\}) \cap \text{Büchi}(\{b\})$ (left), and an arena in which positional strategies do not suffice for \mathcal{P}_1 to play optimally for this objective (right).

As will be shown formally, being recognized by a DBA built on top of the prefix-classifier is necessary for half-positionality of *DBA-recognizable* objectives over finite one-player arenas. The first two conditions are actually necessary for half-positionality of general objectives, but this third condition is not, even for objectives recognized by other standard classes of ω -automata.

► **Example 9.** We consider the complement \overline{W} of the objective $W = \text{Büchi}(\{a\}) \cap \text{Büchi}(\{b\})$ of Example 8, which consists of the words ending with a^ω or b^ω . Objective \overline{W} is not DBA-recognizable (a close proof can be found in [5, Theorem 4.50]). Still, it is recognizable by a *deterministic coBüchi automaton* similar to the automaton in Figure 3, but which accepts infinite words that visit transitions labeled by \bullet only finitely often. This objective is half-positional, which can be shown using [27, Theorem 6]. However, its prefix-classifier has just one state, and there is no way to recognize \overline{W} by building a coBüchi (or even parity) automaton on top of it. \lrcorner

3.2 Characterization and corollaries

We have now defined the three conditions required for our characterization.

► **Theorem 10.** *Let $W \subseteq C^\omega$ be a DBA-recognizable objective. Objective W is half-positional (over all arenas) if and only if*

- *its prefix preorder \preceq is total,*
- *it is progress-consistent, and*
- *it can be recognized by a Büchi automaton built on top of its prefix-classifier \mathcal{S}_{\sim} .*

High-level details about the proof of this theorem are provided in Section 4. The complete proof of the necessity of the three conditions can be found in [8, Section 4]; the proof of the sufficiency of the conjunction of the three conditions can be found in [8, Section 5].

This characterization is valuable to prove (and disprove) half-positionality of DBA-recognizable objectives. Examples 4 and 8 are not half-positional, and they falsify respectively the first and the third condition from the statement. On the other hand, Example 7 is half-positional. We have already discussed its progress-consistency, but it is also straightforward to verify that its prefix preorder is total and that it is recognizable by its prefix-classifier: the right congruence has three totally ordered equivalence classes corresponding to the states of the automaton of Figure 2.

We state two notable consequences of Theorem 10 and of its proof technique. The first one is the specialization of Theorem 10 to objectives that are prefix-independent, a frequent assumption in the literature [36, 26, 30, 24] – under this assumption, half-positionality of DBA-recognizable objectives is very easy to understand and characterize.

► **Proposition 11.** *Let $W \subseteq C^\omega$ be a prefix-independent, DBA-recognizable objective. Objective W is half-positional if and only if there exists $F \subseteq C$ such that $W = \text{Büchi}(F)$.*

► **Remark 12.** A corollary of this result is that when W is prefix-independent, DBA-recognizable and half-positional, we also have that \overline{W} is half-positional. Indeed, the complement of objective $W = \text{Büchi}(F)$ is a so-called *coBüchi objective*, which is also known to be half-positional (it is a special case of a parity objective [29]). This statement does not hold in general when W is not prefix-independent, as was shown in Example 7. Moreover, the reciprocal of the statement also does not hold, as was shown in Example 9. ◻

► **Remark 13.** A second corollary is that prefix-independent DBA-recognizable half-positional objectives are closed under finite union (since a finite union of Büchi conditions is a Büchi condition). This settles Kopczyński’s conjecture for DBA-recognizable objectives. ◻

A second consequence of Theorem 10 and its proof technique shows that half-positionality of DBA-recognizable objectives can be reduced to half-positionality over the restricted class of *finite, one-player arenas*. Results reducing strategy complexity in two-player arenas to the easier question of strategy complexity in one-player arenas are sometimes called *one-to-two-player lifts* and appear in multiple places in the literature [32, 10, 40, 13].

► **Proposition 14 (One-to-two-player and finite-to-infinite lift).** *Let $W \subseteq C^\omega$ be a DBA-recognizable objective. If objective W is half-positional over finite one-player arenas, then it is half-positional over all arenas (of any cardinality).*

One-to-two-player lifts from the literature all require an assumption on the strategy complexity of *both* players, and are either stated solely over finite arenas, or solely over infinite arenas. Proposition 14, albeit set in the more restricted context of DBA-recognizable objectives, therefore displays stronger properties than the known one-to-two-player lifts.

3.3 Deciding half-positionality in polynomial time

In this section, we assume that C is finite. We show that the problem of deciding, given a DBA $\mathcal{B} = (Q, C, q_{\text{init}}, \Delta, \alpha)$ as an input, whether $\mathcal{L}(\mathcal{B})$ is half-positional can be solved in polynomial time, and more precisely in time $\mathcal{O}(|C|^2 \cdot |Q|^4)$.

We investigate how to verify each property used in the characterization of Theorem 10. Let $\mathcal{B} = (Q, C, q_{\text{init}}, \Delta, \alpha)$ be a DBA (we assume w.l.o.g. that all states in Q are reachable from q_{init}) and $W = \mathcal{L}(\mathcal{B})$ be the objective it recognizes. Our algorithm first verifies that the prefix preorder is total and recognizability by \mathcal{S}_∞ , and then, under these first two assumptions, progress-consistency. For each condition, we sketch an algorithm to decide it, and we discuss the time complexity of this algorithm.

Total prefix preorder. To check that W has a total prefix preorder, it suffices to check that the states of \mathcal{B} are totally preordered by $\preceq_{\mathcal{B}}$. We start by computing, for each pair of states $q, q' \in Q$, whether $q \preceq_{\mathcal{B}} q'$, $q' \preceq_{\mathcal{B}} q$, or none of these. This can be rephrased as an *inclusion problem* for two DBA-recognizable objectives: if $\mathcal{B}_q = (Q, C, q, \Delta, \alpha)$ and $\mathcal{B}_{q'} = (Q, C, q', \Delta, \alpha)$, we have that $q \preceq_{\mathcal{B}} q'$ if and only if $\mathcal{L}(\mathcal{B}_q) \subseteq \mathcal{L}(\mathcal{B}_{q'})$. Such a problem can be solved in time $\mathcal{O}(|C|^2 \cdot |Q|^2)$ [23]. We can therefore know for all $|Q|^2$ pairs $q, q' \in Q$ whether $q \preceq_{\mathcal{B}} q'$, $q' \preceq_{\mathcal{B}} q$, $q' \sim_{\mathcal{B}} q$ (as $\sim_{\mathcal{B}} = \preceq_{\mathcal{B}} \cap \succeq_{\mathcal{B}}$), or none of these in time $\mathcal{O}(|Q|^2 \cdot (|C|^2 \cdot |Q|^2)) = \mathcal{O}(|C|^2 \cdot |Q|^4)$. In particular, the prefix preorder is total if and only if for all $q, q' \in Q$, we have $q \preceq_{\mathcal{B}} q'$ or $q' \preceq_{\mathcal{B}} q$.

Recognizability by the prefix-classifier. After all the relations $\preceq_{\mathcal{B}}$ and $\sim_{\mathcal{B}}$ between pairs of states are computed in the previous step, we can compute the states and transitions of the prefix-classifier $\mathcal{S}_{\sim} = (Q_{\sim}, C, \tilde{q}_{\text{init}}, \Delta_{\sim})$ by merging all the equivalence classes for $\sim_{\mathcal{B}}$. We assume for simplicity that $Q_{\sim} = Q / \sim_{\mathcal{B}}$.

We now wonder whether it is possible to recognize W by carefully selecting a set α_{\sim} of Büchi transitions in \mathcal{S}_{\sim} . After a simple transformation of \mathcal{B} (called *saturation* [8, Section 2]), it actually suffices to try with the specific, easy-to-compute set of transitions of the prefix-classifier such that all corresponding transitions in the original DBA were Büchi:

$$\alpha_{\sim} = \{([q], c) \in Q_{\sim} \times C \mid \forall q' \in [q], (q', c) \in \alpha\}.$$

We then simply check whether $W = \mathcal{L}((Q_{\sim}, C, \tilde{q}_{\text{init}}, \Delta_{\sim}, \alpha_{\sim}))$, an equivalence query which, as discussed above, can be performed in time $\mathcal{O}(|C|^2 \cdot |Q|^2)$.

Progress-consistency. We assume that we have already checked that W is recognizable by a Büchi automaton built on top of \mathcal{S}_{\sim} , and that we know the (total) ordering of the states. We show that checking progress-consistency, under these two hypotheses, can be done in polynomial time. We state a lemma reducing the search for words witnessing that W is not progress-consistent to a known problem on regular languages.

► **Lemma 15.** *We assume that \mathcal{B} is built on top of the prefix-classifier \mathcal{S}_{\sim} and that the prefix preorder of W is total. Then, W is progress-consistent if and only if for all $q, q' \in Q$ with $q \prec_{\mathcal{B}} q'$, $\{w \in C^+ \mid \delta^*(q, w) = q'\} \cap \alpha\text{-FreeCycles}_{\mathcal{B}}(q') = \emptyset$.*

Notice that for each pair of states $q, q' \in Q$, the sets $\{w \in C^+ \mid \delta(q, w) = q'\}$ and $\alpha\text{-FreeCycles}_{\mathcal{B}}(q')$ are both regular languages recognized by deterministic finite automata with at most $|Q|$ states. The emptiness of their intersection can be decided in time $\mathcal{O}(|C|^2 \cdot |Q|^2)$ [52]. By Lemma 15, we can therefore decide whether \mathcal{B} is progress-consistent in time $\mathcal{O}(|Q|^2 \cdot (|C|^2 \cdot |Q|^2)) = \mathcal{O}(|C|^2 \cdot |Q|^4)$: for all $|Q|^2$ pairs of states $q, q' \in Q$, if $q \prec q'$, we test the emptiness of the intersection of these two regular languages.

4 Technical sketch

We discuss each direction of the proof of Theorem 10 (necessity and sufficiency of the conditions), for which complete arguments can be found in [8, Sections 4 & 5].

4.1 Necessity of the three conditions

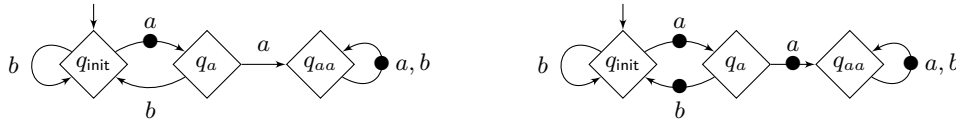
The necessity of the first two conditions (total prefix preorder and progress-consistency) is relatively straightforward: by contrapositive, we can use the words witnessing that these properties are not satisfied to build finite one-player arenas in which positional strategies do not suffice to play optimally.

For the third condition, we need to show the following statement.

► **Proposition 16.** *Let $W \subseteq C^\omega$ be a DBA-recognizable objective that is half-positional over finite one-player arenas. Then, W is recognized by a Büchi automaton built on top of \mathcal{S}_\sim .*

The proof of Proposition 16 makes extensive use of a “normal form” of Büchi automata verifying that any α -free path can be extended to an α -free cycle [8, Section 2]. Such a normal form can be produced by *saturating* a given DBA \mathcal{B} with Büchi transitions [41, 1, 2]. To do so, we add to α all transitions that do not appear in an α -free cycle of \mathcal{B} . This can be done by decomposing into strongly connected components the structure obtained by removing the Büchi transitions from \mathcal{B} .

In Figure 4, we show an intuitive example of the saturation process.



■ **Figure 4** A DBA (left) and its unique saturation (right).

The proof of Proposition 16 is split in two steps: we first give a proof for prefix-independent objectives, and then build on it for the general case.

► **Lemma 17.** *Let $W \subseteq C^\omega$ be a prefix-independent DBA-recognizable objective that is half-positional over finite one-player arenas. Then, there exists $F \subseteq C$ such that $W = \text{Büchi}(F)$.*

Proof sketch of Lemma 17. We assume that the objective W is recognized by a DBA $\mathcal{B} = (Q, C, q_{\text{init}}, \Delta, \alpha)$ (which has been saturated) and is prefix-independent, so all the states of \mathcal{B} are equivalent for \sim . The goal is to find a suitable definition for F , so that $W = \text{Büchi}(F)$. To do so, we exhibit a state q_{max} of \mathcal{B} that is “the most rejecting state of the automaton”: it satisfies that the set of α -free words from q_{max} contains the α -free words from all the other states (q_{max} is then called an *α -free-maximum*) and that the set of α -free cycles on q_{max} contains the α -free cycles on all the other states (it is also an *α -free-cycle-maximum*). We define F as the set of colors c such that $(q_{\text{max}}, c) \in \alpha$.

We first show that if an α -free-maximum exists, we can assume w.l.o.g. that it is unique. Then, we show the existence of an α -free-cycle-maximum. This part of the proof relies on the half-positionality over finite one-player arenas of W , as well as on the saturation of \mathcal{B} . Finally, defining F using q_{max} as explained above, we prove that $W = \text{Büchi}(F)$. ◀

We show how to reduce the general case to the prefix-independent case.

Proof sketch of Proposition 16. We now relax the prefix-independence assumption on W . If \mathcal{B} has exactly one state per equivalence class of \sim , it means that it is built on top of \mathcal{S}_\sim , and we are done. If not, let $q_\sim \in Q$ be a state such that $|[q_\sim]| \geq 2$. Our proof shows how to modify \mathcal{B} by “merging” all states in equivalence class $[q_\sim]$ into a single state, while still recognizing the same objective W . The main technical argument is to build a variant $W_{[q_\sim]}$ of objective W on a new set of colors $C_{[q_\sim]}$, that turns out to also be half-positional over finite one-player arenas and DBA-recognizable, but which is *prefix-independent*. We can therefore use the prefix-independent case and find $F_{[q_\sim]} \subseteq C_{[q_\sim]}$ such that $W_{[q_\sim]} = \text{Büchi}(F_{[q_\sim]})$. Then, we exhibit a state $q_{\text{max}} \in [q_\sim]$ whose α -free words are tightly linked to the elements of $F_{[q_\sim]}$. Finally, we show that it is still possible to recognize W while keeping only state q_{max} in $[q_\sim]$.

Once we know how to merge the equivalence class $[q_{\sim}]$ into a single state, we can simply repeat the operation for each equivalence class with multiple states, until we obtain a DBA built on top of \mathcal{S}_{\sim} . ◀

4.2 Sufficiency of the conditions

We now focus on the other direction of the proof of Theorem 10. We want to show the following statement.

► **Proposition 18.** *Let $W \subseteq C^{\omega}$ be an objective that has a total prefix preorder, is progress-consistent, and is recognizable by a DBA built on top of \mathcal{S}_{\sim} . Then, W is half-positional.*

The main technical tool to prove Proposition 18 is the notion of well-monotonic universal graph for an objective W , whose existence is sufficient to prove the half-positionality of W [48]. We will show how to build such a graph in our case.

Well-monotonic universal graphs. Let $\mathcal{G} = (V, E)$ be a graph and $W \subseteq C^{\omega}$ be an objective. A vertex v of \mathcal{G} satisfies W if for all infinite paths $v_0 \xrightarrow{c_1} v_1 \xrightarrow{c_2} \dots$ from v , we have $c_1 c_2 \dots \in W$.

Given two graphs $\mathcal{G} = (V, E)$ and $\mathcal{G}' = (V', E')$, a (graph) morphism from \mathcal{G} to \mathcal{G}' is a function $\phi: V \rightarrow V'$ such that $(v_1, c, v_2) \in E$ implies $(\phi(v_1), c, \phi(v_2)) \in E'$. A morphism ϕ from \mathcal{G} to \mathcal{G}' is W -preserving if for all $v \in V$, v satisfies W implies that $\phi(v)$ satisfies W . Notice that if $\phi(v)$ satisfies W , then v satisfies W , as any path $v \xrightarrow{c_1} v_1 \xrightarrow{c_2} \dots$ of \mathcal{G} implies the existence of a path $\phi(v) \xrightarrow{c_1} \phi(v_1) \xrightarrow{c_2} \dots$ of \mathcal{G}' – there are “more paths” in \mathcal{G}' .

A graph \mathcal{U} is (κ, W) -universal if for all graphs \mathcal{G} of cardinality $\leq \kappa$, there is a W -preserving morphism from \mathcal{G} to \mathcal{U} .

We consider a graph $\mathcal{G} = (V, E)$ along with a total order \leq on its vertex set V . We say that \mathcal{G} is *monotonic* if for all $v, v', v'' \in V$, for all $c \in C$,

- $(v \xrightarrow{c} v' \text{ and } v' \geq v'') \implies v \xrightarrow{c} v''$, and
- $(v \geq v' \text{ and } v' \xrightarrow{c} v'') \implies v \xrightarrow{c} v''$.

This means that (i) whenever there is an edge $v \xrightarrow{c} v'$, there is also an edge with color c from v to all states smaller than v' for \leq , and (ii) whenever $v \geq v'$, then v has at least the same outgoing edges as v' . Graph \mathcal{G} is *well-monotonic* if it is monotonic and the total order \leq is a well-order (i.e., any set of vertices has a minimum). Graph \mathcal{G} is *completely well-monotonic* if it is well-monotonic and there exists a vertex $\top \in V$ maximum for \leq such that for all $v \in V$, $c \in C$, $\top \xrightarrow{c} v$.

► **Theorem 19** (Consequence of [48, Theorem 1.1]). *Let $W \subseteq C^{\omega}$ be an objective. If for all cardinals κ , there exists a completely well-monotonic (κ, W) -universal graph, then W is half-positional (over all arenas).*

The exact result [48, Theorem 1.1] can actually be instantiated on more precise classes of arenas. However, we use it to prove here half-positionality of a family of objectives over *all* arenas, so the above result turns out to be sufficient.

Universal graphs for Büchi automata. We show that for a DBA-recognizable objective W , the three conditions from Theorem 10 imply half-positionality of W by providing a completely well-monotonic (κ, W) -universal graph for any κ .

Let $W \subseteq C^{\omega}$ be an objective with a total prefix preorder, that is progress-consistent, and that is recognized by a $\mathcal{B} = (Q, C, q_{\text{init}}, \Delta, \alpha)$ built on top of \mathcal{S}_{\sim} . We assume w.l.o.g. that \mathcal{B} is *saturated*, as in Section 4.1. For θ an ordinal, we build a graph $\mathcal{U}_{\mathcal{B}, \theta}$ as follows.

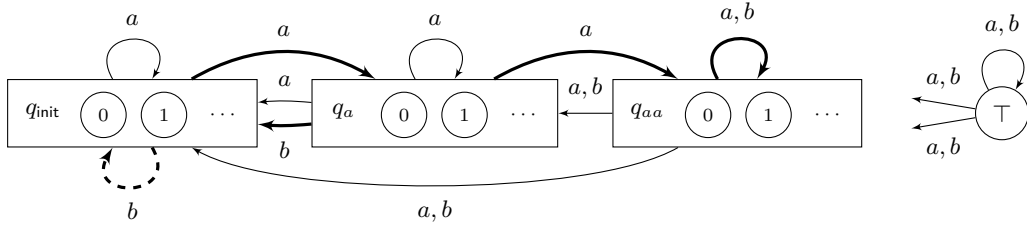
- We set the vertices as $U_{\mathcal{B},\theta} = \{(q, \lambda) \mid q \in Q, \lambda < \theta\} \cup \{\top\}$.
- For every transition $\delta(q, c) = q'$ of \mathcal{B} ,
 - if $(q, c) \in \alpha$, then for all ordinals λ, λ' , we define an edge $(q, \lambda) \xrightarrow{c} (q', \lambda')$;
 - if $(q, c) \notin \alpha$, then for all ordinals λ, λ' s.t. $\lambda' < \lambda$, we define an edge $(q, \lambda) \xrightarrow{c} (q', \lambda')$.
 - for $q'' \prec q'$, then for all ordinals λ, λ'' , we define an edge $(q, \lambda) \xrightarrow{c} (q'', \lambda'')$.
- For all $c \in C, v \in U_{\mathcal{B},\theta}$, we define an edge $\top \xrightarrow{c} v$.

We order the vertices lexicographically: $(q, \lambda) \leq (q', \lambda')$ if $q \prec q'$ or $(q = q' \text{ and } \lambda \leq \lambda')$, and we define \top as the maximum for \leq ($(q, \lambda) < \top$ for all $q \in Q, \lambda < \theta$).

Graph $\mathcal{U}_{\mathcal{B},\theta}$ is built such that on the one hand, it is sufficiently large and has sufficiently many edges so that there is a morphism from any graph \mathcal{G} (of cardinality smaller than some function of $|\theta|$) to $\mathcal{U}_{\mathcal{B},\theta}$. On the other hand, for the morphism to be W -preserving, at least some vertices of $\mathcal{U}_{\mathcal{B},\theta}$ need to satisfy W , which imposes a restriction on the infinite paths from vertices. Graph $\mathcal{U}_{\mathcal{B},\theta}$ is actually built so that for any automaton state $q \in Q$ and ordinal $\lambda < \theta$, the vertex (q, λ) satisfies $q^{-1}W$ [8, Section 5]. The intuitive idea is that for a non-Büchi transition $(q, c) \notin \alpha$ of the automaton such that $\delta(q, c) = q'$, a c -colored edge from a vertex (q, λ) in the graph either (i) reaches a vertex with first component q' , in which case the ordinal must decrease on the second component, or (ii) reaches a vertex with first component $q'' \prec q'$, with no restriction on the second component, but therefore with fewer winning continuations. Using progress-consistency and the fact that there is no infinitely decreasing sequence of ordinals, we can show that this implies that no infinite path in $\mathcal{U}_{\mathcal{B},\theta}$ corresponds to an infinite run in the automaton visiting only non-Büchi transitions.

We give an example of this construction.

► **Example 20.** We consider again the DBA \mathcal{B} from Example 7, recognizing the words seeing a infinitely many times, or a twice in a row at some point. We represent the graph $\mathcal{U}_{\mathcal{B},\theta}$, with $\theta = \omega$ in Figure 5. ┘



■ **Figure 5** The graph $\mathcal{U}_{\mathcal{B},\omega}$, where \mathcal{B} is the automaton from Example 7 ($\mathcal{L}(\mathcal{B}) = \text{Büchi}(\{a\}) \cup C^*aaC^\omega$). The dashed edge with color b indicates that $(q_{\text{init}}, \lambda) \xrightarrow{b} (q_{\text{init}}, \lambda')$ if and only if $\lambda' < \lambda$ (it corresponds to the only non-Büchi transition in \mathcal{B}). Elsewhere, an edge between two rectangles labeled q, q' with color c means that for all ordinals λ, λ' , $(q, \lambda) \xrightarrow{c} (q', \lambda')$. Thick edges correspond to the original transitions of \mathcal{B} . There are edges from \top to all vertices of the graph with colors a and b . Vertices are totally ordered from left to right.

We show that the graph $\mathcal{U}_{\mathcal{B},\theta}$ is completely well-monotonic (Lemma 21) and, for any cardinal κ , it is (κ, W) -universal for sufficiently large θ (Proposition 22) [8, Section 5].

► **Lemma 21.** *Graph $\mathcal{U}_{\mathcal{B},\theta}$ is completely well-monotonic.*

► **Proposition 22.** *Let κ be a cardinal, and θ' be an ordinal such that $\kappa < |\theta'|$. Let $\theta = |Q| \cdot \theta'$. Graph $\mathcal{U}_{\mathcal{B},\theta}$ is (κ, W) -universal.*

Thanks to these two results, we can show Proposition 18 using Theorem 19.

References

- 1 Bader Abu Radi and Orna Kupferman. Minimizing GFG transition-based automata. In *ICALP*, volume 132, pages 100:1–100:16, 2019. doi:10.4230/LIPIcs.ICALP.2019.100.
- 2 Bader Abu Radi and Orna Kupferman. Canonicity in GFG and transition-based automata. In *GandALF*, volume 326, pages 199–215, 2020. doi:10.4204/EPTCS.326.13.
- 3 Benjamin Aminof and Sasha Rubin. First-cycle games. *Inf. Comput.*, 254:195–216, 2017. doi:10.1016/j.ic.2016.10.008.
- 4 Dana Angluin and Dana Fisman. Regular ω -languages with an informative right congruence. *Inf. Comput.*, 278:104598, 2021. doi:10.1016/j.ic.2020.104598.
- 5 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
- 6 Alessandro Bianco, Marco Faella, Fabio Mogavero, and Aniello Murano. Exploring the boundary of half-positionality. *Ann. Math. Artif. Intell.*, 62(1-2):55–77, 2011. doi:10.1007/s10472-011-9250-1.
- 7 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.
- 8 Patricia Bouyer, Antonio Casares, Mickael Randour, and Pierre Vandenhovere. Half-positional objectives recognized by deterministic Büchi automata. *CoRR*, abs/2205.01365, 2022. arXiv:2205.01365, doi:10.48550/arXiv.2205.01365.
- 9 Patricia Bouyer, Ulrich Fahrenberg, Kim G. Larsen, Nicolas Markey, and Jiri 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.
- 10 Patricia Bouyer, Stéphane Le Roux, Youssef Oualhadj, Mickael Randour, and Pierre Vandenhovere. Games where you can play optimally with arena-independent finite memory. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 24:1–24:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.24.
- 11 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.
- 12 Patricia Bouyer, Youssef Oualhadj, Mickael Randour, and Pierre Vandenhovere. Arena-independent finite-memory determinacy in stochastic games. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 26:1–26:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CONCUR.2021.26.
- 13 Patricia Bouyer, Mickael Randour, and Pierre Vandenhovere. Characterizing omega-regularity through finite-memory determinacy of games on infinite graphs. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)*, volume 219 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:16, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.STACS.2022.16.
- 14 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.
- 15 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.

- 16 J. Richard Büchi and Lawrence H. Landweber. Definability in the monadic second-order theory of successor. *J. Symb. Log.*, 34(2):166–170, 1969. doi:10.2307/2271090.
- 17 J. Richard Büchi. On a decision method in restricted second order arithmetic. *Proc. Internat. Congr. on Logic, Methodology and Philosophy of Science*, pages 1–11, 1962.
- 18 Antonio Casares. On the minimisation of transition-based Rabin automata and the chromatic memory requirements of Muller conditions. In *CSL*, volume 216, pages 12:1–12:17, 2022. doi:10.4230/LIPIcs.CSL.2022.12.
- 19 Antonio Casares, Thomas Colcombet, and Karoliina Lehtinen. On the size of good-for-games Rabin automata and its link with the memory in Muller games. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, volume 229, pages 117:1–117:20, 2022. doi:10.4230/LIPIcs.ICALP.2022.117.
- 20 Krishnendu Chatterjee and Laurent Doyen. Energy parity games. *Theor. Comput. Sci.*, 458:49–60, 2012. doi:10.1016/j.tcs.2012.07.038.
- 21 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.
- 22 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.
- 23 Edmund M. Clarke, I. A. Draghicescu, and Robert P. Kurshan. A unified approach for showing language inclusion and equivalence between various types of omega-automata. *Inf. Process. Lett.*, 46(6):301–308, 1993. doi:10.1016/0020-0190(93)90069-L.
- 24 Thomas Colcombet, Nathanaël Fijalkow, Pawel Gawrychowski, and Pierre Ohlmann. The theory of universal graphs for infinite duration games. *CoRR*, abs/2104.05262, 2021. arXiv:2104.05262.
- 25 Thomas Colcombet, Nathanaël Fijalkow, and Florian Horn. Playing safe. 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 379–390. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.379.
- 26 Thomas Colcombet and Damian Niwiński. On the positional determinacy of edge-labeled games. *Theor. Comput. Sci.*, 352(1-3):190–196, 2006. doi:10.1016/j.tcs.2005.10.046.
- 27 Stefan Dziembowski, Marcin Jurdzinski, and Igor Walukiewicz. How much memory is needed to win infinite games? In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 – July 2, 1997*, pages 99–110. IEEE Computer Society, 1997. doi:10.1109/LICS.1997.614939.
- 28 Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *Int. Journal of Game Theory*, 8(2):109–113, 1979. doi:10.1007/BF01768705.
- 29 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377. IEEE Computer Society, 1991. doi:10.1109/SFCS.1991.185392.
- 30 Hugo Gimbert and Edon Kelmendi. Submixing and shift-invariant stochastic games. *CoRR*, abs/1401.6575, 2014. arXiv:1401.6575.
- 31 Hugo Gimbert and Wiesław 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.

- 32 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.
- 33 Yuri Gurevich and Leo Harrington. Trees, automata, and games. In Harry R. Lewis, Barbara B. Simons, Walter A. Burkhard, and Lawrence H. Landweber, editors, *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 60–65. ACM, 1982. doi:10.1145/800070.802177.
- 34 Nils Klarlund. Progress measures, immediate determinacy, and a subset construction for tree automata. *Ann. Pure Appl. Log.*, 69(2-3):243–268, 1994. doi:10.1016/0168-0072(94)90086-8.
- 35 Nils Klarlund and Dexter C. Kozen. Rabin measures and their applications to fairness and automata theory. In *LICS 1991*, pages 256–265, 1991.
- 36 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.
- 37 Eryk Kopczyński. Omega-regular half-positional winning conditions. 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 41–53. Springer, 2007. doi:10.1007/978-3-540-74915-8_7.
- 38 Eryk Kopczyński. *Half-positional Determinacy of Infinite Games*. PhD thesis, Warsaw University, 2008.
- 39 Alexander Kozachinskiy. Energy games over totally ordered groups. *CoRR*, abs/2205.04508, 2022. doi:10.48550/arXiv.2205.04508.
- 40 Alexander Kozachinskiy. One-to-two-player lifting for mildly growing memory. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPICs*, pages 43:1–43:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.STACS.2022.43.
- 41 Denis Kuperberg and Michal Skrzypczak. On determinisation of good-for-games automata. 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 299–310. Springer, 2015. doi:10.1007/978-3-662-47666-6_24.
- 42 Lawrence H. Landweber. Decision problems for omega-automata. *Math. Syst. Theory*, 3(4):376–384, 1969. doi:10.1007/BF01691063.
- 43 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 für Informatik, 2018. doi:10.4230/LIPICs.FSTTCS.2018.38.
- 44 Oded Maler and Ludwig Staiger. On syntactic congruences for omega-languages. *Theor. Comput. Sci.*, 183(1):93–112, 1997. doi:10.1016/S0304-3975(96)00312-X.
- 45 Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Inf. Control.*, 9(5):521–530, 1966. doi:10.1016/S0019-9958(66)80013-X.
- 46 Andrzej Włodzimierz Mostowski. Regular expressions for infinite trees and a standard form of automata. In Andrzej Skowron, editor, *Computation Theory – Fifth Symposium, Zaborów, Poland, December 3-8, 1984, Proceedings*, volume 208 of *Lecture Notes in Computer Science*, pages 157–168. Springer, 1984. doi:10.1007/3-540-16066-3_15.

- 47 A. Nerode. Linear automaton transformations. *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958. doi:10.2307/2033204.
- 48 Pierre Ohlmann. *Monotonic graphs for parity and mean-payoff games*. PhD thesis, IRIF – Research Institute on the Foundations of Computer Science, 2021.
- 49 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.
- 50 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October – 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.
- 51 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989. doi:10.1145/75277.75293.
- 52 Michael O. Rabin and Dana S. Scott. Finite automata and their decision problems. *IBM J. Res. Dev.*, 3(2):114–125, 1959. doi:10.1147/rd.32.0114.
- 53 Lloyd S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953. doi:10.1073/pnas.39.10.1095.
- 54 Ludwig Staiger. Finite-state omega-languages. *J. Comput. Syst. Sci.*, 27(3):434–448, 1983. doi:10.1016/0022-0000(83)90051-X.
- 55 Wolfgang Thomas. Church’s problem and a tour through automata theory. In Arnon Avron, Nachum Dershowitz, and Alexander Rabinovich, editors, *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *Lecture Notes in Computer Science*, pages 635–655. Springer, 2008. doi:10.1007/978-3-540-78127-1_35.
- 56 Klaus Wagner. On ω -regular sets. *Information and control*, 43(2):123–177, 1979.
- 57 Wiesław 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.

Two-Player Boundedness Counter Games

Emmanuel Filiot ✉

Université libre de Bruxelles, Belgium

Edwin Hamel-de le Court ✉

Université libre de Bruxelles, Belgium

Abstract

We consider two-player zero-sum games with winning objectives beyond regular languages, expressed as a parity condition in conjunction with a Boolean combination of boundedness conditions on a finite set of counters which can be incremented, reset to 0, but not tested. A boundedness condition requires that a given counter is bounded along the play. Such games are decidable, though with non-optimal complexity, by an encoding into the logic WMSO with the unbounded and path quantifiers, which is known to be decidable over infinite trees. Our objective is to give tight or tighter complexity results for particular classes of counter games with boundedness conditions, and study their strategy complexity. In particular, counter games with conjunction of boundedness conditions are easily seen to be equivalent to Streett games, so, they are CoNP-c . Moreover, finite-memory strategies suffice for Eve and memoryless strategies suffice for Adam. For counter games with a disjunction of boundedness conditions, we prove that they are in solvable in $\text{NP} \cap \text{CoNP}$, and in PTime if the parity condition is fixed. In that case memoryless strategies suffice for Eve while infinite memory strategies might be necessary for Adam. Finally, we consider an extension of those games with a max operation. In that case, the complexity increases: for conjunctions of boundedness conditions, counter games are EXPTIME-c .

2012 ACM Subject Classification Theory of computation \rightarrow Logic and verification; Theory of computation \rightarrow Automata over infinite objects

Keywords and phrases Controller synthesis, Game theory, Counter Games, Boundedness objectives

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.21

Related Version *Full Version:* <https://hal.archives-ouvertes.fr/hal-03626782> [21]

Funding This work was supported by the Fonds de la Recherche Scientifique – F.R.S.-FNRS under the MIS project F451019F. Emmanuel Filiot is a senior research associate at F.R.S.-FNRS.

Acknowledgements We warmly thank Jean-François Raskin for his valuable inputs.

1 Introduction

Games on graphs are a popular mathematical framework to reason on reactive synthesis problems [2, 9]: the system to synthesize is seen as a protagonist which must enforce a given specification (its winning objective) against any adversarial behaviour of its environment. In this framework, executions of reactive systems are modelled as infinite sequences alternating between actions of the systems and actions of its environment. In the ω -regular setting, the set of correct executions of reactive systems is modelled as an automaton, for example, a non-deterministic Büchi automaton, then determinized into a parity automaton. The synthesis problem then boils down to solving a game played on the graph of the parity automaton, where the goal of the protagonist (Eve) is to satisfy, in the long run, the parity condition whatever her opponent (Adam) does. Motivated by the synthesis of more complex systems, the literature is rich in extensions of this basic two-player zero-sum ω -regular setting: multiple players, imperfect information, quantitative objectives, infinite graphs ... (see [2, 9] for some references). In this paper, we follow this line of work and consider an extension of two-player games beyond ω -regularity: counter games with boundedness conditions.



© Emmanuel Filiot and Edwin Hamel-de le Court;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 21; pp. 21:1–21:23

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Counter games. In this paper, a *two-player counter game with boundedness objectives*, only called counter game hereafter, is given by a finite arena, called counter arena, whose vertices are labelled by counter operations over a finite set of counters C . Those operations can: increment a counter, reset it, or skip it (i.e. leave its value unchanged). We consider objectives given as Boolean combinations of *counter boundedness conditions*. For $c \in C$, the condition $\mathbb{B}(c)$ is satisfied by all infinite paths $\pi = v_0v_1 \dots$, called plays, such that for some $N \in \mathbb{N}$, the value of c along π is bounded by N . Note that the bound N is not uniform, in the sense that it depends on π , and as a consequence, the set of plays satisfying $\mathbb{B}(c)$ is not ω -regular in general. In this paper, we consider particular classes of Boolean combinations of boundedness conditions. Since they do not necessarily capture all ω -regular objectives, we also, by default, equip counter games with a parity condition.

Given an objective W , as a Boolean formula Φ over atoms $\mathbb{B}(c)$ for all $c \in C$, the goal of the protagonist, Eve, is to enforce plays which satisfy W and the parity condition, whatever the adversary, Adam, does. If she has a strategy to meet this objective, she is said to win the game. Counter games are zero-sum, meaning that the goal of Adam is to enforce the complementary objective. The goal of this paper is to build a fine understanding of counter games, by studying the problem of deciding the winner for important classes of counter games.

Motivations. On infinite words, classes of counter automata with boundedness conditions have appeared in various papers, e.g. in [6, 15, 3, 8]. The most relevant models in the context of counter games are the ω BS-automata of [6] and the max-automata of [8]. They are equipped with the same counter operations as the counter games of this paper, plus a max operation in the case of max-automata, and some boundedness conditions. As a consequence, winning objectives in counter games can naturally be expressed with these automata. However, while they are known to have decidable emptiness problem, not much is known when they are used to define objectives in two-player games. A motivation for this paper is to investigate this question, for games where the winning conditions is not given by such an automaton but where counter operations are explicitly given in the arena.

In the same line of works, max-automata, which are deterministic, are known by [3] to correspond to the logic WMSO+U, which extends weak MSO on infinite words with the *unbounded quantifier* $\mathbb{U}X$. A formula $\mathbb{U}X.\phi(X)$ holds if there are arbitrarily large sets X satisfying ϕ . An important and strong result by Bojańczyk states that the extension of WMSO+U to infinite binary trees and with a path quantifier which allows to quantify over infinite paths, has decidable satisfiability problem [7]. Since strategies are definable, modulo a tree encoding, in this latter logic, a direct consequence of this result is that two-player games with objectives given by max-automata are decidable (see also Example 2 of [7]). As a consequence, counter games with boundedness conditions are decidable, though with non-elementary complexity. We aim here at providing conceptually simpler arguments and insights to prove decidability (with tighter complexity results), for particular instances of boundedness conditions, instead of using the general result of [7].

Contributions. Our contributions are summarized in Fig. 1. We consider objectives given as a conjunction of a parity condition and a formula over atoms $\mathbb{B}(c)$ in the following classes: conjunctions, disjunctions, disjunctions of conjunctions, and negation-free formulas. We also consider the extension of counter games with a max operator which can assign a counter with the maximal value of several counters. The table also mentions the strategy complexity. For conditions in $\bigwedge \mathbb{B}$, counter games are easily proved to be interreducible in polynomial time

to Streett games, yielding CoNP-completeness [22]. More interestingly, we prove that when the number of counters is fixed, then, they are interreducible to parity games in polynomial time, using another reduction (Thm 5).

We then prove, in it is our main contribution, that for conditions in $\bigvee \mathbb{B}$, counter games are solvable in $\text{NP} \cap \text{CoNP}$ and in polynomial time when the index of the parity function is fixed. To prove this result, we introduce the notion of *finitely switching strategies* which are, to the best of our knowledge new, and we believe, interesting on their own. This notion is specifically designed for disjunctions of prefix-independent objectives (which is the case of counter boundedness conditions): in a finitely switching strategy, Eve announces which objective from the disjunction she aims to satisfy, and she can change her mind along the play, but only a finite number of times. Eventually, she is bound to satisfy one the objectives. We give general conditions to decide whether Eve has a finitely switching strategy in a two-player game with a disjunction of prefix-independent objectives, and prove that such strategies are sufficient for Eve to win objectives in $\bigvee \mathbb{B}$ and more generally in $\bigvee \bigwedge \mathbb{B}$.

Related works. Two-player games with boundedness conditions have been studied in the literature, first as finitary parity and Streett games [12], then generalized to cost-parity and cost-Streett games [20]. Finitary parity- and Streett-games are request-response games [13], with the additional constraint that the delay (number of edges) between a request and its response is bounded (by a bound which depends on the play). For cost-parity and cost-Streett, instead of the number of edges, costs (including 0) label edges and the delay is defined as the sum of the costs. Cost-parity and cost-Streett games can be encoded as counter games with conditions in $\bigwedge \mathbb{B}$, though with an exponential blowup. The difference between those counter games and finitary- and cost-games can be seen in their complexity: counter games with conditions in $\bigwedge \mathbb{B}$ are CoNP-c, finitary parity games are in PTIME, cost-parity in $\text{NP} \cap \text{CoNP}$, and finitary Streett and cost-Streett are EXPTIME-c.

Delay games with objectives given by a max-automaton have been proved to be decidable in [26]. This result is orthogonal to ours: first, those games allow for some delay, here in the sense that Eve has some look-ahead on Adam's future actions. Second, the decision procedure is non-elementary and rely on an encoding into WMSO+UP on infinite trees, some argument we avoid here, but for less expressive boundedness objectives.

Infinite-state games with boundedness conditions have been considered in [11], over pushdown arenas. Finitary games over these arenas are shown to be decidable, as well as (pushdown) counter games with conditions in $\bigwedge \mathbb{B}$, without complexity results. Interestingly, it is shown that those games are equivalent to games where the objective of Eve is to uniformly bound all counters, for a bound which only depends on her strategy, and not on the plays. For counter games in $\bigwedge \mathbb{B}$ over a *finite* arena, this result can easily be seen as a consequence that finite-memory strategies suffice for Eve.

Last but not least, counter boundedness games have appeared implicitly in some existing works on synthesis [1, 19], though the classes considered in these papers are less general and solved using specific techniques. In [19], the authors consider a synthesis problems over infinite alphabets of data. In particular, they study the problem of synthesising Mealy machines with *registers* satisfying specifications given as deterministic register automata over $(\mathbb{N}, <, 0)$. It is shown that this problem is decidable in 2EXPTIME , and, even though the decidability proof is not based on counter games, it is proved that the synthesis problem reduces to a game with winning conditions given as a (deterministic) max-automaton whose acceptance is a disjunction of a parity condition and a disjunction of conditions of the form “counter c is unbounded”. Although the main technical difficulty in [19] is to prove this reduction, based on it, our results on counter games with max operation yields an alternative procedure to decide the former synthesis problem (with same complexity).

Winning objective parity \wedge	Complexity	Memory of Eve	Memory of Adam	Theorem
$\bigwedge \mathbb{B}$	coNP-C	Finite	none	Th 3
$\bigvee \mathbb{B}$	NP \cap CoNP PTIME for fixed index	Parity Index	Infinite	Th 12
$\bigvee \bigwedge \mathbb{B}$	coNP-C	Finite	Infinite	Th 13
Bool ⁺ (\mathbb{B})	PSPACE, CoNP-H	Finite	Infinite	Th 14
$\bigwedge \mathbb{B} + \max$	EXPTIME-C	Finite	Finite	Th 15
Bool(\mathbb{B}) + \max	Decidable	Infinite	Infinite	from [5]

■ **Figure 1** Complexity of deciding whether Eve has a winning strategy in a counter game for various winning objectives, always taken in conjunction with a parity objective. Bool⁺(\mathbb{B}) means any negation-free Boolean combination of objectives of the form $\mathbb{B}(c)$. Hardness results hold for any parity function of fixed constant index. The notation $+\max$ indicates that counter games are also equipped with a max operation. Since counter games with boundedness objectives are determined, this yields the complexity of deciding whether Eve wins for the complementary objectives: for example, it is NP-C for objectives $\text{parity} \vee \bigvee \mathbb{U}$ and memoryless strategies are sufficient for Eve, and in PTIME for $\text{parity} \vee \bigwedge \mathbb{U}$ but infinite memory might be necessary for Eve.

The work of [1] considers a parameterized synthesis problem called the population control problem. In this problem, an arbitrary number of processes execute the same NFA, with the goal of reaching an accepting state. The controller picks an action (a letter) common to all of them, while the adversary resolves non-determinism for each of them individually. The problem is to decide whether “controller wins for any number of processes”. It is shown that this problem reduces to a finite graph game with a condition of the form “if the play has bounded capacity – where this bound depends on the play –, then the play satisfies some reachability condition” (see Sec 3.2 and Lemma 9 of [1]). Though the authors show that this condition can be equivalently replaced by an ω -regular one, it could also be directly encoded as a counter boundedness condition (with max operation). Our results combined with this reduction would however not provide the optimal complexity found in [1].

While our results on counter games do not provide new decidability results (nor better complexities) with respect to the two applications mentioned before, these two applications show that counter games with boundedness conditions arise naturally in synthesis problems, motivating our general study.

2 Preliminaries

For any set Σ , we denote by Σ^* (Σ^ω) the finite (infinite) sequences of elements of Σ .

Two-player arenas. A two-player *arena* is a tuple $\mathcal{A} = (V, E, V_\exists, V_\forall, v_0)$, where V is finite set, $E \subseteq V \times V$, and V_\exists and V_\forall are two subsets of V such that $\{V_\exists, V_\forall\}$ is a partition of V , and v_0 is an initial vertex. In this paper, we assume that arenas are deadlock-free, *i.e.* that for any $v \in V$, there exists $v' \in V$ such that $(v, v') \in E$. Given $v \in V$, we denote $\mathcal{A}[v] = (V, E, V_\exists, V_\forall, v)$ the arena \mathcal{A} where v_0 has been substituted by v . A play ρ of \mathcal{A} is a mapping from \mathbb{N} to V such that $(\rho(i), \rho(i+1)) \in E$, for all integers $i \in \mathbb{N}$. The set of plays is denoted by $\text{Plays}(\mathcal{A})$. Any play can also be seen as an element of V^ω , and we call a *history* any finite prefix of a play, and denote by $\text{Hist}(\mathcal{A})$ the set of histories of \mathcal{A} .

Strategies and finite-memory. A *strategy* for Eve (resp. Adam) is a function σ from $\text{Hist}(\mathcal{A})$ to V defined for all histories $h = h_0 \cdots h_n$ with $h_n \in V_{\exists}$ (resp. $h_n \in V_{\forall}$), and such that $(h_n, \sigma(h)) \in E$. A play ρ is *consistent* with a strategy for Eve (resp. Adam) if, for any integer n such that $\rho(n) \in V_{\exists}$ (resp. $\rho(n) \in V_{\forall}$), σ is defined on $\rho(0) \cdots \rho(n)$, and $\rho(n+1) = \sigma(\rho(0) \cdots \rho(n))$. We let $\text{Plays}(\mathcal{A}, \sigma)$ (or just $\text{Plays}(\sigma)$ when \mathcal{A} is clear from the context) the set of plays consistent with σ .

A strategy σ of Eve (resp. Adam) is said to be *finite-memory* if there exists a finite set M , an element $m_I \in M$, a mapping δ from $V \times M$ to V , and a mapping g from $V \times M$ to M such that the following is true. When $h = v_0 v_1 \cdots v_l$ is a prefix of a play consistent with σ such that $v_l \in V_{\exists}$ (resp. $v_l \in V_{\forall}$), and the sequence m_0, m_1, \dots, m_l is determined by $m_0 = m_I$ and $m_{i+1} = g(v_i, m_i)$, then $\sigma(w) = \delta(v_l, m_l)$. In that case, we say that (δ, g) is a *memory mapping pair* of σ , and that m_l is the memory state of g at move l . We also say that σ is of memory $|M|$, and *memoryless* if it is of memory 1. Note that a memoryless strategy can just be identified with a mapping from V to V .

Two-player games. A *winning condition* for \mathcal{A} is a subset $W \subseteq V^\omega$. A *strategy* σ of Eve or Adam is said to be winning for objective W if $\text{Plays}(\sigma) \subseteq W$. A *two-player game* is a pair $G = (\mathcal{A}, W)$ where \mathcal{A} is an arena and W is a winning condition. We say that a strategy (of Eve or Adam) is winning in G if it is winning for W . A game $G = (\mathcal{A}, W)$ is *determined* if either Eve wins G or Adam wins $(\mathcal{A}, V^\omega \setminus W)$.

In this paper, we consider the problem of deciding, given a game G with a finitely represented winning condition, whether Eve wins G . For a complexity class \mathcal{C} and a class of games \mathcal{G} , we say that games in \mathcal{G} are in \mathcal{C} (resp. \mathcal{C} -hard, \mathcal{C} -complete) if the latter problem for games $G \in \mathcal{G}$ is in \mathcal{C} (resp. \mathcal{C} -hard, \mathcal{C} -complete).

We also consider the complexity of strategies sufficient or necessary for Eve and Adam to win a game. We say that finite-memory strategies are *sufficient* for Eve (resp. Adam) to win \mathcal{G} if for all $G \in \mathcal{G}$, whenever Eve (resp. Adam) wins G , she has (resp. he has) a finite-memory winning strategy in G . We say that finite-memory is *necessary* for Eve (resp. Adam) to win \mathcal{G} if memoryless strategies do not suffice for Eve (resp. Adam) to win \mathcal{G} . Finally, we say that infinite-memory is *necessary* for Eve (resp. Adam) to win \mathcal{G} if finite-memory strategies do not suffice for Eve (resp. Adam) to win \mathcal{G} .

Parity games. Let \mathcal{A} be an arena with set of vertices V . Let $Q \subseteq \mathbb{N}$ be a finite set of elements called *colours* and $\kappa : V \rightarrow Q$ a mapping from vertices to colours called *parity function* or *priority function*. The size $|Q|$ of Q is called the *index* of κ . The mapping κ defines a winning condition denoted $\text{Parity}(\kappa)$, called a *parity condition*, as follows: $\text{Parity}(\kappa)$ is the set of all infinite words $w = w_0 w_1 \cdots \in V^\omega$ such that the greatest colour occurring an infinitely often in $\kappa(w_0) \kappa(w_1) \cdots$ is even. A parity game is a game whose winning condition is a parity condition. We refer to $\mathcal{A}' = (\mathcal{A}, Q, \kappa)$ as a *coloured arena*, and also denote $\text{Parity}(\kappa)$ as $\text{Parity}(\mathcal{A}')$ to avoid an explicit mention of the colouring κ . Note that a coloured arena $\mathcal{A}' = (\mathcal{A}, Q, \kappa)$ uniquely defines a parity game $G = (\mathcal{A}, \text{Parity}(\mathcal{A}'))$. It is well-known that parity games are in $\text{NP} \cap \text{CoNP}$ [17], and even solvable in quasi-polynomial time [10].

Counter operations. Our goal is now to define counter games. First, we introduce counter operations and their semantics. In the rest of the paper, we fix a countable set \mathcal{C} whose elements are called *counters*. A *counter operation* is a mapping from a finite subset C of \mathcal{C} to $\{i, r, \text{skip}\}$. We let $\text{Op}(C)$ denote the set of counter operations over $C \subseteq \mathcal{C}$. A *counter valuation* is a mapping ν from C to \mathbb{N} . For any infinite word $w \in \text{Op}(C)^\omega$, we define $\lambda(w)$

as the infinite sequence of counter valuations $\nu_0, \nu_1, \nu_2, \dots$ such that for any counter $c \in C$, $\nu_0(c) = 0$ and for any non-negative integer n , $\nu_{n+1}(c) = \nu_n(c) + 1$ if $w_n(c) = i$, $\nu_{n+1}(c) = 0$ if $w_n(c) = r$ and $\nu_{n+1}(c) = \nu_n(c)$ if $w_n(c) = \text{skip}$. We define $\lambda(w)$ for $w \in \text{Op}(C)^*$. To ease notations, we write $\lambda(w, c)_i$ instead of $\lambda(w)_i(c)$. We say that λ is the *evaluation* of w .

Counter games with boundedness objectives. Let \mathcal{A}' be an arena with set of vertices V , $C \subseteq \mathcal{C}$ a finite set of counters, and $\zeta : V \rightarrow \text{Op}(C)$ a mapping from vertices to counter operations, called *vertex labeling*. Let Q be a set of colours and $\kappa : V \rightarrow Q$ be a colouring of V . To avoid cumbersome notations, for any vertex $v \in V$ and counter $c \in C$, we let $\zeta_c(v)$ denote $(\zeta(v))(c)$. We refer to $\mathcal{A} = (\mathcal{A}', C, \zeta, Q, \kappa)$ as a *counter arena*, to \mathcal{A}' as its *underlying arena* and to (\mathcal{A}, Q, κ) as its underlying coloured arena. We let $\text{Parity}(\mathcal{A}) = \text{Parity}(\kappa)$.

We consider a particular type of winning objective for counter games, called boundedness conditions, always together with a parity condition. Let $c \in C$. We let $\mathbb{B}(c)$ be an atomic formula which intuitively requires that counter c is bounded along a play, by some constant. Formally, $\mathbb{B}(c)$ is interpreted in \mathcal{A} by the set of plays ρ of \mathcal{A} , denoted $\text{Plays}(\mathcal{A}, \mathbb{B}(c))$, such that the sequence $\lambda(\zeta(\rho), c)$ is bounded, *i.e.*

$$\text{Plays}(\mathcal{A}, \mathbb{B}(c)) = \{\rho \in \text{Plays}(\mathcal{A}) \mid \exists N \in \mathbb{N}, \forall n \in \mathbb{N}, \lambda(\zeta(\rho), c)_n \leq N\}$$

The set $\text{Plays}(\mathcal{A}, \mathbb{B}(c))$ is called a *boundedness condition*. To ease readability, we may just write $\mathbb{B}(c)$ to denote $\text{Plays}(\mathcal{A}, \mathbb{B}(c))$ when \mathcal{A} is clear from the context. We let $\mathbb{U}(c)$ as a shortcut for $\neg\mathbb{B}(c)$. A *counter condition* for \mathcal{A} is a Boolean formula ϕ over the set of propositions $\{\mathbb{B}(c) \mid c \in C\}$. Its interpretation $\text{Plays}(\mathcal{A}, \phi) \subseteq \text{Plays}(\mathcal{A})$ over \mathcal{A} is defined naturally.

Given a counter condition ϕ , the pair $G = (\mathcal{A}, \phi)$ is called a *counter game*. The game *induced* by $G = (\mathcal{A}, \phi)$ is the game $G_\phi = (\mathcal{A}', \text{Plays}(\mathcal{A}, \phi) \cap \text{Parity}(\mathcal{A}))$, where \mathcal{A}' is the underlying arena of \mathcal{A} . Note that in a counter game, both the counter condition and the parity condition must be satisfied. The notion of strategies and winning strategies carry over to counter games by considering the games they induce. In particular, Eve wins G if she wins G_ϕ , *i.e.*, she has a strategy winning for the objective $\text{Plays}(\mathcal{A}, \phi) \cap \text{Parity}(\mathcal{A})$.

In this paper, we consider several classes of counter conditions. The class of counter conditions of the form $\bigwedge_{c \in C} \mathbb{B}(c)$ for some finite set $C \subseteq \mathcal{C}$ is denoted $\bigwedge \mathbb{B}$. Similarly, we denote by $\bigvee \mathbb{B}$, $\bigvee \bigwedge \mathbb{B}$ and $\text{Bool}^+(\mathbb{B})$ the classes of counter conditions which are respectively, disjunctions of atoms $\mathbb{B}(c)$, disjunction of conjunctions of atoms $\mathbb{B}(c)$ (DNF), any negation-free Boolean formula.

► **Example 1.** First, Fig. 2 illustrates an example (left) with a disjunction of boundedness objectives (see the caption for details). Our second example is given by the 2-counter arena at the right of Fig. 2, where Adam controls all states. Adam has a strategy to win the objective $\bigwedge_{i=1,2} \mathbb{U}(c_i)$. Indeed, he can alternate between q_1 and q_2 by cycling longer and longer in one before cycling to the other. Notice that this strategy requires infinite memory.

► **Lemma 2.** *Counter games (with Boolean combinations of boundedness objectives) are determined and decidable.*

Proof. Given a counter arena \mathcal{A} and a counter c of \mathcal{A} , the set $\text{Plays}(\mathcal{A}, \mathbb{B}(c))$ is a Borel set. Indeed, it is equal to the countable union for all $N \geq 0$ of the sets

$$\text{Plays}_N(\mathcal{A}, \mathbb{B}(c)) = \{\rho \in \text{Plays}(\mathcal{A}) \mid \forall n \in \mathbb{N}, \lambda(\zeta(\rho), c)_n \leq N\}$$

which are ω -regular. Indeed, a Büchi automaton needs $|V| \times N \times |C|$ states to recognize $\text{Plays}_N(\mathcal{A}, \mathbb{B}(c))$. Since ω -regular sets are Borel, so is $\text{Plays}(\mathcal{A}, \mathbb{B}(c))$, as well as any Boolean combination of the latter. By Martin's determinacy theorem [24], the result follows.



■ **Figure 2** (Left) Counter arena $\mathcal{A} = (V, E, V_{\exists}, V_{\forall}, v)$ with $V_{\exists} = \{q_1, q_3\}$ and $V_{\forall} = \{q_2, q_4\}$, $v = 1$. There are two counters (c, d) whose updates are represented on the figure as pairs. We assume no parity condition and a counter condition $\mathbb{B}(c) \vee \mathbb{B}(d)$. From vertex 3, Eve has a memoryless winning strategy σ : always move to 4. Furthermore, she does not have a strategy from 1 to bound counter c , neither does she have a strategy from 1 to bound d . However, she has a memoryless strategy β winning for $\mathbb{B}(c) \vee \mathbb{B}(d)$: from 1, she moves to 2, and from 3 she moves to 4. If the play stays in $\{1, 2\}$, then d is bounded, and if the play eventually moves to 3, then c is bounded. (Right) A 2-counter arena with all states controlled by Adam and no parity condition.

To prove decidability, it suffices to notice that winning strategies in counter games are infinite trees such that all of their branches are accepted by a deterministic max-automaton as defined in [3]. Deterministic max-automata corresponds exactly to the logic WMSO+U over infinite words (weak MSO with the unbounding quantifier). WMSO+U has been extended to WMSO+UP on infinite trees with an additional quantifier over infinite paths (P). Therefore, winning strategies of two-player games with winning conditions definable in WMSO+U over infinite words are definable in WMSO+UP (see Ex. 2 of [5]). The result follows since WMSO+UP has decidable satisfiability problem [5]. ◀

3 Counter games with conjunctions of boundedness conditions

In this section, we study games with counter conditions in the class $\bigwedge \mathbb{B}$. Such games are easily shown to be decidable using known results. Indeed, we prove that they are equivalent in polynomial time to Streett games, known to be CoNP-complete [18], and in PTIME for a fixed number of Streett pairs [25]. This allows us to prove the following theorem:

► **Theorem 3.** *Counter games with winning conditions in $\bigwedge \mathbb{B}$ are CoNP-complete, and in PTIME if both the index of the priority function and the number of counters are fixed constants. Finite memory suffices for Eve and memoryless strategies suffice for Adam. CoNP-hardness holds even if the index of the parity function is any fixed constant.*

Sketch of proof. First, we define Streett games. Given an arena \mathcal{A} with set of vertices V , and a set of k pairs $S = \{(E_i, F_i) \mid 1 \leq i \leq k, E_i, F_i \subseteq V\}$, we let $\text{Streett}(S)$ be the set of words $w \in V^\omega$ such that for all $i = 1, \dots, k$, if w contains infinitely many occurrences of some $e \in E_i$, then it must contain infinitely many occurrences of some $f \in F_i$. A Streett game is a pair $G = (\mathcal{A}, W)$ where W is given as set of k Streett pairs S , i.e., $W = \text{Streett}(S)$. We prove that $\bigwedge \mathbb{B}$ -counter games are interreducible to Streett games in polynomial time. From any counter game G , we construct a Streett game $\Psi(G)$ with the same arena and for each counter c a pair (E_c, F_c) such that E_c is the set of vertices where c is incremented while F_c is the set of vertices where c is reset. The parity function of the counter game can also be split up into Streett pairs. If Eve wins G , it is obvious that Eve wins $\Psi(G)$ with the same winning strategy. For the converse, we use the fact that finite-memory strategies suffice to win Streett games (and memoryless strategies suffice for Adam) as shown in [25]. Any finite-memory Eve's strategy σ winning for $\Psi(G)$ is also winning for G . Indeed, if σ

has some play with some unbounded counter c , then c is necessarily incremented and reset infinitely often. Adam could then find a cycle (both on the arena and the memory structure of σ) containing at least one increment of c and no reset, iterate this cycle *ad infinitum*, and make Eve lose the Streett game if she plays σ . This contradicts that σ is winning. As a consequence, Eve wins G iff she wins $\Psi(G)$. To get the CoNP lower bound, we use the fact that Ψ is actually reversible, in polynomial time. ◀

Theorem 3 does not cover the case where only the number of counters is fixed. We prove that in this case, the complexity is at most $\text{NP} \cap \text{CoNP}$. Any Streett pair can be seen as a parity condition over colors $\{0, 1, 2\}$. Therefore, if in the latter transformation Ψ we use $\{0, 1, 2\}$ -parity conditions instead of Streett pairs and keep exactly the parity condition of G , we obtain that any $\wedge \mathbb{B}$ -counter game with a fixed number ℓ of counters, is equivalent (in the sense that it preserves the winner) to a game with a winning condition which is a conjunction of a fixed number ℓ of $\{0, 1, 2\}$ -parity conditions and a single arbitrary parity condition. We prove that such games are in turn reducible in polynomial time to parity games for $\ell = 1$ in the following lemma, later on applied recursively to show the result the result for any fixed ℓ (Theorem 5).

► **Lemma 4.** *Games of the form $G = (\mathcal{A}, W)$ where $W = \text{Parity}(\kappa) \cap \text{Parity}(\kappa_3)$ for κ an arbitrary colouring of index k and κ_3 a colouring in $\{0, 1, 2\}$, reduce in polynomial time to parity games of index $2k + 1$. Moreover, finite-memory strategies of memory size equal to k are sufficient for Eve to win G .*

Note that Lemma 4 entails that games with a conjunction of a parity condition of index k and a fixed number N of parity conditions over colors $\{0, 1, 2\}$ are solvable in $\text{NP} \cap \text{CoNP}$. Indeed, by iterating Lemma 4 N times, the latter games reduce to parity games of index $2^N(k + 1) - 1$ (and number of states exponential in N). Games with Boolean combinations of parity objectives have been studied in [14]. However, the former complexity result is not covered by [14]. As explained before, Lemma 4 implies the following theorem:

► **Theorem 5.** *For any fixed positive integer N , counter games of parity index k (which is not supposed to be fixed) with winning conditions in $\wedge \mathbb{B}$ and at most N counters, are in $\text{NP} \cap \text{CoNP}$ (and parity-hard). Finite memory strategies with memory size $2^{N-1}(k + 1) - 1$ suffice for Eve and Adam.*

4 Finitely switching strategies for games with disjunction of prefix-independent objectives

Let \mathcal{A} be an arena, let V be its set of vertices, and let \mathcal{W} be a finite set of prefix-independent¹ winning conditions for \mathcal{A} , i.e., $\mathcal{W} \subseteq 2^{V^\omega}$. We let $\bigvee \mathcal{W} = \bigcup \{W \mid W \in \mathcal{W}\}$. In this section, we consider a class of strategies for Eve, called *finitely switching*, whose existence entail that she wins $(\mathcal{A}, \bigvee \mathcal{W})$. We characterize the existence of finitely switching strategies via a least fixpoint and, for some particular classes of winning objectives $\bigvee \mathcal{W}$ of interest in this paper, prove that such strategies suffice for Eve to win $(\mathcal{A}, \bigvee \mathcal{W})$. The complexity of computing the fixpoint for those particular classes of objectives is deferred to Section 5.

Let us first give intuition on the notion of finitely switching strategies. In such a strategy, Eve announces an initial goal $W \in \mathcal{W}$ she wants to satisfy, but she may change her mind during the play, i.e., announce another goal $W' \in \mathcal{W}$, depending on what Adam does. She

¹ A winning condition W is prefix-independent if, for all $(w, u) \in (V^\omega, V^*)$, $w \in W$ iff $uw \in W$.

can do this only a finite number of times and eventually keep the same goal forever and satisfy it. Formally, for $k \geq 0$, a k -switching strategy for Eve is a strategy σ such that there exists a mapping goal from finite histories of σ to \mathcal{W} such that for all $\pi = v_1v_2 \cdots \in \text{Plays}(\sigma)$, there exists $W_1, \dots, W_{k+1} \in \mathcal{W}$ such that $\pi \in W_{k+1}$ and

$$\text{goal}(v_0)\text{goal}(v_0v_1)\text{goal}(v_0v_1v_2) \cdots \in W_1^*W_2^* \cdots W_k^*W_{k+1}^\omega$$

The goal W_{k+1} is called the *ultimate* goal of π . We say that σ is finitely switching if it is k -switching for some $k \geq 0$.

► **Example 6.** Consider the example of Fig. 2. The described strategy β is 1-switching for $\mathcal{W} = \{\mathbb{B}(c), \mathbb{B}(d)\}$: initially, her goal is $\mathbb{B}(d)$. If Adam ever tries to make it so that counter d gets unbounded, by going to vertex 3 from vertex 2, Eve can now set her new goal to $\mathbb{B}(c)$.

Consider now the 2-state arena of Example 1 in which Eve wants to satisfy $\bigvee_{c=1,2} \mathbb{U}(c)$. She has no finitely switching strategy: whenever she announces she wants to satisfy $\mathbb{U}(c_i)$ for some i , Adam loops on state q_{3-i} until Eve changes her mind. If her ultimate goal is $\mathbb{U}(c_i)$ for some i , then Adam will loop forever on q_{3-i} and c_i will be bounded, so that Eve does not meet the ultimate goal she announced. By seeing operations on c_1 and c_2 as priority functions, this example also shows that finitely switching strategies are not sufficient to win disjunctions of parity objectives in general. More precisely, for $i = 1, 2$, we can define the priority functions p_i which colors q_i by 0 and q_{3-i} by 1. If she ultimately announces her goal is to satisfy priority p_i , then Adam takes transition q_{3-i} forever and p_i sees infinitely many times color 1.

Since in a finitely switching strategy, any play consistent with that strategy must satisfy its ultimate goal, the following result is immediate:

► **Lemma 7 (Soundness).** *Any finitely switching strategy for Eve in \mathcal{A} is winning for $(\mathcal{A}, \bigvee \mathcal{W})$.*

We will see later on that the converse holds for some particular classes of boundedness objectives, but for now, let us characterize the existence of finitely switching strategies via some least fixpoint. For a set $X \subseteq V$, we denote the objective of reaching X by $\text{Reach}(X) = V^*XV^\omega$. We let f be the function which associates any $X \subseteq V$ to the set of vertices u from which Eve can win the objective $W \cup \text{Reach}(X)$ for some $W \in \mathcal{W}$. Formally, $f(X) = \{u \in V \mid \exists W \in \mathcal{W}, \text{Eve wins } (\mathcal{A}[u], W \cup \text{Reach}(X))\}$. Note that $X \subseteq f(X)$ for all $X \subseteq V$. Indeed, if $u \in X$, then Eve has a trivial strategy from u to reach X , and so $u \in f(X)$. Since $(2^V, \subseteq)$ is a complete lattice, by Knaster–Tarski theorem, f has a unique least fixpoint denoted $S^\mathcal{W}$. To compute $S^\mathcal{W}$, it suffices to compute the following sequence of sets until it stabilizes:

- $S_0^\mathcal{W} = \emptyset$,
- for $i \geq 0$, $S_{i+1}^\mathcal{W} = \{u \in V \mid \exists W \in \mathcal{W}, \text{Eve wins } (\mathcal{A}[u], W \cup \text{Reach}(S_i^\mathcal{W}))\}$.

For all $i \geq 1$ and $u \in S_i^\mathcal{W}$ (if it exists), we denote by $\sigma_{u,i}$ a strategy for Eve winning in the game $(\mathcal{A}[u], W \cup \text{Reach}(S_{i-1}^\mathcal{W}))$ for some $W \in \mathcal{W}$. It exists by definition of $S_i^\mathcal{W}$.

We now prove the following characterization.

► **Lemma 8 (Fixpoint characterization of finitely switching strategies).** *Let \mathcal{A} be an arena with set of vertices V and \mathcal{W} a finite set of prefix-independent winning conditions for \mathcal{A} . For all $u \in V$, the following are equivalent:*

1. *Eve has a finitely switching strategy from u*
2. *Eve has a $|V|$ -switching strategy from u*
3. $u \in S^\mathcal{W}$

Proof. Clearly $2 \Rightarrow 1$. We first prove $1 \Rightarrow 3$ and then $3 \Rightarrow 2$.

Let σ be a k -switching strategy for some $k \geq 0$. By induction on k , we prove that $u \in S_{k+1}^{\mathcal{W}}$. This implies the claim as $S_{k+1}^{\mathcal{W}} \subseteq S^{\mathcal{W}}$.

If $k = 0$, then Eve never changes her mind and therefore all plays of $\text{Plays}(\sigma)$ are in $\text{goal}(u)$ (the history with only the vertex u), so, $u \in S_1^{\mathcal{W}}$. Suppose that $k > 0$. We take $W = \text{goal}(u)$. Let $\pi \in \text{Plays}\sigma$. We prove that $\pi \in W \cup \text{Reach}(S_k^{\mathcal{W}})$. If Eve never changes her mind during π , then $\pi \in W$. Otherwise, let h the smallest prefix of π such that $\text{goal}(h) \neq W$. Let v be the last vertex of h . Note that the strategy² $\sigma|_h$ is a $(k-1)$ -switching strategy from v . By IH, $v \in \text{Reach}(S_k^{\mathcal{W}})$, which means that $\pi \in \text{Reach}(S_k^{\mathcal{W}})$ and we are done.

We now prove $3 \Rightarrow 2$. Let $u \in S^{\mathcal{W}}$. Let i be smallest index such that $u \in S_i^{\mathcal{W}}$. Note that $i \leq |V|$. We prove by induction on i that Eve has an $(i-1)$ -switching strategy $\beta_{u,i}$ witnessed by a goal function $\text{goal}_{u,i}$. If $u \in S_1^{\mathcal{W}}$, then $\sigma_{u,1}$ wins $(\mathcal{A}[u], W)$ for some $W \in \mathcal{W}$ and so we let $\text{goal}_{u,1}(h) = W$ for any history h of $\sigma_{u,1}$.

Suppose that $i > 1$ and $u \in S_i^{\mathcal{W}}$. Remind that the strategy $\sigma_{u,i}$ wins $(\mathcal{A}[u], W \cup \text{Reach}(S_{i-1}^{\mathcal{W}}))$. We modify $\sigma_{u,i}$ into a strategy $\beta_{u,i}$ as follows: $\beta_{u,i}$ is the same as $\sigma_{u,i}$ as long as $S_{i-1}^{\mathcal{W}}$ has not been reached. If eventually $S_{i-1}^{\mathcal{W}}$ is reached, say at a vertex v , then $\beta_{u,i}$ plays according to $\beta_{v,i-1}$ (which exists by IH).

We prove that $\beta_{u,i}$ is $(i-1)$ -switching. We let $\text{goal}_{u,i}(h) = W$ for any history h which does not visit $S_{i-1}^{\mathcal{W}}$. For any history $h = h_1 v h_2$ such that $|h_1|$ is minimal and $v \in S_{i-1}^{\mathcal{W}}$, we let $\text{goal}_{u,i}(h) = \text{goal}_{v,i-1}(v h_2)$. Let $\pi \in \text{Plays}(\beta_{u,i})$. If $\pi = v_0 v_1 \dots$ never visits $S_{i-1}^{\mathcal{W}}$, then $\text{goal}(v_0) \text{goal}(v_0 v_1) \dots \in W^\omega$, and $\pi \in W^\omega$. If there exists j minimal such that $v_j \in S_{i-1}^{\mathcal{W}}$, then, by IH, there exists $W_1, \dots, W_i \in \mathcal{W}$ such that $\text{goal}_{v_j,i-1}(v_j) \text{goal}_{v_j,i-1}(v_j v_{j+1}) \dots \in W_1^* \dots W_{i-1}^* W_i^\omega$. By definition of $\text{goal}_{u,i}$, we obtain that $\text{goal}_{u,i}(v_0) \text{goal}_{u,i}(v_0 v_1) \dots \in W^* W_1^* \dots W_{i-1}^* W_i^\omega$. Finally, it remains to prove that $\pi \in W_i$: by IH, its suffix $v_j v_{j+1} \dots$ is in W_i , and since W_i is prefix-independent, so is π , concluding the proof. \blacktriangleleft

According to Lemma 8, when Eve has a finitely switching strategy, then she has a $|V|$ -switching strategy. Interestingly, observe that the number of times she possibly needs to change her mind does not depend on the number of winning objectives in \mathcal{W} .

The proof of Lemma 8 constructs, for all $1 \leq i \leq |V|$ and $u \in S_i^{\mathcal{W}}$, a finitely switching strategy $\beta_{u,i}$, which either mimics $\sigma_{u,i}$ or switch to a strategy $\beta_{v,i-1}$. So, Eve needs to remember the current vertex u and index i , in order to know whether she must play according to $\sigma_{u,i}$ or to switch to a strategy $\beta_{v,i-1}$. So, even if for some N , all the strategies $\sigma_{u,i}$ are finite-memory of size at most N , $\beta_{u,i}$ needs memory $O(N \cdot |V|^2)$ in general. We now prove that Eve can do better.

► **Lemma 9** (Memory transfer). *Let \mathcal{A} be a counter arena, V be its set of vertices, and \mathcal{W} a finite set of prefix-independent winning conditions for \mathcal{A} . Let $N \in \mathbb{N}$ and suppose that for all $X \subseteq V$, $u \in V$ and $W \in \mathcal{W}$, strategies of memory size at most N suffice for Eve to win $(\mathcal{A}[u], W \cup \text{Reach}(X))$. Then for all $u \in S^{\mathcal{W}}$, Eve wins $(\mathcal{A}[u], \bigvee \mathcal{W})$ with memory at most N .*

The converse of Lemma 7 does not hold in general, as illustrated in Example 1 for disjunction of unboundedness objectives. However, we show here that it holds for disjunctions of conjunctions of boundedness objectives.

► **Lemma 10** (Completeness for boundedness conditions in DNF). *Let \mathcal{A} be a counter arena and C its set of counters. Let \mathcal{W} be a finite subset of counter conditions for \mathcal{A} in $\bigwedge \mathbb{B}$. If Eve wins the counter game $(\mathcal{A}, \bigvee \mathcal{W})$, then she has a finitely switching strategy from the initial vertex v_0 .*

² The restriction $\sigma|_h$ is defined by $\sigma|_h(h') = \sigma(hh')$ for all h' .

Proof. Let C_1, \dots, C_p be subsets of C such that \mathcal{W} is the set of all counter conditions $\bigwedge_{c \in C_i} \mathbb{B}(c)$, for $i \in \{1, \dots, p\}$. Suppose that Eve does not have a finitely switching strategy from the initial vertex v_0 . This means, by Lemma 8, that $v_0 \notin S^{\mathcal{W}}$. We construct a strategy for Adam winning the complementary objective $Comp = \bigcap_{i \in \{1, \dots, p\}} \left(\bigcup_{c \in C_i} \mathbb{U}(c) \cup \overline{\text{Parity}(\mathcal{A})} \right)$. By definition of $S^{\mathcal{W}}$, $f(S^{\mathcal{W}}) = S^{\mathcal{W}}$. Therefore, by definition of f , for any $v \in V \setminus S^{\mathcal{W}}$ and $i \in \{1, \dots, p\}$, Eve does not win the game $(\mathcal{A}[v], (\text{Parity}(\mathcal{A}[v]) \cap \bigcap_{c \in C_i} \mathbb{B}(c)) \cup \text{Reach}(S^{\mathcal{W}}))$. Moreover, notice that since $\text{Parity}(\mathcal{A}[v]) \cap \bigcap_{c \in C_i} \mathbb{B}(c)$ is a Borel set, so is $(\text{Parity}(\mathcal{A}[v]) \cap \bigcap_{c \in C_i} \mathbb{B}(c)) \cup \text{Reach}(S^{\mathcal{W}})$. Thus, by Martin's theorem, Adam has a winning strategy $\sigma_{v,i}$ in $\mathcal{A}[v]$ for the complementary objective $(\bigcup_{c \in C_i} \mathbb{U}(c) \cup \overline{\text{Parity}(\mathcal{A}[v])}) \cap \text{Reach}(S^{\mathcal{W}})$, for all $i \in \{1, \dots, p\}$. Let us now explain how intuitively we build a strategy for Adam winning for $Comp$. It is defined by breaking it down into the following steps:

- Adam begins by step $(1, 1)$: he follows strategy $\sigma_{v_0,1}$ until the play of the game reaches a vertex where the value of a counter of C_1 is 1. If that is never the case, then Adam follows $\sigma_{v_0,1}$ *ad. infinitum*. Notice that, if the value of every counter of C_1 is bounded by a certain integer, Adam wins, since the play does not belong to $\text{Parity}(\mathcal{A}[v_0]) = \text{Parity}(\mathcal{A})$.
- After completing step (i, j) in a vertex v , two cases arise:
 - If $j < p$, then Adam carries out step $(i, j + 1)$ by following $\sigma_{v,j+1}$ until the play of the game reaches a vertex where the value of a counter of C_{j+1} is i . If that is never the case, Adam follows $\sigma_{v,j+1}$ *ad. infinitum*, and he wins since the play then satisfies $\overline{\text{Parity}(\mathcal{A}[v])}$ starting from v , and thus $\overline{\text{Parity}(\mathcal{A})}$ globally.
 - If $j = p$, then Adam carries out step $(i + 1, 1)$ by following $\sigma_{v,1}$ until the play of the game reaches a vertex where the value of a counter of C_1 is $i + 1$. If that is never the case, Adam follows $\sigma_{v,1}$ *ad. infinitum*. ◀

5 Complexity of games with disjunctions of boundedness conditions

The next result gives sufficient conditions on a class of games \mathcal{G} , to guarantee decidability of the problem of deciding if Eve has a finitely switching strategy for a disjunction of objectives in the class. In this result, we assume that the winning objectives of \mathcal{G} are finitely represented in some way. This is the case of all classes to which we apply this lemma in the paper.

► **Lemma 11.** *Let $\mathcal{C} \in \{\text{PTIME}, \text{NP}, \text{coNP}, \text{EXPTIME}\}$. Let \mathcal{G} be a class of games with prefix-independent objectives, such that deciding whether, given $(\mathcal{A}, W) \in \mathcal{G}$, a vertex v of \mathcal{A} , and a subset X of vertices of \mathcal{A} , Eve wins $(\mathcal{A}[v], W \cup \text{Reach}(X))$, is in \mathcal{C} . Then, deciding, given an arena \mathcal{A} and a finite subset of winning conditions \mathcal{W} such that $\{(\mathcal{A}, W) \mid W \in \mathcal{W}\} \subseteq \mathcal{G}$, whether Eve has a winning finitely switching strategy for $(\mathcal{A}, \bigvee \mathcal{W})$, is in \mathcal{C} .*

Proof. Suppose first that $\mathcal{C} = \text{PTIME}$. From Lemma 8, Eve has a winning finitely switching strategy for $(\mathcal{A}, \bigvee \mathcal{W})$ if and only if the initial vertex v_0 of \mathcal{A} is in $S^{\mathcal{W}}$. Thus, we can decide whether Eve has a finitely switching strategy by recursively computing the $S_i^{\mathcal{W}}$, one after the other, until $S_i^{\mathcal{W}} = S_{i+1}^{\mathcal{W}} = S^{\mathcal{W}}$. In order to compute $S_{i+1}^{\mathcal{W}}$ from $S_i^{\mathcal{W}}$, we check for every vertex v of \mathcal{A} whether Eve wins the game $(\mathcal{A}[v], W \cup \text{Reach}(S_i^{\mathcal{W}}))$. Thus, since $S_{|V|}^{\mathcal{W}} = S^{\mathcal{W}}$, in order to compute $S^{\mathcal{W}}$, we only need to check, in ptime, whether Eve wins a game of the form $(\mathcal{A}[v], W \cup \text{Reach}(X))$ at most $|V| \times |V| \times |\mathcal{W}|$ times. As a consequence, the problem of deciding whether Eve has a winning finitely switching strategy for $(\mathcal{A}, \bigvee \mathcal{W})$ is in PTIME . We present this generic fixpoint algorithm in Algorithm 1, as it is useful to treat the other complexity cases. In that figure, slv is an algorithm that terminates in polynomial time, and such that $\text{slv}(\mathcal{A}, v, W, H)$ returns true if and only if Eve wins $(\mathcal{A}[v], W \cup \text{Reach}(H))$. The case where $\mathcal{C} = \text{EXPTIME}$ is similar to the case where $\mathcal{C} = \text{PTIME}$.

■ **Algorithm 1** Generic algorithm to check $v \in S^{\mathcal{W}}$.

```

SOLVE( $\mathcal{A}, \mathcal{W}$ )
//  $v_0$  is the initial vertex of  $\mathcal{A}$ 
//  $V = \{v_1, \dots, v_n\}$ 
//  $\mathcal{W} = \{W_1, \dots, W_p\}$ 
1.  $N \leftarrow n^2 \times p$ 
2.  $\alpha \leftarrow 0$ 
3.  $H_0, H_1, \dots, H_N \leftarrow \emptyset$ 
4. While  $\alpha < n$ 
5.   For  $i = 1, \dots, n$ 
6.   For  $j = 1, \dots, p$ 
7.     If  $\text{sly}(\mathcal{A}, v_i, W_j, H_\alpha)$ 
8.        $H_{\alpha+1} \leftarrow \{v_i\} \cup H_{\alpha+1}$ 
9.    $\alpha \leftarrow \alpha + 1$ 
10. Return ( $v_0 \in H_\alpha$ )

```

In the case where $\mathcal{C} = \text{NP}$, we transform the algorithm SOLVE into an ptime algorithm VERIF, which is defined as the algorithm SOLVE, except that line 7 is replaced by a call to a ptime verifier that Eve wins $(\mathcal{A}[v_i], W_j \cup \text{Reach}(H_\alpha))$ given a certificate. Notice that the algorithm given is directly written as an algorithm in NP, *i.e.* an algorithm that verifies if Eve wins given a certificate, and *not* as an algorithm in P with oracle NP. All the certificates needed for each call at line 7 are taken as input of the algorithm VERIF. This approach works because the algorithm VERIF returns **True** if and only if the answers to some well-chosen questions of the type “Does Eve win $(\mathcal{A}[v], W \cup \text{Reach}(X))$?” are true. The case where $\mathcal{C} = \text{coNP}$ is done in a similar way, but this time by guessing the complement of $S^{\mathcal{W}}$. ◀

We are now ready to prove complexity results for solving counter games with disjunction of boundedness objectives. We start with the case of $\bigvee \mathbb{B}$.

► **Theorem 12.** *Counter games with counter conditions in $\bigvee \mathbb{B}$ are in $\text{NP} \cap \text{coNP}$, and are in PTIME if the index of the colouring is fixed. A memory of size equal to the index of the colouring suffices for Eve, and infinite memory is required for Adam.*

Proof. Let G be a game over counter arena \mathcal{A} with set of counters C , initial vertex v_0 and objective $\bigvee \mathcal{W}$ where $\mathcal{W} = \{\text{Parity}(\mathcal{A}) \cap \mathbb{B}(c) \mid c \in C'\}$ for some $C' \subseteq C$. It should be clear that those conditions are prefix-independent, therefore, by Lemma 8 and Lemma 10, Eve wins G iff she has a finitely switching strategy iff $v_0 \in S^{\mathcal{W}}$. So, to check whether Eve wins G , it suffices to compute the fixpoint $S^{\mathcal{W}}$. We prove that each step of the fixpoint computation (line 7 in algorithm SOLVE) is done in $\text{NP} \cap \text{coNP}$, and in PTIME if the index of the colouring is fixed. By Lemma 11, the complexity statement of the theorem follows. It remains to show that for all subset $X \subseteq V$, any vertex $u \in V$ and any counter $c \in C'$, it is decidable in $\text{NP} \cap \text{coNP}$ (and in ptime for fixed parity) whether Eve wins the game $(\mathcal{A}[u], (\text{Parity}(\mathcal{A}) \cap \mathbb{B}(c)) \cup \text{Reach}(X))$. First, we evacuate the reachability condition, *i.e.*, reduce in ptime the latter problem to solving a game $(\mathcal{A}', \text{Parity}(\mathcal{A}') \cap \mathbb{B}(c))$. This is easily done by adding a sink state to \mathcal{A} reached whenever X is visited, with operation skip on c and priority 0. This reduction works for more general boundedness conditions. Finally, the game $(\mathcal{A}', \text{Parity}(\mathcal{A}') \cap \mathbb{B}(c))$ is solvable in $\text{NP} \cap \text{coNP}$ by Theorem 5, and in ptime for fixed parity, which is the case of \mathcal{A}' when the index of \mathcal{A} is fixed, because they have the same colours.

For Adam, infinite memory might be necessary to enforce the complementary objective, as illustrated by Example 1. For Eve, Theorem 5 states that a memory of size the index of the parity function is sufficient to solve the “local” games $(\mathcal{A}', \text{Parity}(\mathcal{A}') \cap \mathbb{B}(c))$, which can be translated back to strategies of same size in $(\mathcal{A}[u], (\text{Parity}(\mathcal{A}) \cap \mathbb{B}(c)) \cup \text{Reach}(X))$. Therefore, the memory transfer lemma (Lemma 9) yields the result. ◀

We now turn to games on arenas \mathcal{A} with conditions in $\bigvee \bigwedge \mathbb{B}$, i.e., where $\mathcal{W} = \{\text{Parity}(\mathcal{A}) \wedge \bigwedge_{c \in C_i} \mathbb{B}(c) \mid i = 1, \dots, n\}$ for C_1, \dots, C_n finite subsets of counters. The same reasoning as in the proof of Theorem 12 applies. The only difference here is that, to solve the “local” games of the fixpoint computation (line 7 of algorithm SOLVE), we rely on Theorem 3.

► **Theorem 13.** *Counter games with winning conditions in $\bigvee \bigwedge \mathbb{B}$ are CONP-complete. Finite memory suffices for Eve, and infinite memory is required for Adam.*

We conclude this section by the case of Boolean combination of boundedness objectives.

► **Theorem 14.** *Counter games with winning conditions in $\text{Bool}^+(\mathbb{B})$ are in PSPACE and CONP-hard. Finite memory suffices for Eve, and infinite memory is required for Adam.*

Proof. Any counter condition which is a positive boolean combination $\phi \in \text{Bool}^+(\mathbb{B})$ can be written in disjunctive normal form $\psi = \bigvee_{i \in \{1, \dots, p\}} \bigwedge_{c \in C_i} \mathbb{B}(c)$, where the C_i are subsets of \mathcal{C} . Let $\mathcal{W} = \{\text{Parity}(\mathcal{A}) \wedge \bigwedge_{c \in C_i} \mathbb{B}(c) \mid i = 1, \dots, n\}$. A direct application of Theorem 13 yields a CoNExpTime, because p might be exponential. Instead, we do not construct ψ explicitly. Recall that, from Theorem 3, counter games with counter conditions in $\bigwedge \mathbb{B}$ are in CONP, and thus in PSPACE. Thus, since it is well-known that, even if p may be exponential in the size of ϕ , we can enumerate \mathcal{W} in polynomial space, we can use this enumeration algorithm at line 6 of algorithm SOLVE in Algorithm 1 to compute the fixpoint $S^{\mathcal{W}}$ in polynomial space. As a consequence, the problem of deciding whether Eve has a winning finitely switching strategy for counter games with winning conditions in $\text{Bool}^+(\mathbb{B})$ is in PSPACE. Hence, the result follows because, as for Theorem 13, these strategies suffice for Eve. ◀

6 Extensions of counter games with max operation

In this section, we consider counter games where the players can, in addition, put into a counter the maximum value of a subset of counters. In other words, max-counter games are defined in the same exact way as counter games, the only difference being *counter operations* are now mappings from a finite subset C of \mathcal{C} to $\{i, r, \text{skip}\} \cup \{\max(c) \mid S \subseteq C\}$.

► **Theorem 15.** *Let \mathcal{G} be the class of counter games G with counter condition $\bigwedge_{c \in C} \mathbb{B}(c)$, where C is the set of counters of G . Given a game G in \mathcal{G} , the problem of deciding whether Eve wins G is EXPTIME-c. Finite memory is sufficient for Eve and Adam.*

Proof. For hardness, we reduce the emptiness problem of the intersection of n deterministic top-down tree automata, which is known to be EXPTIME-hard [16]. We first show PSPACE-hardness in the case of arenas where Adam plays no role, i.e., $V_{\forall} = \emptyset$. The proof is by reduction from the emptiness problem of the intersection of n DFA. The latter reduction is inspired from the proof that deterministic min-automata have PSPACE-c emptiness problem [8]. Using the fact that strategies are trees, we lift the latter reduction to tree automata. It is non-trivial but standard. The detailed proof is in Appendix, in Lemma 16.

It remains to show that solving a game in \mathcal{G} can be done in exponential time. The difficulty for solving a game G of \mathcal{G} comes from the fact that counters interact with each other, since the value of counters can “flow” from one to another via the max operation.

That was not case for $\wedge \mathbb{B}$ -counter games without max, which are CoNP-C , and we could track each counter separately, replacing each boundedness condition by a condition of the form “if c is incremented infinitely often, then it is reset infinitely often”. Here, we need to track sequences of counters that flow one into another, called traces. We rather solve games with the complementary objective, which is correct since max-counter games are determined (see Lemma 19 in Appendix). To formalize this idea, we use the notion of \mathbb{U} -automata, *i.e.* automata with counters accepting some positive boolean combination of unboundedness conditions, that is a notion very close to the notion of S -automata described in [6]. We define a (non-deterministic) \mathbb{U} -automaton \mathcal{B} with a single counter d and acceptance condition $\mathbb{U}(d)$ that guesses either a new trace, or a valid continuation to the current trace, at every move of a play of \mathcal{G} . Every operation on the counters of the trace are mimicked on d , and it accepts a play iff there exists a run such that d is unbounded. That same idea is already used in the proof of Theorem 1 of [4], from which this proof is inspired. So, solving G boils down to solving a game on the same arena but with objective given by the language $L(\mathcal{B})$.

We show that the class of games G with an objective given by a non-deterministic \mathbb{U} -automaton with an acceptance condition of the form $\bigvee \mathbb{U}$ is in EXPTIME . To that end, we convert \mathcal{B} into a non-deterministic parity automaton \mathcal{T} , which does not preserve the language, but preserves the existence of winning strategy for Eve: when playing on the arena of G , Eve wins the objective $\mathcal{L}(\mathcal{B})$ if and only if she wins the objective $\mathcal{L}(\mathcal{T})$. Correctness is ensured by a pumping-like argument based on the fact that finite-memory strategies are sufficient to win ω -regular games. The automata \mathcal{B} and \mathcal{T} are constructed in ptime from G . Then we determinize \mathcal{T} in exponential time, take its product with G , and obtain a classical parity game of exponential size and linear index. We can conclude since parity games with m edges, n vertices and index k can be solved in $O(mn^k)$ (see e.g. [14]). The detailed proof is in the Appendix, in Lemma 15. \blacktriangleleft

7 Future work

In this paper, we have proved new complexity results for important classes of counter games, with the aim of finely understanding why they are decidable. We observe that they are mainly two types of boundedness conditions, which require different techniques: conjunctions of boundedness conditions, which are equivalent to Streett games, and disjunction of boundedness conditions (for which we introduce the notion of finitely switching strategies). To emphasize this dichotomy, we note that even for a parity function of fixed index, counters games with conjunctions of boundedness conditions are CoNP-C , while they are in PTIME for disjunctions. By determinacy, those results also yield complexity bounds for the complementary classes of *unboundedness* objectives. For example, we get that games with conjunctions of objectives of the form $\mathbb{U}(c)$ can be solved in $\text{NP} \cap \text{CoNP}$ and that infinite memory is required. However, note that our counter games are always taken in conjunction with a parity condition. Therefore, in the complementary objectives, this parity condition is now taken in disjunction. We leave conjunction of parity and unboundedness objectives as future work. Another important direction is to consider classes of conditions that mix boundedness and unboundedness objectives. Since the techniques used to solve them individually are different, this would require new techniques. More generally, the only known upper bound for any Boolean combination (not necessarily negation-free) of boundedness objective is non-elementary. We believe there is space for improvement.

References

- 1 Nathalie Bertrand, Miheer Dewaskar, Blaise Genest, Hugo Gimbert, and Adwait Amit Godbole. Controlling a population. *Log. Methods Comput. Sci.*, 15(3), 2019. doi:10.23638/LMCS-15(3:6)2019.
- 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 Mikolaj Bojanczyk. Weak MSO with the unbounding quantifier. *Theory Comput. Syst.*, 48(3):554–576, 2011. doi:10.1007/s00224-010-9279-2.
- 4 Mikolaj Bojanczyk. Weak MSO with the unbounding quantifier. *Theory Comput. Syst.*, 48(3):554–576, 2011.
- 5 Mikolaj Bojanczyk. Weak MSO+U with path quantifiers over infinite trees. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 38–49. Springer, 2014. doi:10.1007/978-3-662-43951-7_4.
- 6 Mikolaj Bojanczyk and Thomas Colcombet. Bounds in w-regularity. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 285–296. IEEE Computer Society, 2006. doi:10.1109/LICS.2006.17.
- 7 Mikolaj Bojanczyk, Tomasz Gogacz, Henryk Michalewski, and Michal Skrzypczak. On the decidability of MSO+U on infinite trees. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 50–61. Springer, 2014. doi:10.1007/978-3-662-43951-7_5.
- 8 Mikolaj Bojanczyk and Szymon Torunczyk. Deterministic automata and extensions of weak MSO. In Ravi Kannan and K. Narayan Kumar, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, volume 4 of *LIPICs*, pages 73–84. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2009. doi:10.4230/LIPICs.FSTTCS.2009.2308.
- 9 Véronique Bruyère. Computer aided synthesis: A game-theoretic approach. In Émilie Charlier, Julien Leroy, and Michel Rigo, editors, *Developments in Language Theory - 21st International Conference, DLT 2017, Liège, Belgium, August 7-11, 2017, Proceedings*, volume 10396 of *Lecture Notes in Computer Science*, pages 3–35. Springer, 2017. doi:10.1007/978-3-319-62809-7_1.
- 10 Cristian S. Calude, Sanjay Jain, Bakhadyr Khossainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *STOC*, pages 252–263, 2017.
- 11 Krishnendu Chatterjee and Nathanaël Fijalkow. Infinite-state games with finitary conditions. In Simona Ronchi Della Rocca, editor, *Computer Science Logic 2013 (CSL 2013), CSL 2013, September 2-5, 2013, Torino, Italy*, volume 23 of *LIPICs*, pages 181–196. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPICs.CSL.2013.181.
- 12 Krishnendu Chatterjee, Thomas A. Henzinger, and Florian Horn. Finitary winning in omega-regular games. *ACM Trans. Comput. Log.*, 11(1):1:1–1:27, 2009. doi:10.1145/1614431.1614432.
- 13 Krishnendu Chatterjee, Thomas A. Henzinger, and Florian Horn. The complexity of request-response games. In *LATA*, volume 6638, pages 227–237, 2011.
- 14 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.

- 15 Thomas Colcombet and Christof Löding. The non-deterministic mostowski hierarchy and distance-parity automata. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfssdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II - Track B: Logic, Semantics, and Theory of Programming & Track C: Security and Cryptography Foundations*, volume 5126 of *Lecture Notes in Computer Science*, pages 398–409. Springer, 2008.
- 16 Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. *Tree Automata Techniques and Applications*, 2008. URL: <https://hal.inria.fr/hal-03367725>.
- 17 E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1-4 October 1991*, pages 368–377. IEEE Computer Society, 1991.
- 18 E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs. *SIAM J. Comput.*, 29(1):132–158, 1999.
- 19 Léo Exibard, Emmanuel Filiot, and Ayrat Khalimov. Church synthesis on register automata over linearly ordered data domains. In Markus Bläser and Benjamin Monmege, editors, *38th International Symposium on Theoretical Aspects of Computer Science, STACS 2021, March 16-19, 2021, Saarbrücken, Germany (Virtual Conference)*, volume 187 of *LIPIcs*, pages 28:1–28:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.
- 20 Nathanaël Fijalkow and Martin Zimmermann. Cost-parity and cost-streect games. In Deepak D’Souza, Telikepalli Kavitha, and Jaikumar Radhakrishnan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPIcs*, pages 124–135. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.124.
- 21 Emmanuel Filiot and Edwin Hamel-de Le Court. Two-player Boundedness Counter Games. working paper or preprint, March 2022. doi:10.4230/LIPIcs.
- 22 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata Logics, and Infinite Games: A Guide to Current Research*. Springer-Verlag, Berlin, Heidelberg, 2002.
- 23 Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. Pushdown specifications. In *LPAR*, volume 2514, pages 262–277, 2002.
- 24 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.
- 25 Nir Piterman and Amir Pnueli. Faster solutions of rabin and streect games. In *LICS*, pages 275–284, 2006.
- 26 Martin Zimmermann. Delay games with WMSO+U winning conditions. *RAIRO Theor. Informatics Appl.*, 50(2):145–165, 2016. doi:10.1051/ita/2016018.

A Detailed proofs of Section 6

In this section, we prove the following theorem:

► **Theorem 15.** *Let \mathcal{G} be the class of counter games G with counter condition $\bigwedge_{c \in C} \mathbb{B}(c)$, where C is the set of counters of G . Given a game G in \mathcal{G} , the problem of deciding whether Eve wins G is EXPTIME- c . Finite memory is sufficient for Eve and Adam.*

The proof of Theorem 15 is split into two parts, each covered by a different lemma. Lemma 16 gives the EXPTIME-hardness, and Lemma 22 gives the EXPTIME-easiness.

► **Lemma 16.** *Max-counter games with a single winning condition $\mathbb{B}(c)$ for some counter c , and no parity condition, are EXPTIME-hard.*

Proof of Lemma 16. We prove EXPTIME-hardness of max-counter games with no parity condition and a conjunction of boundedness conditions $\bigwedge_{c \in C} \mathbb{B}(c)$. This entails the result because one can always add a counter c_m which takes the maximal value of all other counters $c \in C$ at each step, so that $\bigwedge_{c \in C} \mathbb{B}(c)$ is satisfied iff $\mathbb{B}(c_m)$ is satisfied.

To prove the theorem for conjunctions of boundedness conditions, we reduce the problem, called $\bigcap_n DTOP$, of deciding if the intersection of n languages recognized by deterministic top-down tree automata (DTOP) is empty, which is known to be EXPTIME-c [23]. Before giving the EXPTIME-hardness proof, we first prove PSPACE-hardness for the particular class of counter games where $V_\forall = \emptyset$, i.e., where Adam plays no role. We reduce the problem of deciding if the intersection of n languages recognized by deterministic finite-automata (DFA) is empty. We call the latter problem $\bigcap_n DFA$. The proof is inspired by a PSPACE-hardness proof of deciding non-emptiness of the language recognized by a deterministic min-automaton [8]. Then we lift the reduction from $\bigcap_n DFA$ to the problem $\bigcap_n DTOP$, i.e., to trees, by using the branching nature of counter games induced by Adam.

Consider an alphabet Σ and n complete DFA $D_i = (\Sigma, Q_i, q_0^i, F_i, \delta_i)$ such that all Q_i are pairwise disjoint. We construct a counter arena $\mathcal{A}[D_1, \dots, D_n]$ with $V_\forall = \emptyset$ and a set C of $n + 1$ counters, and no parity condition, such that Eve has a strategy to satisfy objective $\bigwedge_{c \in C} \mathbb{B}(c)$ iff $\bigcap_i L(D_i) \neq \emptyset$. This construction is similar to that of [8], which is a reduction from the universality problem for NFA. We assume that Σ contains a symbol $\# \in \Sigma$ and for all i , $L(D_i) \subseteq (\Sigma - \#)^* \#$. The counter arena $\mathcal{A}[D_1, \dots, D_n]$ is defined by $V_\exists = \Sigma$ and $V_\forall = \emptyset$, and the set of transitions is $E = V_\exists \times V_\exists$. The vertex $\#$ is initial. The set of counters is $C = \{c_0\} \cup \{c_q \mid q \in Q_i, i = 1, \dots, n\}$, and they are updated as follows for $i = 1, \dots, n$, where $\max(\emptyset) = 0$:

- on vertex $f \neq \#$: for all $q \in Q_i$, $c_q := \max\{c_{q'} + 1 \mid \exists q' \in Q_i, \delta(q', f) = q\}$ and $c_0 := c_0 + 1$
- on vertex $\#$: $c_{q_0^i} := \max\{c_q \mid q \in Q_{i'} \text{ for some } i' \text{ and } \delta_{i'}(q, \#) \notin F_{i'}\}$, and the counters c_q for all $q \in Q_i \setminus \{q_0^i\}$ are reset, as well as c_0 .

Note that for $f \neq \#$, two operations are performed at once: increment counters $c_{q'}$ and take the max. This is done to simplify the presentation and can be simulated by doubling the number of vertices of the arena.

Now, observe that $\text{Plays}(\mathcal{A}[D_1, \dots, D_n]) = \#\Sigma^\omega$ and a strategy for Eve is nothing but an infinite word w in $\#\Sigma^\omega$. We prove the following claims:

▷ **Claim 17.** For all non-empty finite set $X \subseteq \bigcap_{i=1}^n L(D_i)$, any play in $\#.X^\omega$ satisfies $\bigwedge_{c \in C} \mathbb{B}(c)$.

▷ **Claim 18.** No play in $\# \cdot (\bigcup_{i=1}^n ((\Sigma - \#)^* \#) \setminus L(D_i))^\omega$ satisfies $\bigwedge_{c \in C} \mathbb{B}(c)$.

Proof of Claim 17. Let $m = \max\{|u| \mid u \in X\}$. Let $w = \#u_1u_2\dots$ such that for all $j \geq 1$, $u_j \in X$. We prove that w , which is a play of $\mathcal{A}[D_1, \dots, D_n]$ satisfies that all the counters are bounded by $2m$. First, note that each u_j is of the form $v_j\#$, because $u_j \in \bigcap_i L(D_i)$ and the DFA D_i are assumed to accept words where $\#$ is an endmarker. First, consider counter c_0 : it is reset every time $\#$ is read, so, its maximal value is bounded by m . Now, for all $j \geq 1$ and $q \in \bigcup_i Q_i$, we let $in_{j,q}$ be the value of counter c_q after prefix $\#u_1\dots u_{j-1}$ and $out_{j,q}$ is value after prefix $\#u_1\dots u_{j-1}v_j$. By definition of the counter updates, we have:

1. $in_{j,q} = 0$ for all $j \geq 1$ and q not initial
2. $in_{j,q_0^i} = \max\{out_{j-1,q} \mid q \in Q_{i'} \text{ for some } i' \text{ and } \delta_{i'}(q, \#) \notin F_{i'}\}$ for all $j \geq 1$
3. $out_{j,q} = in_{j,q_0^i} + |v_j|$ if $q \in Q_i$ for some i and there exists a run of D_i from q_0^i to q on v_j
4. otherwise, $out_{j,q} = |r|$ where r is a run of maximal length on a prefix of v_j , ending in q .

21:18 Two-Player Boundedness Counter Games

Let $q \in Q_i$ for some i such that $\delta_i(q, \#) \notin F_i$. For all $j \geq 1$, there is no run from q_0^i to q on v_j , since $u_j = v_j \# \in L(D_i)$. So, we are in case 4 above and we have $out_{j,q} \leq |v_j| \leq m$. From the latter fact and 2, we get that $in_{j,q_0^i} \leq m$ for all i, j . From that and 3, we get that $out_{j,q} \leq m + |v_j| \leq 2m$ for all j . So, all the counter have value at most $2m$ after each v_j , which concludes the proof that they are bounded. \triangleleft

Proof of Claim 18. Let w be a play of $\mathcal{A}[D_1, \dots, D_n]$ in $\# \cdot (\bigcup_{i=1}^n ((\Sigma - \#)^* \#) \setminus L(D_i))^\omega$. Then, $w = \#w_1\#w_2\#w_3\#\dots$ such that $w_j \in (\Sigma - \#)^*$ for all $j \geq 1$. Moreover, for all $j \geq 1$, there exists $i_j \in \{1, \dots, n\}$ such that $w_j \# \notin L(D_{i_j})$ and there exists a run of D_{i_j} on w_j from $q_0^{i_j}$ to some non-accepting state q_{i_j} . Denote by $in(i_j)$ the value of counter $c_{q_{i_j}}^0$ before reading $w_j\#w_{j+1}\dots$ in w , and by $out(i_j)$ the value of counter $c_{q_{i_j}}$ before reading $\#w_{j+1}\#w_{j+2}\dots$ in w . By definition of the counter updates, we have $out(i_1) \geq in(i_1) + |u|$, $out(i_2) \geq in(i_2) + |u|$, and so on. Moreover, $in(i_2) \geq out(i_1)$, $in(i_3) \geq out(i_2)$, and so on, since the states q_{i_j} are non-accepting. This yields that the sequence $(in(i_j))_j$ is unbounded, concluding the proof. \triangleleft

As a side note, observe that the two claims imply the following: $\bigcap_{i=1}^n L(D_i) \neq \emptyset$ iff there exists a word $w \in \#\Sigma^\omega$ which satisfies $\bigwedge_{c \in C} \mathbb{B}(c)$. Indeed, if there exists $u \in \bigcap_{i=1}^n L(D_i)$, then it suffices to apply Claim 1 to $X = \{u\}$. Conversely, if $\bigcap_{i=1}^n L(D_i) = \emptyset$, then $(\bigcup_{i=1}^n (\Sigma^* \setminus L(D_i)))^\omega = \Sigma^\omega$ and Claim 2 implies that no word of Σ^ω satisfy $\bigwedge_{c \in C} \mathbb{B}(c)$.

We now lift the latter reduction to (binary) trees. We let Σ be a finite alphabet containing a symbol $\#$ called a constant symbol, and all other symbols are called *binary* symbols. We let $\Sigma_2 = \Sigma - \#$ be the set of binary symbols. A Σ -tree is defined as a term where terms t are inductively defined by $t, t_1, t_2 ::= \# \mid f(t_1, t_2), f \in \Sigma_2$. The set of branches of a Σ -tree t is inductively defined as $br(\#) = \{\#\}$, and $br(f(t_1, t_2)) = \{(f, d).b \mid d \in \{1, 2\}, b \in br(t_d)\}$.

A *deterministic top-down tree automaton* is a tuple $\mathcal{T} = (Q, q_0, F, \delta)$ where Q is a finite set of states, $q_0 \in Q$ the initial state, $F \subseteq Q$ the final states, and $\delta : Q \times (\{\#\} \cup (\Sigma_2 \times \{1, 2\})) \rightarrow Q$ is a (total) transition function. We see \mathcal{T} as a DFA $DFA(\mathcal{T})$ recognizing a language in $(\Sigma_2 \times \{1, 2\})^* \#$ naturally as follows: $DFA(\mathcal{T}) = (Q, q_0, F, \delta')$ where for all $q \in Q$, for all $(f, d) \in \Sigma_2 \times \{1, 2\}$, $\delta'(q, f) = \text{proj}_d(\delta(q, f))$, with proj_d the d th projection, and $\delta'(q, \#) = \delta(q, \#)$, and we denote by $L_{br}(\mathcal{T})$ the language recognized by this DFA. The language of Σ -trees accepted by \mathcal{T} is the set

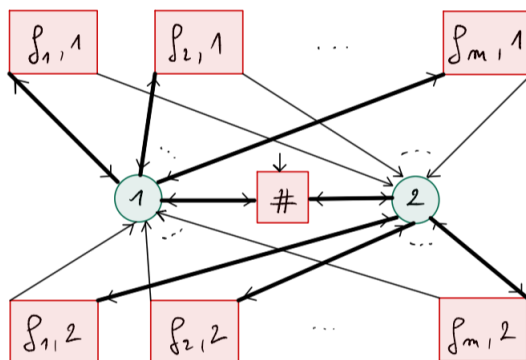
$$L(\mathcal{T}) = \{t \in \text{Trees}_\Sigma \mid br(t) \subseteq L_{br}(\mathcal{T})\}$$

Deciding³, given n DTOP $\mathcal{T}_1, \dots, \mathcal{T}_n$, whether $\bigcap_{i=1}^n L(\mathcal{T}_i) = \emptyset$ is EXPTIME-c [16].

Given $\mathcal{T}_1, \dots, \mathcal{T}_n$ such that $\mathcal{T}_i = (Q_i, q_0^i, F_i, \delta_i)$ for all i , we construct a max-counter game G winnable by Eve iff $\bigcap_{i=1}^n L(\mathcal{T}_i) \neq \emptyset$. The main idea of the proof is construct a game where Adam picks a direction $d \in \{1, 2\}$ (1 means left and 2 right), while Eve picks the labels in Σ . The arena $\mathcal{A}[\mathcal{T}_1, \dots, \mathcal{T}_n]$ of G (without the counters) is depicted on Fig. 3.

We now define counter conditions which make sure that if Eve has a strategy to keep all the counters bounded iff there exists $t \in \bigcap_i L(\mathcal{T}_i)$. For all i , let $\mathcal{T}_i = (Q_i, q_0^i, F_i, \delta_i)$. The set of counters is $C = \{c_q \mid q \in \bigcup_i Q_i\} \cup \{c_0\}$ (we assume wlog that all the sets Q_i are pairwise disjoint). Let us define counter updates. They are defined as for the arena $\mathcal{A}[DFA(\mathcal{T}_1), \dots, DFA(\mathcal{T}_n)]$. To simplify the presentation (and in particular the structure of the arena), we perform several operations at once. Let us define the updates, for all $1 \leq i \leq n$:

³ In [16], the definition of DTOP is slightly different, but less general: there are no accepting states but the transition function can be partial. A tree is accepted if there is a run on it which traverses the whole tree (it is not in an inner node). Those automata can easily be encoded into (our) DTOP by completing the transition function into a sink state q_s , declaring all states to be final but q_s .



■ **Figure 3** Arena for the proof of Theorem 16, where $\Sigma = \{\#, f_1, \dots, f_m\}$. Transitions in bold are in both directions. Square vertices are controlled by Adam, and the initial vertex is $\#$. When Adam picks a direction $d \in \{1, 2\}$, then Eve is forced to pick a vertex in $\Sigma_2 \times \{d\}$, or $\#$.

- on vertex $(f, j) \in \Sigma_2 \times \{1, 2\}$: for all $q_j \in Q_i$, $c_{q_j} := \max\{c_q + 1 \mid \exists q, q_{3-j} \in Q_i, \delta(q, f) = (q_1, q_2)\}$ and $c_0 := c_0 + 1$
- on vertex $\#$: $c_{q_0^i} := \max\{c_q \mid q \in Q_{i'}$ for some i' and $\delta_{i'}(q, \lambda) \notin F_{i'}\}$, and the counters c_q for all $q \in Q_i \setminus \{q_0^i\}$ are reset, as well as c_0 .
- on vertices $i \in \{1, 2\}$: counters are unchanged.

There is no parity condition and the counter condition is that the counters in C must be bounded. Let G be the constructed max-counter game. Before showing correctness, let us introduce some useful notation. Note that the histories and plays of G are elements of $\{\#\} \cup \Sigma_2 \times \{1, 2\}$ alternating with directions in $\{1, 2\}$. The following function removes the intermediate directions. Given $w = \lambda_1 d_1 \lambda_2 d_2 \dots \lambda_n d_n$ such that for all i , $\lambda_i \in \{\#\} \cup \Sigma_2 \times \{1, 2\}$ and $d_i \in \{1, 2\}$, we let $lab(w) = \lambda_1 \lambda_2 \dots \lambda_n$.

We now show correctness of the reduction. Suppose that there exists some $t \in \bigcap_i L(\mathcal{T}_i)$. We first define a strategy σ_t for Eve and then show it is winning in G . The strategy σ_t just mimics t : it plays as t dictates when a leaf of t is reached, its behaviour is reset to the root of t . Formally, the construction of σ_t satisfies the following invariant: all histories ending with an Eve vertex are words of the form $h = \#h_1 h_2 \dots h_k p d$ where:

- all h_i are such that $lab(h_i) \in br(t)$,
- $lab(p)$ is a prefix of a branch of t
- $d \in \{1, 2\}$

Given such a history h , we consider two cases: if $lab(p) \in br(t)$, then σ_t is reset to the root of t , which means that $\sigma_t(h) = (f, d)$ such that f is the label of the root of t . Otherwise, $\sigma_t(h) = \#$ if $lab(p)\# \in br(t)$, and $\sigma_t(h) = (f, d)$ if $lab(p)(f, d) \in br(t)$. Let us show that σ_t is winning. Let $\pi \in \text{Plays}(\sigma_t)$. First, we observe that $lab(\pi)$ is a play of $\mathcal{A}[DFA(\mathcal{T}_1), \dots, DFA(\mathcal{T}_n)]$. Let C' be the set of counters of the latter arena. By definition of σ_t , $lab(\pi)$ is of the form $\#b_1 \#b_2 \# \dots$ with infinitely many $\#$ such that for all $j \geq 1$, $b_j \#$ is a branch of t . Since $t \in \bigcap_i L(\mathcal{T}_i)$, we also get that $b_j \# \in \bigcap_i L(DFA(\mathcal{T}_i))$. The set $X = \{b_j \# \mid j \geq 1\}$ is finite since its elements correspond to branches of t . Therefore, by Claim 1, π satisfies $\bigwedge_{c \in C'} \mathbb{B}(c)$. We conclude by observing that $C = C'$, that $\mathcal{A}[\mathcal{T}_1, \dots, \mathcal{T}_n]$ has the same vertices as $\mathcal{A}[DFA(\mathcal{T}_1), \dots, DFA(\mathcal{T}_n)]$ plus the two vertices 1 and 2, with the same counter updates for their common vertices and no update on 1 and 2. Therefore, π satisfies $\bigwedge_{c \in C} \mathbb{B}(c)$ in $\mathcal{A}[\mathcal{T}_1, \dots, \mathcal{T}_n]$.

Conversely, suppose that $\bigcap_i L(\mathcal{T}_i) = \emptyset$. Take an arbitrary strategy σ of Eve. We show it is not winning. Intuitively, σ can be seen as an infinite tree. If there is a branch of the tree which visits $\#$ finitely many times, then σ is not winning because by following the directions corresponding to that branch, Adam can guarantee that counter c_0 is unbounded. So, we can assume that σ is such that all plays consistent with it sees infinitely many $\#$. We construct a play π of the form $\#h_1\#h_2\#\dots$ such that for all $j \geq 1$, there exists i such that $lab(h_j)\# \notin L(DFA(\mathcal{T}_i))$, and we conclude by Claim 2.

Consider the set of histories H_1 of σ which contains a $\#$ symbol only at their end. Clearly, H_1 can be identified with a Σ -tree t_1 . Since $t_1 \notin \bigcap_i L(\mathcal{T}_i)$, there exists i such that $t_1 \notin L(\mathcal{T}_i)$ and therefore, a history $h_1\# \in H_1$ such that $lab(h_1)\# \notin L(DFA(\mathcal{T}_i))$. To construct h_2, h_3, \dots , we proceed similarly. Let us explain how to construct h_2 . We let H_2 be the set of histories of the form $h_1\#g_2\#$ such that $h_1\#g_2\#$ is a history of σ such that g_2 does not contain $\#$. The set $(h_1\#)^{-1}H_2$ can be identified with a Σ -tree t_2 . Now, it suffices to take $h_2\# \in (h_1\#)^{-1}H_2$ such that $lab(h_2\#) \notin L(DFA(\mathcal{T}_i))$ for some $i = 1, \dots, n$. It exists since $t_2 \notin \bigcap_i L(\mathcal{T}_i)$. This concludes the proof. \blacktriangleleft

In order to prove Lemma 22, we first prove the following, in a very similar way to the proof of Lemma 2.

► **Lemma 19.** *Max-counter games (with Boolean combinations of boundedness objectives) are determined.*

Proof. Given a counter arena \mathcal{A} and a counter c of \mathcal{A} , the set $Plays(\mathcal{A}, \mathbb{B}(c))$ is a Borel set. Indeed, it is equal to the countable union for all $N \geq 0$ of the sets

$$Plays_N(\mathcal{A}, \mathbb{B}(c)) = \{\rho \in Plays(\mathcal{A}) \mid \forall n \in \mathbb{N}, \lambda(\zeta(\rho), c)_n \leq N\}$$

which are ω -regular. Indeed, a Büchi automaton that stores, in every state, the maximums between N and the value of each counter of C needs $|V| \times N^{|C|}$ states to recognize $Plays_N(\mathcal{A}, \mathbb{B}(c))$. Since ω -regular sets are Borel, so is $Plays(\mathcal{A}, \mathbb{B}(c))$, as well as any Boolean combination of the latter. By Martin's determinacy theorem [24], the result follows. \blacktriangleleft

Furthermore, to make the proof of Lemma 22 clearer, we now define two models of automata: non-deterministic \mathbb{U} -automata, and Parity-Rabin automata.

A (non-deterministic) \mathbb{U} -automaton B is a nine-tuple $(\Sigma, S, s_i, \Delta, Q, \kappa, C, \zeta, C_1)$, where Σ is an alphabet, S is a finite set of states, $s_i \in S$ is the initial state, $\Delta \in S \times S \times \Sigma$ is a transition function, Q is finite set of colors, κ is an alphabet colouring from Σ to Q , C is a finite set of counters, ζ is a state labeling from S to $\text{Op}(C)$, and C_1 is a subset of C . A run in B is an infinite word $\pi = y_0y_1 \dots \in \Delta^\omega$ such that $y_0 = s_i$, and such that the second element of each y_i is the first element of y_{i+1} for any non-negative integer i . We let $\text{States}(\pi)$ denote the word $v_0v_1 \dots$, where each v_i is the first element of y_i , and we let $\text{Input}(\pi)$ denote the word $z_0z_1 \dots$, where each z_i is the third element of y_i (*i.e.* the label of the edge y_i). A word w is *accepted* by B if either $w \in \text{Parity}(\kappa)$ (*i.e.* if the greatest color seen infinitely often in w is even), or if there exists a run π of B such that $\text{Input}(\pi) = w$ and such that $\text{States}(\pi)$ satisfies $\bigvee_{c \in C_1} \mathbb{U}(c)$. The language accepted by B is the set of accepted words.

A (non-deterministic) *Parity-Rabin automaton* D is a variant of a Rabin automaton, and is defined as a seven-tuple $(\Sigma, S, q_i, \Delta, Q, \kappa, \{\kappa_i\}_{i \in \{1, \dots, \ell\}})$ where $\Sigma, S, q_i, \Delta, Q$ and κ are defined in the same way as in the the case of \mathbb{U} -automata, where and $\{\kappa_i\}_{i \in \{1, \dots, \ell\}}$ is a finite set of colourings from S to $\{1, 2, 3\}$. Furthermore, a word w is *accepted* by D if and only if either w is in $\text{Parity}(\kappa)$, or there exists an integer $i \in \{1, \dots, \ell\}$ and a run ρ of D such that $\text{Input}(\rho) = w$ and such that $\text{States}(\rho)$ is in $\text{Parity}(\kappa_i)$. The language recognized by D , denoted $\mathcal{L}(D)$, is the set of words accepted by D .

► **Lemma 20.** *The language recognized by a non-deterministic Parity-Rabin automaton is ω -regular. Furthermore, games with an objective given by a non-deterministic Parity-Rabin automaton are solvable in EXPTIME.*

Proof. A Parity-Rabin automaton D can be converted into a non-deterministic automaton D_1 , with $\ell + 1$ colours whose domains are the set of states, by copying each state for every transition that goes to it, and transferring the colour κ to the states depending on which incoming transition the copy represents. The acceptance condition of D_1 is expressed by the union of the parity conditions induced by its colourings. The automaton D_1 can be further converted into a non-deterministic parity automaton D_2 with a single colouring, by copying it for every colouring it has, colouring the first copy with the first colouring, the second copy with the second colouring, etc. Thus, there exists a parity automaton D_2 that recognizes the same language as D , with a size polynomial in the size of D . One of the consequences of that statement is that $\mathcal{L}(D)$ is thus an ω -regular language. Furthermore, it is well-known that we can determinize D_2 into a deterministic parity automaton D_3 with exponential size and linear index, in exponential time. In addition, if G' is the game obtained from the product of a game G and the deterministic parity automaton D_3 , G' is a parity game of exponential size in the size of G and D , and index linear in the number of colours of D , such that Eve wins G' if and only if Eve wins G . Thus, since parity games with m edges, n vertices and index k can be solved in $O(mn^k)$ (see e.g. [14]), the class of games with an objective given by a non-deterministic Parity-Rabin automaton is in EXPTIME. ◀

We now show that the class of counter games with a counter condition given by a non-deterministic \mathbb{U} -automaton with an acceptance condition of the form $\bigvee \mathbb{U}$ is also decidable in EXPTIME, by converting them into Parity-Rabin automata.

► **Lemma 21.** *Let B be a non-deterministic \mathbb{U} -automaton with acceptance condition of the form $\bigvee_{c \in C_1} \mathbb{U}(c)$, and \mathcal{A} be a two-player arena. We can decide if Eve wins $(\mathcal{A}, \mathcal{L}(B))$ in EXPTIME.*

Proof. Let $B = (S, \Delta, i, \zeta, \bigvee_{c \in C_1} \mathbb{U}(c), \kappa)$. We construct in polynomial time a Parity-Rabin automaton D such that Eve wins $(\mathcal{A}, \mathcal{L}(B))$ if and only if Eve wins $(\mathcal{A}, \mathcal{L}(D))$. The idea is to keep the same automata structure as B , the same parity function, and to replace each atom $\mathbb{U}(c)$ by a parity function which is satisfied iff there is infinitely many increase of c and finitely many reset of c . So, for each counter c we introduce the parity function κ_c defined by:

$$\kappa_c(x) = \begin{cases} 1 & \text{if } \zeta_c(x) = \text{skip} \\ 2 & \text{if } \zeta_c(x) = i \\ 3 & \text{if } \zeta_c(x) = r \end{cases}$$

We show that Eve wins $(\mathcal{A}, \mathcal{L}(B))$ if and only if Eve wins $(\mathcal{A}, \mathcal{L}(D))$. If Eve wins $(\mathcal{A}, \mathcal{L}(D))$, then she wins $(\mathcal{A}, \mathcal{L}(B))$ with the same winning strategy, as $\mathcal{L}(D) \subseteq \mathcal{L}(B)$. Suppose now that σ is a winning strategy of Eve for $(\mathcal{A}, \mathcal{L}(B))$, and that Eve does not win $(\mathcal{A}, \mathcal{L}(D))$. However, by Lemma 20, $\mathcal{L}(D)$ is ω -regular and $(\mathcal{A}, \mathcal{L}(D))$ is thus determined. Therefore, Adam has a finite memory winning strategy τ for $(\mathcal{A}, \mathcal{L}(D))$. We exhibit a contradiction. Let ρ be a play of \mathcal{A} consistent with σ and τ . Then ρ satisfies both of the following properties:

1. $\rho \notin \text{Parity}(\kappa)$, and for any run π of D over ρ , for any counter c , if π sees infinitely many increase of c , then it sees infinitely many reset of c (because τ is winning)
2. either $\rho \in \text{Parity}(\kappa)$, or there exists a counter $c_0 \in C_1$, and there exists a run π of B over ρ such that $\text{States}(\pi)$ satisfies $\mathbb{U}(c)$ (it is because σ is winning)

21:22 Two-Player Boundedness Counter Games

Now, since $\rho \notin \text{Parity}(\kappa)$ from property 1, along ρ , c_0 is unbounded from property 2, so it sees infinitely many increase, and by property 1 it must see infinitely many reset. Intuitively, it implies that they are longer and longer segments in between two consecutive resets with more and more increase of c_0 . Since τ is finite-memory, Eve can find a cycle (both cycling on the arena, the memory-structure of the strategy and the automaton B) which contains at least one increase of c_0 and no reset. By iterating this cycle *ad infinitum*, she creates a play which is consistent with τ and a run of D over that new play, which sees infinitely many increase of c_0 but finitely many reset, contradicting Property 1.

Since D can be computed in polynomial time from B , and since the class of games with an objective given by a non-deterministic Parity-Rabin automaton is in EXPTIME, we can decide if Eve wins $(\mathcal{A}, \mathcal{L}(B))$ in EXPTIME. \blacktriangleleft

► **Lemma 22.** *Given a game in \mathcal{G} , the problem of deciding whether Eve wins \mathcal{G} is in EXPTIME. Finite memory is sufficient for Eve and Adam.*

Proof. We show that counter games G with counter condition of the form

$$\text{Plays} \left(\mathcal{A}, \bigvee_{c \in C} \mathbb{U}(c) \right) \cup \text{Parity}(\mathcal{A}),$$

where C is the set of counters of G , and \mathcal{A} its underlying two-player arena, can be solved in EXPTIME, which implies the lemma by Lemma 19.

We construct, from a max-counter game G , a game G' whose acceptance condition is \mathbb{U} -automaton D of size polynomial in the size of G .

Let G be a counter game with underlying two-player arena $\mathcal{A} = (V, E, V_\exists, V_\forall, v)$, vertex labeling ζ , set of colors Q , colouring κ , and winning condition $\text{Plays}(\mathcal{A}, \bigvee_{c \in C} \mathbb{U}(c)) \cup \text{Parity}(\mathcal{A})$. We construct a \mathbb{U} -automaton B , of size polynomial in $|C|$, with a single counter denoted d (we assume $d \notin C$), that recognizes the language of all words $w \in V^\omega$ such that either $w \in \text{Parity}(\kappa)$, or $\zeta(w)$ satisfies the condition $\bigvee_{c \in C} \mathbb{U}(c)$. To make the construction more easily understood, we first introduce the notion of trace. A *trace* of a word $w = z_0 z_1 \dots \in \mathbf{Op}(C)^\omega$ is a mapping θ from $\{i, \dots, j\}$ to C , where $i \leq j$ are two integers, such that, for any $l \in \{i, \dots, j-1\}$,

- either $\theta(l) = \theta(l+1)$ and $z_l(\theta(l)) \in \{\text{i}, \text{r}, \text{skip}\}$,
- or $\theta(l+1) \neq \theta(l)$ and $z_l(\theta(l+1)) = \max_{c \in S} (c)$ with $S \subseteq C$ and $\theta(l) \in S$.

The *value* of θ at move $t \in \{i, \dots, j\}$ is defined inductively as 0 if $t = i$, one plus the value at move $t-1$ if $z_{t-1}(\theta(t-1)) = \text{i}$, 0 if $z_{t-1}(\theta(t-1)) = \text{r}$, and the value at move $t-1$ otherwise. If a counter c reaches a value $N \geq 1$ at some point in w , then it is always possible to “track back”, with a trace of w , the sequence of counter operations which led to c having that value, by choosing, every time we go back to a previous counter operation of the type $c' = \max_{d \in S} (d)$ with $S \subseteq C$, the good counter d of S (the one with the maximum value), until reaching a counter whose value is 0. Thus, there exists a counter $c \in C$ and two integers t and N such that $\lambda(w, c)_t = N$ if and only if there exists a trace θ of w , such $\theta(t) = c$, and such that the value of θ at move t is N . As a consequence, there exists counter c such that $\lambda(w, c)$ is unbounded if and only if the values of the traces of w are unbounded.

This result allows us to define \mathcal{B} in the following way. The \mathbb{U} -automaton B works, on input w , by guessing all the possible traces of $\zeta(w)$, by using non-determinism. The value of a trace is stored inside the counter d . More precisely, every time B reads a letter, it either guesses a new trace, or guesses the next counter c' of C of the trace it is following, while applying, if c' is equal to the current counter c of the trace, the operation over c induced by

the letter read, to counter d . Thus, the \mathbb{U} -automaton B is constructed so that the value of d is unbounded if and only if there are traces of its input of arbitrarily large values. Moreover, we set the colouring of B as κ . Thus, B recognizes the language of all words $w \in V^\omega$ such that either $w \in \text{Parity}(\kappa)$, or $\zeta(w)$ satisfies the condition $\bigvee_{c \in C} \mathbb{U}(c)$, *i.e.* the language recognized

by B is the winning condition of the game G . The precise definition of B is given below.

We let $V_1 = C \times \{\text{i}, \text{r}, \text{skip}\}$, and $v_1 = (c, \text{r})$ where c is any counter in C . Furthermore, we let ζ^1 denote the mapping from V_1 to $\mathbf{Op}(\{d\})$ such that $(\zeta^1(c, \alpha))(d) = \alpha$, and E_1 denote the the smallest subset of $V_1 \times V_1 \times V$ such that, for any $\alpha \in \{\text{i}, \text{r}, \text{skip}\}$ and any $v \in V$, we have

- for any $c, c' \in C$, $((c, \alpha), (c', \text{r}), v) \in E_1$ (this comes from the fact that B should be able to guess a new trace at any time),
- for any $c \in C$, if $\zeta_c(v) \in \{\text{i}, \text{skip}\}$, $((c, \alpha), (c, \zeta_c(v)), v) \in E_1$ (the trace follows the increment or skip operation of a counter while updating d),
- for any $c \in C$, if $\zeta_c(v) = \max_{c' \in S}(c)$, then $((c', \alpha), (c, \text{skip}), v) \in E_1$, for any $c' \in S$ (the trace changes counters on a max operation while leaving d unchanged).

The \mathbb{U} -automaton B is the \mathbb{U} -automaton $(V, V_1, v_1, E_1, Q, \kappa, \{d\}, \zeta_1, \{d\})$. By Lemma 21 and Lemma 20, since B can be computed in a polynomial time from G , and since Eve wins G if and only if Eve wins $(\mathcal{A}, \mathcal{L}(B))$, we can decide if Eve wins G in EXPTIME. ◀

Different Strokes in Randomised Strategies: Revisiting Kuhn’s Theorem Under Finite-Memory Assumptions

James C. A. Main

F.R.S.-FNRS & UMONS – Université de Mons, Belgium

Mickael Randour

F.R.S.-FNRS & UMONS – Université de Mons, Belgium

Abstract

Two-player (antagonistic) games on (possibly stochastic) graphs are a prevalent model in theoretical computer science, notably as a framework for reactive synthesis.

Optimal strategies may require randomisation when dealing with inherently probabilistic goals, balancing multiple objectives, or in contexts of partial information. There is no unique way to define randomised strategies. For instance, one can use so-called *mixed* strategies or *behavioural* ones. In the most general settings, these two classes do not share the same expressiveness. A seminal result in game theory – *Kuhn’s theorem* – asserts their equivalence in games of perfect recall.

This result crucially relies on the possibility for strategies to use *infinite memory*, i.e., unlimited knowledge of all past observations. However, computer systems are finite in practice. Hence it is pertinent to restrict our attention to *finite-memory* strategies, defined as automata with outputs. Randomisation can be implemented in these in different ways: the *initialisation*, *outputs* or *transitions* can be randomised or deterministic respectively. Depending on which aspects are randomised, the expressiveness of the corresponding class of finite-memory strategies differs.

In this work, we study two-player turn-based stochastic games and provide a complete taxonomy of the classes of finite-memory strategies obtained by varying which of the three aforementioned components are randomised. Our taxonomy holds both in settings of perfect and imperfect information, and in games with more than two players.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases two-player games on graphs, stochastic games, Markov decision processes, finite-memory strategies, randomised strategies

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.22

Related Version *Full Version*: <https://arxiv.org/abs/2201.10825> [31]

Funding *James C. A. Main*: F.R.S.-FNRS Research Fellow.

Mickael Randour: F.R.S.-FNRS Research Associate, member of the TRAIL Institute.

1 Introduction

Games on graphs. Games on (possibly stochastic) graphs have been studied for decades, both for their own interest (e.g., [25, 20, 27]) and for their value as a framework for *reactive synthesis* (e.g., [28, 35, 9, 3]). The core problem is almost always to find *optimal strategies* for the players: strategies that guarantee winning for Boolean winning conditions (e.g., [26, 39, 12, 10]), or strategies that achieve the best possible payoff in quantitative contexts (e.g., [25, 5, 13]). In multi-objective settings, one is interested in *Pareto-optimal* strategies (e.g., [19, 38, 36, 23]), but the bottom line is the same: players are looking for strategies that guarantee the best possible results.



© James C. A. Main and Mickael Randour;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 22; pp. 22:1–22:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In reactive synthesis, we model the interaction between a system and its uncontrollable environment as a two-player antagonistic game, and we represent the specification to ensure as a winning objective. An optimal strategy for the system in this game then constitutes a formal blueprint for a *controller* to implement in the real world [3].

Randomness in strategies. In essence, a *pure strategy* is simply a function mapping histories (i.e., the past and present of a play) to an action deterministically.

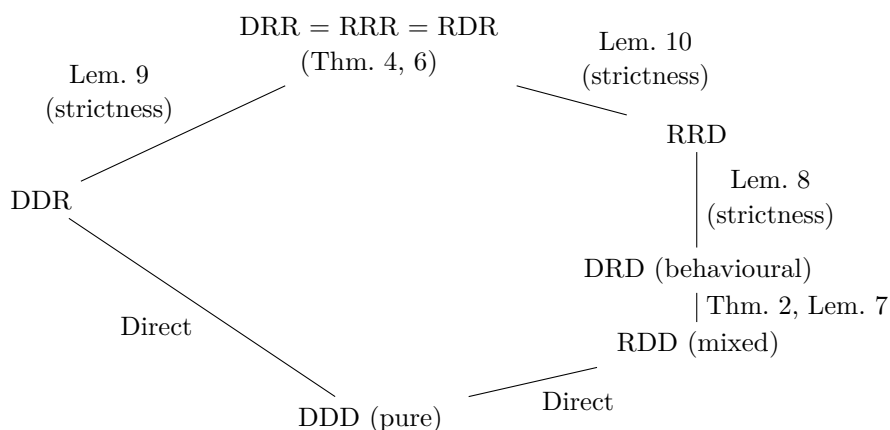
Optimal strategies may require *randomisation* when dealing with inherently probabilistic goals, balancing multiple objectives, or in contexts of partial information: see, e.g., [15, 36, 2, 23]. There are different ways of randomising strategies. For instance, a *mixed* strategy is essentially a probability distribution over a set of pure strategies. That is, the player randomly selects a pure strategy at the beginning of the game and then follows it for the entirety of the play without resorting to randomness ever again. By contrast, a *behavioural* strategy randomly selects an action at each step: it thus maps histories to probability distributions over actions.

Kuhn's theorem. In full generality, these two definitions yield different classes of strategies (e.g., [21], [34, Chapter 11]). Nonetheless, Kuhn's theorem [1] proves their equivalence under a mild hypothesis: in games of *perfect recall*, for any mixed strategy there is an equivalent behavioural strategy and vice-versa. A game is said to be of perfect recall for a given player if said player never forgets their previous knowledge and the actions they have played (i.e., they can observe their own actions). Let us note that perfect recall and *perfect information* are two different notions: perfect information is not required to have perfect recall.

Let us highlight that Kuhn's theorem crucially relies on two elements. First, mixed strategies can be distributions over an *infinite* set of pure strategies. Second, strategies can use *infinite memory*, i.e., they are able to remember the past completely, however long it might be. Indeed, consider a game in which a player can choose one of two actions in each round. One could define a (memoryless) behavioural strategy that selects one of the two actions by flipping a coin each round. This strategy generates infinitely many sequences of actions, therefore any equivalent mixed strategy needs the ability to randomise between infinitely many different sequences, and thus, infinitely many pure strategies. Moreover, infinitely many of these sequences require infinite memory to be generated (due to their non-regularity).

Finite-memory strategies. From the point of view of reactive synthesis, infinite-memory strategies, along with randomised ones relying on infinite supports, are undesirable for implementation. This is why a plethora of recent advances has focused on *finite-memory* strategies, usually represented as (a variation on) Mealy machines, i.e., finite automata with outputs. See, e.g., [27, 19, 11, 23, 4, 6]. Randomisation can be implemented in these finite-memory strategies in different ways: the *initialisation*, *outputs* or *transitions* can be randomised or deterministic respectively.

Depending on which aspects are randomised, the expressiveness of the corresponding class of finite-memory strategies differs: in a nutshell, *Kuhn's theorem crumbles when restricting ourselves to finite memory*. For instance, we show that some finite-memory strategies with only randomised outputs (i.e., the natural equivalent of behavioural strategies) cannot be emulated by finite-memory strategies with only randomised initialisation (i.e., the natural equivalent of mixed strategies) – see Lemma 7. Similarly, it is known that some finite-memory strategies that are encoded by Mealy machines using randomisation in all three components admit no equivalent using randomisation only in outputs [22, 21].



■ **Figure 1.1** Lattice of strategy classes in terms of expressible probability distributions over plays against all strategies of the other player. In the three-letter acronyms, the letters, in order, refer to the initialisation, outputs and updates of the Mealy machines: D and R respectively denote deterministic and randomised components.

Our contributions. We consider *two-player zero-sum stochastic games* (e.g., [37, 20, 32, 6]), encompassing two-player (deterministic) games and Markov decision processes as particular subcases. We establish a *Kuhn-like taxonomy* of the classes of finite-memory strategies obtained by varying which of the three aforementioned components are randomised: we illustrate it in Figure 1.1, and describe it fully in Section 3.

Let us highlight a few elements. Naturally, the least expressive model corresponds to pure strategies. In contrast to what happens with infinite memory, and as noted in the previous paragraph, we see that mixed strategies are strictly less expressive than behavioural ones. We also observe that allowing randomness both in initialisation and in outputs (RRD strategies) yields an even more expressive class – and incomparable to what is obtained by allowing randomness in updates only. Finally, the most expressive class is obviously obtained when allowing randomness in all components; yet it may be dropped in initialisation or in outputs without reducing the expressiveness – but not in both simultaneously.

To compare the expressiveness of strategy classes, we consider *outcome-equivalence*, as defined in Section 2. Intuitively, two strategies are outcome-equivalent if, against any strategy of the opponent, they yield identical probability distributions (i.e., they induce identical Markov chains). Hence we are agnostic with regard to the objective, winning condition, payoff function, or preference relation of the game, and with regard to how they are defined (e.g., colours on actions, states, transitions, etc).

Finally, let us note that in our setting of two-player stochastic games, the perfect recall hypothesis holds. Most importantly, we assume that actions are visible. Lifting this hypothesis drastically changes the relationships between the different models. While our main presentation considers two-player perfect-information games for the sake of simplicity, we argue in Section 6 that *our results hold in games of imperfect information* too, assuming visible actions, and that our results hold in games with more than two players.

Related work. There are three main axes of research related to our work.

The first one deals with the *various types of randomness* one can inject in strategies and their consequences. Obviously, Kuhn’s theorem [1] is a major inspiration, as well as the examples of differences between strategy models presented in [21]. On a different but related note, [16] studies when randomness is not helpful in games nor strategies (as it can be simulated by other means).

The second direction focuses on trying to characterise the *power of finite-memory strategies*, with or without randomness. One can notably cite [27] for memoryless strategies, and [30, 4, 6], and [7] for finite-memory ones in deterministic, stochastic, and infinite-arena games respectively.

The third axis concentrates on the use of *randomness as a means to simplify strategies* and/or reduce their memory requirements. Examples of this endeavour can be found in [14, 17, 29, 19, 33]. These are further motivations to understand randomised strategies even in contexts where randomness is not needed a priori to play optimally.

Outline. Due to space constraints, we only provide an overview of our work. All technical details and proofs can be found in the full version of this paper [31]. Section 2 summarises all preliminary notions. In Section 3, we present the taxonomy illustrated in Figure 1.1 and comment on it. We divide its proofs into two sections: Section 4 establishes the inclusions, and Section 5 proves their strictness. Finally, we discuss extensions of our results to games with imperfect information and multi-player games in Section 6.

2 Preliminaries

Probability. Given any finite or countable set A , we write $\mathcal{D}(A)$ for the set of probability distributions over A , i.e., the set of functions $p: A \rightarrow [0, 1]$ such that $\sum_{a \in A} p(a) = 1$. Similarly, given some set A and some σ -algebra \mathcal{F} over A , we denote by $\mathcal{D}(A, \mathcal{F})$ the set of probability distributions over the measurable space (A, \mathcal{F}) .

Games. We consider two-player stochastic games of perfect information played on graphs. We denote the two players by \mathcal{P}_1 and \mathcal{P}_2 . In such a game, the set of states is partitioned between the two players. At the start of a play, a pebble is placed on some initial state and each round, the owner of the current state selects an action available in said state and the next state is chosen randomly following a distribution depending on the current state and chosen action. The game proceeds for an infinite number of rounds, yielding an infinite play.

Formally, a (two-player) *stochastic game* (of perfect information) is defined as a tuple $\mathcal{G} = (S_1, S_2, A, \delta)$ where $S = S_1 \uplus S_2$ is a non-empty finite set of states partitioned into a set S_1 of states of \mathcal{P}_1 and a set S_2 of states of \mathcal{P}_2 , A is a finite set of actions and $\delta: S \times A \rightarrow \mathcal{D}(S)$ is a (partial) probabilistic transition function. For any state $s \in S$, we write $A(s)$ for the set of actions available in s , which are the actions $a \in A$ such that $\delta(s, a)$ is defined. We assume that for all $s \in S$, $A(s)$ is non-empty, i.e., there are no deadlocks in the game.

A *play* of \mathcal{G} is a sequence $s_0 a_0 s_1 \dots \in (SA)^\omega$ such that for all $k \in \mathbb{N}$, $\delta(s_k, a_k)(s_{k+1}) > 0$. A *history* is a finite prefix of a play ending in a state. Given a play $\pi = s_0 a_0 s_1 a_1 \dots$ and $k \in \mathbb{N}$, we write $\pi|_k$ for the history $s_0 a_0 \dots a_{k-1} s_k$. For any history $h = s_0 a_0 \dots a_{k-1} s_k$, we let $\text{last}(h) = s_k$. We write $\text{Plays}(\mathcal{G})$ to denote the set of plays of \mathcal{G} , $\text{Hist}(\mathcal{G})$ to denote the set of histories of \mathcal{G} and $\text{Hist}_i(\mathcal{G}) = \text{Hist}(\mathcal{G}) \cap (SA)^* S_i$ for the set of histories ending in states controlled by \mathcal{P}_i . Given some initial state $s_{\text{init}} \in S$, we write $\text{Plays}(\mathcal{G}, s_{\text{init}})$ and $\text{Hist}(\mathcal{G}, s_{\text{init}})$ for the set of plays and histories starting in state s_{init} respectively.

An interesting class of stochastic games which has been extensively studied is that of *deterministic games*; a game $\mathcal{G} = (S_1, S_2, A, \delta)$ is a deterministic game if for all $s \in S$ and $a \in A(s)$, $\delta(s, a)$ is a Dirac distribution. Another interesting class of games is that of one-player games. A game $\mathcal{G} = (S_1, S_2, A, \delta)$ is a one-player game of \mathcal{P}_i if S_{3-i} is empty, i.e., all states belong to \mathcal{P}_i . These one-player games are the equivalent of *Markov decision processes* in our context, and will be referred to as such.

Strategies and outcomes. A strategy is a function that describes how a player should act based on a history. Players need not act in a deterministic fashion: they can use randomisation to select an action. Formally, a *strategy* of \mathcal{P}_i is a function $\sigma_i: \text{Hist}_i(\mathcal{G}) \rightarrow \mathcal{D}(A)$ such that for all histories h and all actions $a \in A$, $\sigma_i(h)(a) > 0$ implies $a \in A(\text{last}(h))$. In other words, a strategy assigns to any history ending in a state controlled by \mathcal{P}_i a distribution over the actions available in this state.

When both players fix a strategy and an initial state is decided, the game becomes a purely stochastic process (a Markov chain). Let us recall the relevant σ -algebra for the definition of probabilities over plays. For any history h , we define $\text{Cyl}(h) = \{\pi \in \text{Plays}(\mathcal{G}) \mid h \text{ is a prefix of } \pi\}$, the *cylinder* of h , consisting of plays that extend h . Let us denote by $\mathcal{F}_{\mathcal{G}}$ the σ -algebra generated by all cylinder sets.

Let σ_1 and σ_2 be strategies of \mathcal{P}_1 and \mathcal{P}_2 respectively and $s_{\text{init}} \in S$ be an initial state. We define the probability measure (over $(\text{Plays}(\mathcal{G}), \mathcal{F}_{\mathcal{G}})$) induced by playing σ_1 and σ_2 from s_{init} in \mathcal{G} , written $\mathbb{P}_{s_{\text{init}}}^{\sigma_1, \sigma_2}$, in the following way: for any history $h = s_0 a_0 \dots s_n \in \text{Hist}(\mathcal{G}, s_{\text{init}})$, the probability assigned to $\text{Cyl}(h)$ is given by the product

$$\mathbb{P}_{s_{\text{init}}}^{\sigma_1, \sigma_2}(\text{Cyl}(h)) = \prod_{k=0}^{n-1} \tau_k(s_0 a_0 \dots s_k)(a_k) \cdot \delta(s_k, a_k, s_{k+1}),$$

where $\tau_k = \sigma_1$ if $s_k \in S_1$ and $\tau_k = \sigma_2$ otherwise. For any history $h \in \text{Hist}(\mathcal{G}) \setminus \text{Hist}(\mathcal{G}, s_{\text{init}})$, we set $\mathbb{P}_{s_{\text{init}}}^{\sigma_1, \sigma_2}(\text{Cyl}(h)) = 0$. By Carathéodory's extension theorem [24, Theorem A.1.3], the measure described above can be extended in a unique fashion to $(\text{Plays}(\mathcal{G}), \mathcal{F}_{\mathcal{G}})$.

Let σ_1 be a strategy of \mathcal{P}_1 . A play or prefix of play $s_0 a_0 s_1 \dots$ is said to be *consistent* with σ_1 if for all indices k , $s_k \in S_1$ implies $\sigma_1(s_0 a_0 \dots s_k)(a_k) > 0$.¹ Consistency with respect to strategies of \mathcal{P}_2 is defined analogously.

Outcome-equivalence of strategies. In the next sections, we study the expressiveness of finite-memory strategy models depending on the type of randomisation allowed. Two strategies may yield the same outcomes despite being different: the actions suggested by a strategy in an inconsistent history can be changed without affecting the outcome. Therefore, instead of using the equality of strategies as a measure of equivalence, we consider some weaker notion of equivalence, referred to as outcome-equivalence.

We say that two strategies σ_1 and τ_1 of \mathcal{P}_1 are *outcome-equivalent* if for any strategy σ_2 of \mathcal{P}_2 and for any initial state s_{init} , the probability distributions $\mathbb{P}_{s_{\text{init}}}^{\sigma_1, \sigma_2}$ and $\mathbb{P}_{s_{\text{init}}}^{\tau_1, \sigma_2}$ coincide. Outcome-equivalence of strategies can also be established without invoking induced probability distributions; two strategies are outcome-equivalent if and only if they coincide over the set of histories consistent with (one of) them. This can be shown by exploiting the definition of the probability of cylinder sets.

► **Lemma 1.** *Let σ_1 and τ_1 be two strategies of \mathcal{P}_1 . These two strategies are outcome-equivalent if and only if for all histories $h \in \text{Hist}_1(\mathcal{G})$, h consistent with σ_1 implies $\sigma_1(h) = \tau_1(h)$.*

Proof. Let us assume that σ_1 and τ_1 are outcome-equivalent. Let $h \in \text{Hist}_1(\mathcal{G})$ be a history that is consistent with σ_1 . Let s_{init} denote the first state of h and let σ_2 be a strategy of \mathcal{P}_2 consistent with h . It follows from the definition of the probability distribution of cylinders that for any $a \in A(\text{last}(h))$ and any $s \in \text{supp}(\delta(\text{last}(h), a))$, we have

¹ We use the terminology of consistency not only for plays and histories, but also for prefixes of plays that end with an action.

$$\sigma_1(h)(a) = \frac{\mathbb{P}_{s_{\text{init}}}^{\sigma_1, \sigma_2}(\text{Cyl}(has))}{\mathbb{P}_{s_{\text{init}}}^{\sigma_1, \sigma_2}(\text{Cyl}(h)) \cdot \delta(\text{last}(h), a)(s)} = \frac{\mathbb{P}_{s_{\text{init}}}^{\tau_1, \sigma_2}(\text{Cyl}(has))}{\mathbb{P}_{s_{\text{init}}}^{\tau_1, \sigma_2}(\text{Cyl}(h)) \cdot \delta(\text{last}(h), a)(s)} = \tau_1(h)(a),$$

which shows that $\sigma_1(h) = \tau_1(h)$. This ends the proof of the first direction.

Let us now assume that σ_1 and τ_1 coincide over histories consistent with σ_1 . Let σ_2 be a strategy of \mathcal{P}_2 and $s_{\text{init}} \in S$ be an initial state. It suffices to study the probability of cylinder sets. Let $h \in \text{Hist}(\mathcal{G})$ be a history starting in s_{init} . If h is consistent with σ_1 , then all prefixes of h also are, therefore the definition of the probability of a cylinder ensures that $\mathbb{P}_{s_{\text{init}}}^{\sigma_1, \sigma_2}(h) = \mathbb{P}_{s_{\text{init}}}^{\tau_1, \sigma_2}(h)$. Otherwise, if h is not consistent with σ_1 , then h is necessarily of the form $h'ah''$ with h' consistent with σ_1 and $\sigma_1(h')(a) = 0$. It follows that $\tau_1(h')(a) = 0$, thus $\mathbb{P}_{s_{\text{init}}}^{\sigma_1, \sigma_2}(h) = \mathbb{P}_{s_{\text{init}}}^{\tau_1, \sigma_2}(h) = 0$. This shows that σ_1 and σ_2 are outcome-equivalent, ending the proof. \blacktriangleleft

Subclasses of strategies. A strategy is called *pure* if it does not use randomisation; a pure strategy can be viewed as a function $\text{Hist}_i(\mathcal{G}) \rightarrow A$. A strategy that only uses information on the current state of the play is called *memoryless*: a strategy σ_i of \mathcal{P}_i is memoryless if for all histories $h, h' \in \text{Hist}_i(\mathcal{G})$, $\text{last}(h) = \text{last}(h')$ implies $\sigma_i(h) = \sigma_i(h')$. Memoryless strategies can be viewed as functions $S_i \rightarrow \mathcal{D}(A)$. Strategies that are both memoryless and pure can be viewed as functions $S_i \rightarrow A$.

A strategy σ is said to be *finite-memory* (FM) if it can be encoded by a Mealy machine, i.e., an automaton with outputs along its edges. We can include randomisation in the initialisation, outputs and updates (i.e., transitions) of the Mealy machine. Formally, a *stochastic Mealy machine* of \mathcal{P}_i is a tuple $\mathcal{M} = (M, \mu_{\text{init}}, \alpha_{\text{next}}, \alpha_{\text{up}})$, where M is a finite set of memory states, $\mu_{\text{init}} \in \mathcal{D}(M)$ is an initial distribution, $\alpha_{\text{next}}: M \times S_i \rightarrow \mathcal{D}(A)$ is the (stochastic) next-move function and $\alpha_{\text{up}}: M \times S \times A \rightarrow \mathcal{D}(M)$ is the (stochastic) update function.

Before we explain how to define the strategy induced by a Mealy machine, let us first describe how these machines work. Fix a Mealy machine $\mathcal{M} = (M, \mu_{\text{init}}, \alpha_{\text{next}}, \alpha_{\text{up}})$. Let $s_0 \in S$. At the start of a play, an initial memory state m_0 is selected randomly following μ_{init} . Then, at each step of the play such that $s_k \in S_i$, an action a_k is chosen following the distribution $\alpha_{\text{next}}(m_k, s_k)$, and otherwise an action is chosen following the other player's strategy. The memory state m_{k+1} is then randomly selected following the distribution $\alpha_{\text{up}}(m_k, s_k, a_k)$ and the game state s_{k+1} is chosen following the distribution $\delta(s_k, a_k)$, both choices being made independently.

Let us now explain how a strategy can be derived from a Mealy machine. As explained previously, when in a certain memory state $m \in M$ and game state $s \in S_i$, the probability of an action $a \in A(s)$ being chosen is given by $\alpha_{\text{next}}(m, s)(a)$. Therefore, the probability of choosing the action $a \in A$ after some history $h = ws$ (where $w \in (SA)^*$ and $s = \text{last}(h)$) is given by the sum, for each memory state $m \in M$, of the probability that m was reached after \mathcal{M} processes w , multiplied by $\alpha_{\text{next}}(m, s)(a)$.

To provide a formal definition of the strategy induced by \mathcal{M} , we must first describe the distribution over memory states after \mathcal{M} processes elements of $(SA)^*$. We formally define this distribution inductively. Details on how to derive the formulae for the update of these distributions, which use conditional probabilities, are presented in the full paper [31].

The distribution μ_ε over memory states after reading the empty word ε is by definition μ_{init} . Assume inductively we know the distribution μ_w for $w = s_0 a_0 \dots s_{k-1} a_{k-1}$ and let us explain how one derives $\mu_{ws_k a_k}$ from μ_w for any state $s_k \in \text{supp}(\delta(s_{k-1}, a_{k-1}))$ and for any action $a_k \in A(s_k)$.

If $s_k \in S_{3-i}$, i.e., s_k is not controlled by the owner of the strategy, the action a_k does not introduce any conditions on the current memory state. Therefore, we set, for any memory state $m \in M$,

$$\mu_{ws_k a_k}(m) = \sum_{m' \in M} \mu_w(m') \cdot \alpha_{\text{up}}(m', s_k, a_k)(m),$$

which consists in checking for each predecessor state m' , what the probability of moving to memory state m is and weighing the sum by the probability of being in m' .

If $s_k \in S_i$, i.e., s_k is controlled by the owner of the strategy, then the choice of an action conditions what the predecessor memory states could be. If we have, for all memory states $m' \in M$ such that $\mu_w(m') > 0$, that $\alpha_{\text{next}}(m', s_k)(a_k) = 0$, then the action a_k is actually never chosen. In this case, to ensure a complete definition, we perform an update as in the previous case. Otherwise, we condition updates on the likelihood of being in a memory state knowing that the action a_k was chosen. We define, for any memory state $m \in M$,

$$\mu_{ws_k a_k}(m) = \frac{\sum_{m' \in M} \mu_w(m') \cdot \alpha_{\text{up}}(m', s_k, a_k)(m) \cdot \alpha_{\text{next}}(m', s_k)(a_k)}{\sum_{m' \in M} \mu_w(m') \cdot \alpha_{\text{next}}(m', s_k)(a_k)}.$$

This quotient is not well-defined whenever for all $m' \in \text{supp}(\mu_w)$, $\alpha_{\text{next}}(m', s_k)(a_k) = 0$, justifying the distinction above.

Using these distributions, we formally define the strategy $\sigma_i^{\mathcal{M}}$ induced by the Mealy machine $\mathcal{M} = (M, \mu_{\text{init}}, \alpha_{\text{next}}, \alpha_{\text{up}})$ as the strategy $\sigma_i^{\mathcal{M}}: \text{Hist}_i(\mathcal{G}) \rightarrow \mathcal{D}(A)$ such that for all histories $h = ws$, for all actions $a \in A(s)$, $\sigma_i^{\mathcal{M}}(h)(a) = \sum_{m \in M} \mu_w(m) \cdot \alpha_{\text{next}}(m, s)(a)$.

Classifying finite-memory strategies. In the sequel, we investigate the relationships between different classes of finite-memory strategies with respect to expressive power. We classify finite-memory strategies following the type of stochastic Mealy machines that can induce them. We introduce a concise notation for each class: we use three-letter acronyms of the form XXX with $X \in \{D, R\}$, where the letters, in order, refer to the initialisation, outputs and updates of the Mealy machines, with D and R respectively denoting deterministic and randomised components. For instance, we will write RRD to denote the class of Mealy machines that have randomised initialisation and outputs, but deterministic updates. We also apply this terminology to FM strategies: we will say that an FM strategy is in the class XXX – i.e., it is an XXX strategy – if it is induced by an XXX Mealy machine.

Moreover, in the remainder of the paper, we will abusively identify Mealy machines and their induced FM strategies. For instance, we will say that \mathcal{M} is an XXX strategy to mean that \mathcal{M} is an XXX Mealy machine (thus inducing an XXX strategy). As a by-product of this identification, we apply the terminology introduced previously for strategies to Mealy machines, without explicitly referring to the strategy they induce. For instance, we may say a history is consistent with some Mealy machine, or that two Mealy machines are outcome-equivalent. Let us note however that we will not use a Mealy machine in lieu of its induced strategy whenever we are interested in the strategy itself as a function. This choice lightens notations; the strategy induced by a Mealy machine need not be introduced unless it is required as a function.

We close this section by commenting on some of the classes, and discuss previous appearances in the literature, under different names. Pure strategies use no randomisation: hence, the class DDD corresponds to pure FM strategies, which can be represented by Mealy machines that do not rely on randomisation.

Strategies in the class DRD have been referred to as *behavioural* FM strategies in [21]. The name comes from the randomised outputs, reminiscent of behavioural strategies that output a distribution over actions after a history. We note that stochastic Mealy machines that induce DRD strategies are such that their distributions over memory states are Dirac due to the deterministic initialisation and updates.

Similarly, RDD strategies have been referred to as *mixed* FM strategies [21]. The general definition of a mixed strategy is a distribution over pure strategies: under a mixed strategy, a player randomly selects a pure strategy at the start of a play and plays according to it for the whole play. RDD strategies are similar in the way that the random initialisation can be viewed as randomly selecting some DDD strategy (i.e., a pure FM strategy) among a *finite* selection of such strategies.

The elements of RRR, the broadest class of FM strategies, have been referred to as general FM strategies [21] and stochastic-update FM strategies [8, 18]. The latter name highlights the random nature of updates and insists on the difference with models that rely on deterministic updates, more common in the literature.

3 Taxonomy of finite-memory strategies

In this section, we comment on the relationships between the classes of finite-memory strategies in terms of expressiveness. We say that a class \mathcal{C}_1 of FM strategies is no less expressive than a class \mathcal{C}_2 if for all games \mathcal{G} , for all FM strategies $\mathcal{M} \in \mathcal{C}_2$ in \mathcal{G} , one can find some FM strategy $\mathcal{M}' \in \mathcal{C}_1$ of \mathcal{G} such that \mathcal{M} and \mathcal{M}' are outcome-equivalent strategies. For the sake of brevity, we will say that \mathcal{C}_2 is included in \mathcal{C}_1 , and write $\mathcal{C}_2 \subseteq \mathcal{C}_1$.

Figure 1.1 summarises our results. In terms of set inclusion, each line in the figure indicates that the class below is strictly included in the class above. Each line is decorated with a reference to the relevant results. The strictness results hold in one-player deterministic games. In particular, there are no collapses in the diagram in either two-player deterministic games and in Markov decision processes.

Some inclusions follow purely from syntactic arguments. For instance, the inclusion $\text{DRD} \subseteq \text{RRD}$ follows from the fact that RRD Mealy machines have more randomisation power than DRD ones. The inclusions $\text{RDD} \subseteq \text{DRD}$, $\text{RRR} \subseteq \text{DRR}$ and $\text{RRR} \subseteq \text{RDR}$, which do not follow from such arguments, are covered in Section 4.

Pure strategies are strictly less expressive than any other class of FM strategies; pure strategies cannot induce any non-Dirac distributions on plays in deterministic one-player games. The strictness of the other inclusions is presented in Section 5.

We close this section by comparing our results with Kuhn's theorem, which asserts that the classes of behavioural strategies and mixed strategies in games of perfect recall share the same expressiveness. Games of perfect recall have two traits: players never forget the sequence of histories controlled by them that have taken place and they can see their own actions. In particular, stochastic games of perfect information are a special case of games of perfect recall. Recall that mixed strategies are distributions over pure strategies. We comment briefly on the techniques used in the proof of Kuhn's theorem, and compare them with the finite-memory setting. Let us fix a game $\mathcal{G} = (S_1, S_2, A, \delta)$.

On the one hand, the emulation of mixed strategies with behavioural strategies is performed as follows. Let p_i be a mixed strategy of \mathcal{P}_i , i.e., a distribution over pure strategies of \mathcal{G} . An outcome-equivalent behavioural strategy σ_i is constructed such that, for all histories $h \in \text{Hist}_i(\mathcal{G})$ and actions $a \in A(\text{last}(h))$,

$$\sigma_i(h)(a) = \frac{p_i(\{\tau_i \text{ pure strategy} \mid \tau_i \text{ consistent with } h \text{ and } \tau_i(h) = a\})}{p_i(\{\tau_i \text{ pure strategy} \mid \tau_i \text{ consistent with } h\})}.$$

In the finite-memory case, similar ideas can be used to show that $\text{RDD} \subseteq \text{DRD}$. In the proof of Theorem 2, from some RDD strategy (i.e., a so-called mixed FM strategy), we construct a DRD strategy (i.e., a so-called behavioural FM strategy) that keeps track of the finitely many pure FM strategies that the RDD strategy mixes and that are consistent with the current history. An adaption of the quotient above is used in the next-move function of the DRD strategy.

On the other hand, the emulation of behavioural strategies by mixed strategies exploits the fact that mixed strategies may randomise over *infinite* sets. In a finite-memory setting, the same techniques cannot be applied. As a consequence, the class of RDD strategies is strictly included in the class of DRD strategies. In a certain sense, one could say that Kuhn's theorem only partially holds in the case of FM strategies.

4 Non-trivial inclusions

This section covers the non-trivial inclusions that are asserted in the lattice of Figure 1.1. The structure of this section is as follows. Section 4.1 covers the inclusion $\text{RDD} \subseteq \text{DRD}$. The inclusion $\text{RRR} \subseteq \text{DRR}$ is presented in Section 4.2. Finally, we close this section with the inclusion $\text{RRR} \subseteq \text{RDR}$ in Section 4.3. Full proofs and details of this section are presented in the full paper [31].

4.1 Simulating RDD strategies with DRD ones

We argue that for all RDD strategies in any game, one can find some outcome-equivalent DRD strategy (Theorem 2). The converse inclusion is not true; this discussion is relegated to Section 5.1. The outlined construction yields a DRD strategy that has a state space of size exponential in the size of the state space of the original RDD strategy; we complement Theorem 2 by proving that there are some RDD strategies for which this exponential blow-up in the number of states is necessary for any outcome-equivalent DRD strategy (Lemma 3). We argue that this blow-up is unavoidable in both deterministic two-player games and Markov decision processes.

Let $\mathcal{G} = (S_1, S_2, A, \delta)$ be a game. Fix an RDD strategy $\mathcal{M} = (M, \mu_{\text{init}}, \alpha_{\text{next}}, \alpha_{\text{up}})$ of \mathcal{P}_i . Let us sketch how to emulate \mathcal{M} with a DRD strategy $\mathcal{B} = (B, b_{\text{init}}, \beta_{\text{next}}, \beta_{\text{up}})$ built with a subset construction-like approach. The memory states of \mathcal{B} are functions $f: \text{supp}(\mu_{\text{init}}) \rightarrow M \cup \{\perp\}$. A memory state f is interpreted as follows. For all initial memory states $m_0 \in \text{supp}(\mu_{\text{init}})$, we have $f(m_0) = \perp$ if the history seen up to now is not consistent with the pure FM strategy $(M, m_0, \alpha_{\text{next}}, \alpha_{\text{up}})$, and otherwise $f(m_0)$ is the memory state reached in the same pure FM strategy after processing the current history. Updates are naturally derived from these semantics.

Using this state space and update scheme, we can compute the likelihood of each memory state of the mixed FM strategy \mathcal{M} after some sequence $w \in (SA)^*$ has taken place. Indeed, we keep track of each initial memory state from which it was possible to be consistent with w , and, for each such initial memory state m_0 , the memory state reached after w was processed starting in m_0 . Therefore, this likelihood can be inferred from μ_{init} ; the probability of \mathcal{M} being in $m \in M$ after w has been processed is given by the (normalised) sum of the probability of each initial memory state $m_0 \in \text{supp}(\mu_{\text{init}})$ such that $f(m_0) = m$.

The definition of the next-move function of \mathcal{B} is directly based on the distribution over states of \mathcal{M} described in the previous paragraph, and ensures that the two strategies select actions with the same probabilities at any given state. For any action $a \in A(s)$, the probability

of a being chosen in game state s and in memory state f is determined by the probability of \mathcal{M} being in some memory state m such that $\alpha_{\text{next}}(m, s) = a$, where this probability is inferred from f . It follows from Lemma 1 that \mathcal{M} and \mathcal{B} are outcome-equivalent.

Intuitively, we postpone the initial randomisation and instead randomise at each step in an attempt of replicating the initial distribution in the long run.

► **Theorem 2.** *Let $\mathcal{G} = (S_1, S_2, A, \delta)$ be a game. Let $\mathcal{M} = (M, \mu_{\text{init}}, \alpha_{\text{next}}, \alpha_{\text{up}})$ be an RDD strategy of \mathcal{P}_i . There exists a DRD strategy $\mathcal{B} = (B, b_{\text{init}}, \beta_{\text{next}}, \beta_{\text{up}})$ such that \mathcal{B} and \mathcal{M} are outcome-equivalent.*

The DRD strategy outlined prior to Theorem 2 leads to an exponential blow-up of the memory state space. For an RDD strategy $\mathcal{M} = (M, \mu_{\text{init}}, \alpha_{\text{next}}, \alpha_{\text{up}})$, we have described an outcome-equivalent DRD strategy with a state space consisting of functions $\text{supp}(\mu_{\text{init}}) \rightarrow M \cup \{\perp\}$, therefore with a state space of size $(|M| + 1)^{|\text{supp}(\mu_{\text{init}})|}$.

An exponential blow-up in the number of initial memory states cannot be avoided in general. Intuitively, due to the other player's actions (or stochastic transitions in Markov decision processes), it may be the case that for each subset of the initial states, there are histories ending in some fixed state s that are consistent with the pure FM strategies starting in these initial states. If these pure strategies each prescribe a different action in s , then there must be at least one memory state per non-empty subset of initial states in an outcome-equivalent DRD Mealy machine; this can be deduced by counting the number of necessary next-move functions $\alpha_{\text{next}}(\cdot, s)$. We obtain the following result.

► **Lemma 3.** *For all $n \in \mathbb{N}$, there exists a two-player deterministic game (respectively a Markov decision process) \mathcal{G}_n with $n + 2$ states, $4n + 2$ transitions, $n + 1$ actions, and an RDD strategy \mathcal{M}_n of \mathcal{P}_1 with n states such that any outcome-equivalent DRD strategy must have at least $2^n - 1$ states.*

4.2 Simulating RRR strategies with DRR ones

We establish that DRR strategies are as expressive as RRR strategies, i.e., randomness in the initialisation can be removed. We outline the ideas behind the construction of a DRR strategy that is outcome-equivalent to a given RRR strategy. The rough idea behind the construction is to simulate the behaviour of the RRR strategy at the start of the play using a new initial memory state and then move back into the RRR strategy we simulate.

We substitute the random selection of an initial memory element in two stages. To ensure the first action is selected in the same way under both the supplied strategy and the strategy we construct, we rely on the randomised outputs. The probability of selecting an action a in a given state s of the game in our new initial memory state is given as the sum of selecting action a in state s in each memory state m weighed by the initial probability of m .

We then leverage the stochastic updates to behave as though we had been using the supplied FM strategy from the start. If the first game state was controlled by the player who does not own the strategy, the probability of moving into a memory state m is also described by a weighted sum similar in spirit to the case of the first action (albeit by considering the update function in place of the next-move function). Whenever the owner of the strategy controls the first state of the game, the chosen action conditions which possible initial memory states we could have found ourselves in. The reasoning in this case is similar to the one for the update of the distribution over memory states (denoted by μ_w in Section 2) after processing some sequence in $(SA)^*$. In light of the above and Lemma 1, we obtain the following expressiveness result.

► **Theorem 4.** *Let $\mathcal{G} = (S_1, S_2, A, \delta)$ be a game. Let $\mathcal{M} = (M, \mu_{\text{init}}, \alpha_{\text{next}}, \alpha_{\text{up}})$ be an RRR strategy owned by \mathcal{P}_i . There exists a DRR strategy $\mathcal{B} = (B, b_{\text{init}}, \beta_{\text{next}}, \beta_{\text{up}})$ such that \mathcal{B} and \mathcal{M} are outcome-equivalent, and such that $|B| = |M| + 1$.*

4.3 Simulating RRR strategies with RDR ones

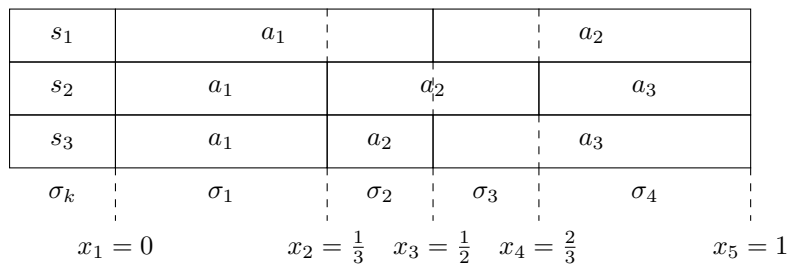
We are concerned with the simulation of RRR strategies by RDR strategies, i.e., with substituting randomised outputs with deterministic outputs. The idea behind the removal of randomisation in outputs is to simulate said randomisation by means of both stochastic initialisation and updates. These are used to preemptively perform the random selection of an action, simultaneously with the selection of an initial or successor memory state.

Let $\mathcal{G} = (S_1, S_2, A, \delta)$ be a stochastic game and let $\mathcal{M} = (M, \mu_{\text{init}}, \alpha_{\text{next}}, \alpha_{\text{up}})$ be an RRR strategy of \mathcal{P}_i . We construct an RDR strategy $\mathcal{B} = (B, \beta_{\text{init}}, \beta_{\text{next}}, \beta_{\text{up}})$ that is outcome-equivalent to \mathcal{M} and such that $|B| \leq |M| \cdot |S| \cdot |A|$. The state space of \mathcal{B} consists of pairs (m, σ_i) where $m \in M$ and σ_i is a pure memoryless strategy of \mathcal{P}_i . To achieve our bound on the size of B , we cannot take all pure memoryless strategies of \mathcal{P}_i . To illustrate how we perform the selection of these pure memoryless strategies, we provide a simple example of the construction on a DRD strategy (which is a special case of RRR strategies) with a single memory state (i.e., a memoryless randomised strategy).

► **Example 5.** We consider a game $\mathcal{G} = (S_1, S_2, A, \delta)$ where $S_1 = \{s_1, s_2, s_3\}$, $S_2 = \emptyset$, $A = \{a_1, a_2, a_3\}$ and all actions are enabled in all states. We need not specify δ exactly for our purposes. For our construction, we fix an order on the actions of \mathcal{G} : $a_1 < a_2 < a_3$.

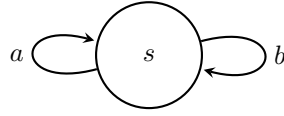
Let $\mathcal{M} = (\{m\}, m, \alpha_{\text{next}}, \alpha_{\text{up}})$ be the DRD strategy such that $\alpha_{\text{next}}(m, s_1)$ and $\alpha_{\text{next}}(m, s_2)$ are uniform distributions over $\{a_1, a_2\}$ and A respectively and $\alpha_{\text{next}}(m, s_3)$ is defined by $\alpha_{\text{next}}(m, s_3)(a_1) = \frac{1}{3}$, $\alpha_{\text{next}}(m, s_3)(a_2) = \frac{1}{6}$ and $\alpha_{\text{next}}(m, s_3)(a_3) = \frac{1}{2}$.

Figure 4.1 illustrates the probability of each action being chosen in each state as the length of a segment. Let us write $0 = x_1 < x_2 < x_3 < x_4 < x_5 = 1$ for all of the endpoints of the segments appearing in the illustration. For each index $k \in \{1, \dots, 4\}$, we define a pure memoryless strategy σ_k that assigns to each state the action lying in the segment above it in the figure. For instance, σ_2 is such that $\sigma_2(s_1) = a_1$ and $\sigma_2(s_2) = \sigma_2(s_3) = a_2$. Furthermore, for all $k \in \{1, \dots, 4\}$, the length $x_{k+1} - x_k$ of its corresponding interval denotes the probability of the strategy being chosen during stochastic updates.



■ **Figure 4.1** Representation of cumulative probability of actions under strategy \mathcal{M} and derived memoryless strategies.

We construct an RDR strategy $\mathcal{B} = (B, \beta_{\text{init}}, \beta_{\text{next}}, \beta_{\text{up}})$ that is outcome-equivalent to \mathcal{M} in the following way. We let $B = \{m\} \times \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$. The initial distribution is given by $\beta_{\text{init}}(m, \sigma_k) = x_{k+1} - x_k$, i.e., the probability of σ_k in the illustration. We set, for any $j, k \in \{1, \dots, 4\}$, $s \in S$ and $a \in A$, $\beta_{\text{up}}((m, \sigma_k), s, a)((m, \sigma_j)) = x_{j+1} - x_j$. Finally, we let $\beta_{\text{next}}((m, \sigma_k), s) = \sigma_k(s)$ for all $k \in \{1, \dots, 4\}$ and $s \in S$.



■ **Figure 5.1** A (one-player) game with a single state and two actions.

The argument for the outcome-equivalence of \mathcal{B} and \mathcal{M} is the following; for any state $s \in S_1$, the probability of moving into a memory state (m, σ_k) such that $\sigma_k(s) = a$ is by construction the probability $\alpha_{\text{next}}(m, s)$. ◀

In the previous example, we had a unique memory state m and we defined some memoryless strategies from the next-move function partially evaluated in this state (i.e., from $\alpha_{\text{next}}(m, \cdot)$). In general, each memory state may have a different partially evaluated next-move function, and therefore we must define some memoryless strategies for each individual memory state. For each memory state, we can bound the number of derived memoryless strategies by $|S_i| \cdot |A|$; we look at cumulative probabilities over actions (of which there are at most $|A|$) for each state of \mathcal{P}_i . This explains our announced bound on $|B|$.

Furthermore, in general, the memory update function is not trivial. Generalising the construction above can be done in a straightforward manner to handle updates. Intuitively, the probability to move to some memory state of the form (m, σ) is defined by the probability of moving into m multiplied by the probability of σ (in the sense of Figure 4.1).

We now formally state our result in the general setting.

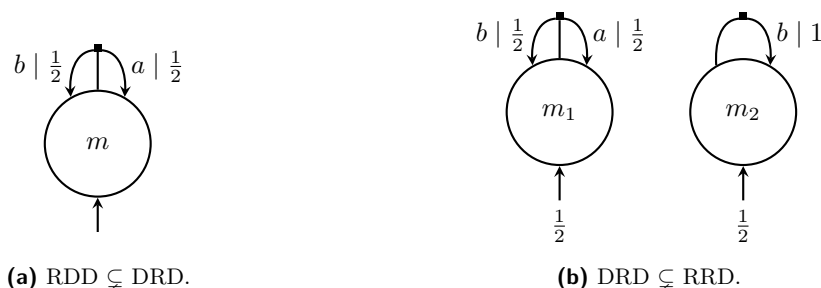
► **Theorem 6.** *Let $\mathcal{G} = (S_1, S_2, A, \delta)$ be a game. Let $\mathcal{M} = (M, \mu_{\text{init}}, \alpha_{\text{next}}, \alpha_{\text{up}})$ be an RRR strategy owned by \mathcal{P}_i . There exists an RDR strategy $\mathcal{B} = (B, \beta_{\text{init}}, \beta_{\text{next}}, \beta_{\text{up}})$ such that \mathcal{B} and \mathcal{M} are outcome-equivalent, and such that $|B| \leq |M| \cdot |S_i| \cdot |A|$.*

5 Strictness of inclusions

We now discuss the strictness of inclusions in the lattice of Figure 1.1. Section 5.1 complements the previous Section 4.1 and presents a DRD strategy that has no outcome-equivalent RDD counterpart. The strict inclusion of the class DRD in the class of RRD strategies is covered in Section 5.2. Finally, we provide the necessary results to establish that the class DDR is incomparable to the classes of RDD, DRD and RRD strategies in Section 5.3. Technical details are presented in the full paper [31].

All strictness results hold in one-player deterministic games with a single state and two actions. This is one of the simplest possible settings to show that strategy classes are distinct. Indeed, in a game with a single state and a single action, the only strategy is to always play the unique action, and therefore all strategy classes collapse into one. For the entirety of this section, we let \mathcal{G} denote the game depicted in Figure 5.1, and only consider strategies of \mathcal{G} in the upcoming statements.

We illustrate FM strategies witnessing the strictness of inclusions asserted in the lattice of Figure 1.1 in Figures 5.2 and 5.3. The Mealy machines are interpreted as follows. Edges that exit memory states read a game state (omitted in these figures due to s being the sole involved game state) and split into edges labelled by an action and a probability of this action being played, e.g., for $c \in \{a, b\}$ and $p \in [0, 1]$, the notation $c | p$ indicates that the probability of playing action c in the current memory state is p . In Figure 5.3, the edges are further split after the choice of an action for randomised updates. The edge labels following this second split represent the probabilities of stochastic updates. This second split is omitted whenever an update is deterministic.



■ **Figure 5.2** Depictions of Mealy machines witnessing the strictness of two inclusions asserted in Figure 1.1. For the sake of readability, we do not label transitions by s as it is the sole state the Mealy machines can read in \mathcal{G} .

5.1 RDD strategies are strictly less expressive than DRD ones

We argue that there exists a DRD strategy that cannot be emulated by any RDD strategy. Let us first explain some intuition behind this statement. Intuitively, an RDD strategy can only randomise once at the start between a finite number of pure FM (DDD) strategies. After this initial randomisation, the sequence of actions prescribed by the RDD strategy is fixed relative to the play in progress. Any DRD strategy that chooses an action randomly at each step, such as the strategy depicted in Figure 5.2a, i.e., the strategy playing actions a and b with uniform probability at each step in \mathcal{G} , cannot be reproduced by an RDD strategy. Indeed, this randomisation generates an infinite number of patterns of actions. These patterns cannot all be captured by an RDD strategy due to the fact that its initial randomisation is over a finite set.

► **Lemma 7.** *There exists a DRD strategy of \mathcal{P}_1 such that there is no outcome-equivalent RDD strategy.*

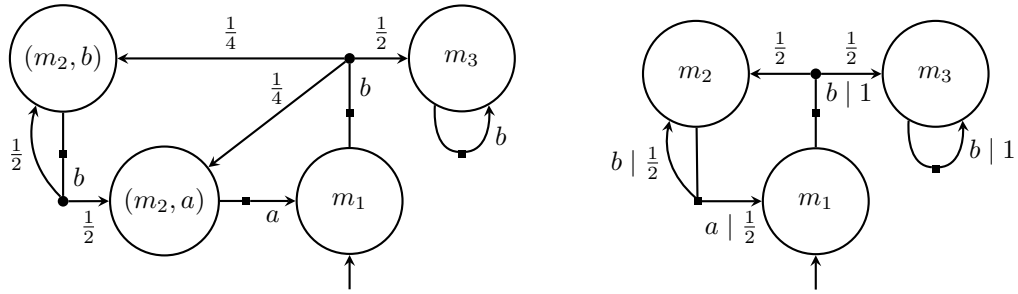
5.2 DRD strategies are strictly less expressive than RRD ones

We argue that there exists an RRD strategy that has no outcome-equivalent DRD strategy. The example we provide is based on results of [21, 22]; the authors of [21] illustrate that some behaviour proven to not be achievable by DRD strategies in concurrent reachability games by [22] can be achieved using RRD strategies.

Consider the Mealy machine depicted in Figure 5.2b. The main idea underlying its induced strategy is the following. This strategy attempts the action a at all steps with a positive probability due to memory state m_1 . It also has a positive probability of never playing a due to memory state m_2 .

This behaviour cannot be achieved with a DRD strategy. The distribution over memory states of a DRD strategy following a history is a Dirac distribution due to the deterministic initialisation and deterministic updates. It follows that DRD strategies suggest actions with probabilities given directly by the next-move function. In particular, if an action is attempted at each round by a DRD strategy, then there exists a positive lower bound on the probability of the action being chosen (as there are finitely many memory states), therefore the action is eventually selected almost-surely. It follows that there is no DRD strategy that is outcome-equivalent to the strategy depicted in Figure 5.2b.

► **Lemma 8.** *There exists an RRD strategy of \mathcal{P}_1 such that there is no outcome-equivalent DRD strategy.*



(a) A DDR strategy witnessing $\text{DDR} \not\subseteq \text{RRD}$. (b) An outcome-equivalent RRR strategy with fewer states.

■ **Figure 5.3** Outcome-equivalent strategies witnessing the non-inclusion $\text{DDR} \not\subseteq \text{RRD}$. For the sake of readability, we do not label transitions by s as it is the sole state the Mealy machines can read in \mathcal{G} . We omit the probability of actions in Figure 5.3a as outputs are deterministic.

5.3 RRD and DDR strategies are incomparable

We argue that the classes RRD and DDR of finite-memory strategies are incomparable. While we have shown that RDR and DRR strategies are as powerful as RRR strategies, DDR strategies are not because they lack the ability to provide a random output at the first step of a game. Due to this trait, one can even construct some RDD strategy that cannot be emulated by any DDR strategy; any strategy that randomises between two pure FM strategies prescribing action a and action b respectively at the first turn in \mathcal{G} has no outcome-equivalent DDR strategy. The following result follows immediately.

► **Lemma 9.** *There exists an RDD strategy of \mathcal{P}_1 such that there is no outcome-equivalent DDR strategy.*

On the other hand, one can construct a DDR strategy that has no outcome-equivalent RRD strategy. For instance, the DDR strategy that is depicted in Figure 5.3a has no outcome-equivalent RRD strategy. For ease of analysis, we illustrate in Figure 5.3b a DRR strategy with fewer states that is outcome-equivalent to the Mealy machine depicted in Figure 5.3a. Note that the DDR strategy of Figure 5.3a can be obtained by applying the construction of Theorem 6 to Figure 5.3b.

Intuitively, these strategies have a non-zero probability of never using action a after any history, while they have a positive probability of using action a at any time besides the first round and right after a use of the action a . The behaviour described above cannot be reproduced by an RRD strategy. There are two reasons to this.

First, along any play consistent with an RRD strategy, the support of the distribution over memory states cannot increase in size. Because of deterministic updates, the probability carried by a memory state m can only be transferred to at most one other state, and may be lost if the used action cannot be used while in m . This is not the case for strategies that have stochastic updates, such as those of Figure 5.3.

Second, one can force situations in which the size of the support of the distribution over memory states of an RRD strategy must decrease. If after a given history h , the action a has a positive probability of never being used despite being assigned a positive probability at each round after h , then at some point there must be some memory state of the RRD strategy that has positive probability and that assigns (via the next-move function) probability zero to action a . For instance, this is the case from the start with the RRD strategy depicted in

Figure 5.2b. Intuitively, in general, if at all times all memory states in the support of the distribution over memory states after the current history assign a positive probability to action a , the probability of using a at each round after h would be bounded from below by the smallest positive probability assigned to a by the next-move function. Therefore a would eventually be played almost-surely assuming h has taken place, contradicting the fact that there was a positive probability of never using action a after h . By using action a at a point in which some memory state in the support of the distribution over memory states assigns probability zero to a , the size of the support of the strategy decreases.

By design of our DDR strategy, if one assumes the existence of an outcome-equivalent RRD strategy, then it is possible to construct a play along which the size of the support of the distribution over memory states of the RRD strategy decreases between two consecutive steps infinitely often. Because this size cannot increase along a play, this is not possible, i.e., there is no such RRD strategy. We obtain the following lemma.

► **Lemma 10.** *There exists a DDR strategy of \mathcal{P}_1 such that there is no outcome-equivalent RRD strategy.*

6 Extensions

In the following, we discuss settings beyond two-player games with perfect information to which our results carry over.

Multi-player games. In this work, we have considered two-player games. Our result also extends to games with more than two players. The definition of outcome-equivalence naturally extends to multi-player games; instead of quantifying universally over strategies of the other player as is done in the two-player setting, one quantifies universally over strategy profiles of all other players when defining outcome-equivalence with more players.

In practice, when studying the outcome-equivalence of strategies of some given player \mathcal{P}_i , there being more than one other player is no different than having a fictitious single other player obtained as a coalition of all players besides \mathcal{P}_i . Therefore, all of our results carry over to the multi-player setting directly. Furthermore, due to outcome-equivalence being a criterion that depends solely on the studied player, as highlighted by the equivalent formulation given in Lemma 1, the way players are arranged in coalitions does not affect our taxonomy in multi-player games with coalitions.

Imperfect information. Our presentation assumes perfect information; the players are fully informed of the sequence of witnessed states and used actions. Our results also hold in a context of partial observation with observable actions. A formalisation of the following is provided in the full paper [31].

First, it is not necessary to have full knowledge of the states; an observation suffices. Intuitively, the Mealy machines are formally agnostic to the nature of the inputs, therefore states can be substituted with observations while preserving correctness of all of our constructions.

Second, we need not be capable to observe the actions of the other player for our constructions. However, it is required to observe the actions of the player we consider; we exploit this in the memory update functions in the constructions relevant to Theorems 2 and 4. In addition to the visibility of actions, these constructions need also the capability of distinguishing from a sequence of observations whose turn it was at each step; this can be achieved either by requiring that the two players have disjoint action sets or by encoding in the state observations to whom the states belong.

References

- 1 Robert J. Aumann. *28. Mixed and Behavior Strategies in Infinite Extensive Games*, pages 627–650. Princeton University Press, 2016. doi:10.1515/9781400882014-029.
- 2 Raphaël Berthon, Mickael Randour, and Jean-François Raskin. Threshold constraints with guarantees for parity objectives in Markov decision processes. 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 121:1–121:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.121.
- 3 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.
- 4 Patricia Bouyer, Stéphane Le Roux, Youssouf Oualhadj, Mickael Randour, and Pierre Vandenhove. Games where you can play optimally with arena-independent finite memory. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 24:1–24:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.24.
- 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, Youssouf Oualhadj, Mickael Randour, and Pierre Vandenhove. Arena-independent finite-memory determinacy in stochastic games. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPIcs*, pages 26:1–26:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.CONCUR.2021.26.
- 7 Patricia Bouyer, Mickael Randour, and Pierre Vandenhove. Characterizing omega-regularity through finite-memory determinacy of games on infinite graphs. In Petra Berenbrink and Benjamin Monmege, editors, *39th International Symposium on Theoretical Aspects of Computer Science, STACS 2022, March 15-18, 2022, Marseille, France (Virtual Conference)*, volume 219 of *LIPIcs*, pages 16:1–16:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.STACS.2022.16.
- 8 Tomáš Brázdil, Václav Brozek, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. Markov decision processes with multiple long-run average objectives. *Log. Methods Comput. Sci.*, 10(1), 2014. doi:10.2168/LMCS-10(1:13)2014.
- 9 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.
- 10 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.
- 11 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.

- 12 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.
- 13 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.
- 14 Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. Trading memory for randomness. In *1st International Conference on Quantitative Evaluation of Systems (QEST 2004), 27-30 September 2004, Enschede, The Netherlands*, pages 206–217. IEEE Computer Society, 2004. doi:10.1109/QEST.2004.1348035.
- 15 Krishnendu Chatterjee and Laurent Doyen. Partial-observation stochastic games: How to win when belief fails. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 175–184. IEEE Computer Society, 2012. doi:10.1109/LICS.2012.28.
- 16 Krishnendu Chatterjee, Laurent Doyen, Hugo Gimbert, and Thomas A. Henzinger. Randomness for free. *Inf. Comput.*, 245:3–16, 2015. doi:10.1016/j.ic.2015.06.003.
- 17 Krishnendu Chatterjee, Thomas A. Henzinger, and Vinayak S. Prabhu. Trading infinite memory for uniform randomness in timed games. In Magnus Egerstedt and Bud Mishra, editors, *Hybrid Systems: Computation and Control, 11th International Workshop, HSCC 2008, St. Louis, MO, USA, April 22-24, 2008. Proceedings*, volume 4981 of *Lecture Notes in Computer Science*, pages 87–100. Springer, 2008. doi:10.1007/978-3-540-78929-1_7.
- 18 Krishnendu Chatterjee, Zuzana Kretínská, and Jan Kretínský. Unifying two views on multiple mean-payoff objectives in Markov decision processes. *Log. Methods Comput. Sci.*, 13(2), 2017. doi:10.23638/LMCS-13(2:15)2017.
- 19 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.
- 20 Anne Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992. doi:10.1016/0890-5401(92)90048-K.
- 21 Julien Cristau, Claire David, and Florian Horn. How do we remember the past in randomised strategies? In Angelo Montanari, Margherita Napoli, and Mimmo Parente, editors, *Proceedings First Symposium on Games, Automata, Logic, and Formal Verification, GANDALF 2010, Minori (Amalfi Coast), Italy, 17-18th June 2010*, volume 25 of *EPTCS*, pages 30–39, 2010. doi:10.4204/EPTCS.25.7.
- 22 Luca de Alfaro, Thomas A. Henzinger, and Orna Kupferman. Concurrent reachability games. *Theor. Comput. Sci.*, 386(3):188–217, 2007. doi:10.1016/j.tcs.2007.07.008.
- 23 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.
- 24 Rick Durrett. *Probability: Theory and Examples*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 5th edition, 2019. doi:10.1017/9781108591034.
- 25 Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *Int. Journal of Game Theory*, 8(2):109–113, 1979.

- 26 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.
- 27 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.
- 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 Florian Horn. Random fruits on the Zielonka tree. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, volume 3 of *LIPICs*, pages 541–552. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2009. doi:10.4230/LIPICs.STACS.2009.1848.
- 30 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.
- 31 James C. A. Main and Mickael Randour. Different strokes in randomised strategies: Revisiting Kuhn's theorem under finite-memory assumptions. *CoRR*, abs/2201.10825, 2022. arXiv:2201.10825.
- 32 A. Maitra and W. Sudderth. Stochastic games with Borel payoffs. In Abraham Neyman and Sylvain Sorin, editors, *Stochastic Games and Applications*, pages 367–373. Dordrecht, 2003. Springer Netherlands.
- 33 Benjamin Monmege, Julie Parreaux, and Pierre-Alain Reynier. Reaching your goal optimally by playing at random with no memory. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPICs*, pages 26:1–26:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICs.CONCUR.2020.26.
- 34 Martin J. Osborne and Ariel Rubinstein. *A course in game theory*. The MIT Press, Cambridge, USA, 1994. electronic edition.
- 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 Mickael Randour, Jean-François Raskin, and Ocan Sankur. Percentile queries in multi-dimensional Markov decision processes. *Formal Methods Syst. Des.*, 50(2-3):207–248, 2017. doi:10.1007/s10703-016-0262-7.
- 37 L. S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953. doi:10.1073/pnas.39.10.1095.
- 38 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.
- 39 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.

Regular Model Checking Upside-Down: An Invariant-Based Approach

Javier Esparza  

Department of Informatics – I7, Technische Universität München, Germany

Mikhail Raskin  

Department of Informatics – I7, Technische Universität München, Germany

Christoph Welzel  

Department of Informatics – I7, Technische Universität München, Germany

Abstract

Regular model checking is a technique for the verification of infinite-state systems whose configurations can be represented as finite words over a suitable alphabet. It applies to systems whose set of initial configurations is regular, and whose transition relation is captured by a length-preserving transducer. To verify safety properties, regular model checking iteratively computes automata recognizing increasingly larger regular sets of reachable configurations, and checks if they contain unsafe configurations. Since this procedure often does not terminate, acceleration, abstraction, and widening techniques have been developed to compute a regular superset of the reachable configurations.

In this paper we develop a complementary procedure. Instead of approaching the set of reachable configurations from below, we start with the set of all configurations and approach it from above. We use that the set of reachable configurations is equal to the intersection of all inductive invariants of the system. Since this intersection is non-regular in general, we introduce b -bounded invariants, defined as those representable by CNF-formulas with at most b clauses. We prove that, for every $b \geq 0$, the intersection of all b -bounded inductive invariants is regular, and we construct an automaton recognizing it. We show that whether this automaton accepts some unsafe configuration is in EXPSpace for every $b \geq 0$, and PSPACE-complete for $b = 1$. Finally, we study how large must b be to prove safety properties of a number of benchmarks.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness; Theory of computation → Program analysis; Theory of computation → Invariants

Keywords and phrases parameterized verification, structural analysis, regular languages, regular model-checking, traps

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.23

Related Version *Full Version*: <https://arxiv.org/abs/2205.03060>

Supplementary Material *Software (Repository of examples)*: <https://doi.org/10.5281/zenodo.6483615>

Funding The 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).

1 Introduction

Regular model checking (RMC) is a framework for the verification of different classes of infinite-state systems (see, e.g., the surveys [5, 1, 6, 2]). In its canonical version, RMC is applied to systems satisfying the following conditions: configurations can be encoded as words, the set of initial configurations is recognized by a finite automaton \mathcal{A}_I , and the transition relation is recognized by a length-preserving transducer \mathcal{A}_T . RMC algorithms



© Javier Esparza, Mikhail Raskin, and Christoph Welzel;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 23; pp. 23:1–23:19

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

address the problem of, given a regular set of unsafe configurations, deciding if its intersection with the set of reachable configurations is empty or not. In the present paper we do not consider generalisations to non-length-preserving transitions.

The fundamental building block of current RMC algorithms is an automata-theoretic construction that, given a non-deterministic automaton (NFA) A recognizing a regular set of configurations, produces another NFA recognizing the set of immediate successors (or predecessors) of $\mathcal{L}(A)$ with respect to the transition relation represented by \mathcal{A}_T . Therefore, if some unsafe configuration is reachable, one can find a witness by, starting with the set of initial configurations, repeatedly adding the set of immediate successors. However, this approach never terminates when all reachable configurations are safe. Research on RMC has produced many acceleration, abstraction, and widening techniques to make the iterative computation “jump over the fixpoint” in finite time, and produce an invariant of the system not satisfied by any unsafe configuration (see, e.g., [10, 22, 15, 4, 7, 9, 11, 8, 23, 14]).

In this paper we develop a complementary approach that, starting with the set of all configurations, computes increasingly smaller regular inductive invariants, i.e., sets of configurations closed under the reachability relation and containing all initial configurations. Our main contribution is the definition of a sequence of *regular* inductive invariants that converges (in the limit) to the set of reachable configurations, and for which automata can be directly constructed from \mathcal{A}_I and \mathcal{A}_T .

Our starting point is the fact that the set of reachable configurations is equal to the intersection of all inductive invariants. Since this intersection is non-regular in general, we introduce *b-bounded* invariants. An invariant is *b-bounded* if, for every $\ell \geq 0$, the configurations of length ℓ satisfying the invariant are those satisfying a Boolean formula in conjunctive normal form with at most b clauses. For example, assume that the configurations of some system are words over the alphabet $\{a, b, c, d\}$, and that the configurations of length five where the second letter is an a or the fourth letter is a b , and the second letter is a b or the third is a c , constitute an inductive invariant. Then this set of configurations is a 2-bounded invariant, represented by the formula $(a_{2:5} \vee b_{4:5}) \wedge (b_{2:5} \vee d_{3:5})$. We prove that, for every bound $b \geq 0$, the intersection of all *b-bounded* inductive invariants, denoted $IndInv_b$, is regular, and recognized by a DFA of double exponential size in \mathcal{A}_I and \mathcal{A}_T . As a corollary, we obtain that, for every $b \geq 0$, deciding if $IndInv_b$ contains some unsafe configuration is in EXPSPACE.

In the second part of the paper, we study the special case $b = 1$ in more detail. We exploit that 1-bounded inductive invariants are closed under union (a special feature of the $b = 1$ case) to prove that deciding if $IndInv_1$ contains some unsafe configuration is PSPACE-complete. The proof also shows that $IndInv_b$ can be recognized by a NFA of single exponential size in \mathcal{A}_I and \mathcal{A}_T .

The index b of a bounded invariant can be seen as a measure of how difficult it is for a human to understand the invariant. So one is interested in the smallest b such that $IndInv_b$ is strong enough to prove a given property. In the third and final part of the paper, we experimentally show that for a large number of systems $IndInv_1$ is strong enough to prove useful safety properties.

Related work. The work closest to ours is [3], which directly computes an overapproximation of the set of reachable configurations of a parameterized system. Contrary to our approach, the paper computes one single approximation, instead of a converging sequence of overapproximations. Further, the method is designed for a model of parameterized systems with existential or universal guarded commands, while our technique can be applied to any

model analyzable by RMC. Our work is also related to [14], which computes an overapproximation using a learning approach, which terminates if the set of reachably configurations is regular; our paper shows that a natural class of invariants is regular, and that automata for them can be constructed explicitly from the syntactic description of the system. This paper generalizes the work of [12, 19, 13, 20] on trap invariants for parameterized Petri nets. Trap invariants are a special class of 1-bounded invariants, and the parameterized Petri nets studied in these papers can be modeled in the RMC framework. An alternative to regular model checking are logical based approaches. The invisible invariant method synthesizes candidate invariants from examples, which are then checked for inductiveness [26]. Our approach does not produce candidates, it generates invariants by construction. Modern tools like Ivy [25, 24] have verified more complex protocols than the ones in Section 5 using a combination of automation and human interaction. The best way of achieving this interaction is beyond the scope of this paper, which focuses on the foundations of regular model checking.

Structure of the paper. Section 2 introduces basic definitions of the RMC framework. Section 3 introduces b -bound invariants and proves regularity of $IndInv_b$. Section 4 proves the PSPACE-completeness result. Sections 5 and 6 contain some experimental results and conclusions.

Full version. All missing proofs can be found in the full version of this paper [21].

2 Preliminaries

Automata and transducers. Given $n, m \in \mathbb{N}$, we let $[n, m]$ denote the set $\{i \in \mathbb{N} : n \leq i \leq m\}$. Given a word w of length ℓ , we let $w[i]$ denote the i -th letter of w , i.e., $w = w[1] \cdots w[\ell]$.

A nondeterministic finite automaton (NFA) is a tuple $\mathcal{A} = \langle Q, \Sigma, \Delta, Q_0, F \rangle$, where Q is a non-empty finite set of states, Σ is an alphabet, $\Delta: Q \times \Sigma \rightarrow 2^Q$ is a transition function, and $Q_0, F \subseteq Q$ are sets of initial and final states, respectively. A run of \mathcal{A} on a word $w \in \Sigma^\ell$ is a sequence $q_0 q_1 \dots q_\ell$ of states such that $q_0 \in Q_0$ and $q_i \in \Delta(q_{i-1}, w[i])$ for every $i \in [1, \ell]$. A run on w is accepting if $q_\ell \in F$, and \mathcal{A} accepts w if there exists an accepting run of \mathcal{A} on w . The language recognized by \mathcal{A} , denoted $\mathcal{L}(\mathcal{A})$ or $\mathcal{L}_{\mathcal{A}}$, is the set of words accepted by \mathcal{A} . We let $|\mathcal{A}|$ denote the number of states of \mathcal{A} . A NFA \mathcal{A} is deterministic (DFA) if $|Q_0| = 1$ and $|\Delta(q, a)| = 1$ for every $q \in Q$ and $a \in \Sigma$.

The function $\delta_{\mathcal{A}}: 2^Q \times \Sigma^* \rightarrow 2^Q$ is defined inductively as follows: $\delta_{\mathcal{A}}(P, \varepsilon) = P$ and $\delta_{\mathcal{A}}(P, aw) = \delta_{\mathcal{A}}(\bigcup_{p \in P} \Delta(p, a), w)$. Observe that \mathcal{A} accepts w iff $\delta_{\mathcal{A}}(Q_0, w) \cap F \neq \emptyset$.

A (length-preserving) transducer over $\Sigma \times \Gamma$ is a NFA over an alphabet $\Sigma \times \Gamma$. We denote elements of $\Sigma \times \Gamma$ as $\langle a, b \rangle$ or $\begin{bmatrix} a \\ b \end{bmatrix}$, where $a \in \Sigma$ and $b \in \Gamma$. Given two words $w \in \Sigma^\ell, u \in \Gamma^\ell$ of the same length ℓ and a transducer \mathcal{A} , we say that \mathcal{A} accepts $\langle w, u \rangle$, or transduces w into u , if it accepts the word $\langle w[1], u[1] \rangle \cdots \langle w[\ell], u[\ell] \rangle \in (\Sigma \times \Gamma)^\ell$.

Regular model checking. Regular model checking (RMC) is a framework for the verification of systems with infinitely many configurations. Each configuration is represented as a finite word over a fixed alphabet Σ . Systems are modeled as regular transition systems (RTS).

► **Definition 1** (Regular transition systems). *A RTS is a triple $\mathcal{R} = \langle \Sigma, \mathcal{A}_I, \mathcal{A}_T \rangle$, where Σ is an alphabet, \mathcal{A}_I is a NFA over Σ , and \mathcal{A}_T is a transducer over $\Sigma \times \Sigma$.*

Words over Σ are called *configurations*. Configurations accepted by \mathcal{A}_I are called *initial*, and pairs of configurations accepted by \mathcal{A}_T are called *transitions*. We write $w \rightsquigarrow u$ to denote that $\langle w, u \rangle$ is a transition. Observe that $w \rightsquigarrow u$ implies $|w| = |u|$. Given two configurations w, u , we say that u is *reachable* from w if $w \rightsquigarrow^* u$, where \rightsquigarrow^* denotes the reflexive and transitive closure of \rightsquigarrow . The set of reachable configurations of \mathcal{R} , denoted $Reach(\mathcal{R})$, or just $Reach$ when there is no confusion, is the set of configurations reachable from the initial configurations. In the following, we use $|\mathcal{R}|$ to refer to $|\mathcal{A}_I| + |\mathcal{A}_T|$.

► **Example 2** (Dining philosophers). We model a very simple version of the dining philosophers as a RTS, for use as running example. Philosophers sit at a round table with forks between them. Philosophers can be *thinking* (t) or *eating* (e). Forks can be *free* (f) or *busy* (b). A thinking philosopher whose left and right forks are free can *simultaneously* grab both forks – the forks become busy – and start eating. After eating, the philosopher puts both forks to the table and returns to thinking. The model includes two corner cases: a table with one philosopher and one fork, which is then both the left and the right fork (unusable as it would need to be grabbed twice in a single transition), and the empty table with no philosophers or forks.

We model the system as a RTS over the alphabet $\Sigma = \{t, e, f, b\}$. A configuration of a table with n philosophers and n forks is represented as a word over Σ of length $2n$. Letters at odd and even positions model the current states of philosophers and forks (positions start at 1). For example, $tftf$ models a table with two thinking philosophers and two free forks. The language of initial configurations is $\mathcal{L}_I = (tf)^*$, and the language of transitions is

$$\mathcal{L}_T = \begin{bmatrix} t \\ e \end{bmatrix} \begin{bmatrix} f \\ b \end{bmatrix} \begin{bmatrix} x \\ x \end{bmatrix}^* \begin{bmatrix} f \\ b \end{bmatrix} \mid \begin{bmatrix} e \\ t \end{bmatrix} \begin{bmatrix} b \\ f \end{bmatrix} \begin{bmatrix} x \\ x \end{bmatrix}^* \begin{bmatrix} b \\ f \end{bmatrix} \mid \begin{bmatrix} x \\ x \end{bmatrix}^* \left(\begin{bmatrix} f \\ b \end{bmatrix} \begin{bmatrix} t \\ e \end{bmatrix} \begin{bmatrix} f \\ b \end{bmatrix} \mid \begin{bmatrix} b \\ f \end{bmatrix} \begin{bmatrix} e \\ t \end{bmatrix} \begin{bmatrix} b \\ f \end{bmatrix} \right) \begin{bmatrix} x \\ x \end{bmatrix}^*$$

where $\begin{bmatrix} x \\ x \end{bmatrix}$ stands for the regular expression $\left(\begin{bmatrix} t \\ t \end{bmatrix} \mid \begin{bmatrix} e \\ e \end{bmatrix} \mid \begin{bmatrix} f \\ f \end{bmatrix} \mid \begin{bmatrix} b \\ b \end{bmatrix} \right)$. The first two terms of \mathcal{L}_T describe the actions of the first philosopher, and the second the actions of the others. It is not difficult to show that $Reach = (t(f \mid beb))^* \mid ebt((f \mid beb)t)^*$. (These are the configurations where no two philosophers are using the same fork, and fork states match their adjacent philosopher states.)

Safety verification problem for RTSs. The safety verification problem for RTSs is defined as follows: Given a RTS \mathcal{R} and a NFA \mathcal{U} recognizing a set of *unsafe* configurations, decide whether $Reach(\mathcal{R}) \cap \mathcal{L}_{\mathcal{U}} = \emptyset$ holds. The problem is known to be undecidable.

3 Bounded inductive invariants of a RTS

We present an invariant-based approach to the safety verification problem for RTSs. Fix a RTS $\mathcal{R} = \langle \Sigma, \mathcal{A}_I, \mathcal{A}_T \rangle$. We introduce an infinite sequence

$$\Sigma^* = IndInv_0 \supseteq IndInv_1 \supseteq IndInv_2 \dots \supseteq Reach$$

of effectively regular inductive invariants of \mathcal{R} that converges to $Reach$, i.e., $IndInv_k$ is effectively regular for every $k \geq 1$, and $Reach = \bigcap_{k=0}^{\infty} IndInv_k$. Section 3.1 recalls basic notions about invariants, Section 3.2 defines the inductive invariant $IndInv_b$ for every $b \geq 0$, and Section 3.3 shows that $IndInv_b$ is regular.

3.1 Invariants

Let $S \subseteq \Sigma^*$ be a set of configurations. S is an *invariant for length ℓ* if $\text{Reach} \cap \Sigma^\ell \subseteq S \cap \Sigma^\ell$, and an *invariant* if $\text{Reach} \subseteq S$. Observe that, since \mathcal{A}_T is a length-preserving transducer, S is an invariant iff it is an invariant for every length. A set S (invariant or not) is *inductive* if it is closed under reachability, i.e. $w \in S$ and $w \rightsquigarrow u$ implies $u \in S$. Given two invariants I_1, I_2 , we say that I_1 is *stronger* than I_2 if $I_1 \subseteq I_2$. Observe that inductive invariants are closed under union and intersection, and so there exists a unique strongest inductive invariant of \mathcal{R} . Since Reach is an inductive set, the strongest inductive invariant is Reach .

► **Example 3.** The set $I_0 = ((t \mid e)(f \mid b))^*$ is an inductive invariant of the dining philosophers. Other inductive invariants are

$$I_1 = \overline{\Sigma^* e f \Sigma^*} \quad I_2 = \overline{\Sigma^* f e \Sigma^*} \quad I_3 = \overline{e \Sigma^* f} \quad I_4 = \overline{\Sigma^* t b t \Sigma^*} \quad I_5 = \overline{t \Sigma^* t b}$$

Taking into account that the table is round, these are the sets of configurations without any occurrence of ef (I_1), fe (I_2 and I_3), and tbt (I_4 and I_5).

3.2 Bounded invariants

Given a length $\ell \geq 0$, we represent certain sets of configurations as Boolean formulas over a set AP_ℓ of atomic propositions. More precisely, a Boolean formula over AP_ℓ describes a set containing *some* configurations of length ℓ , and *all* configurations of other lengths.

The set AP_ℓ contains an atomic proposition $q_{j:\ell}$ for every $q \in \Sigma$ and for $j \in [1, \ell]$. A *formula* φ over AP_ℓ is a positive Boolean combination of atomic propositions of AP_ℓ and the constants *true* and *false*. Formulas are interpreted on configurations. Intuitively, an atomic proposition $q_{j:\ell}$ states that either the configuration does *not* have length ℓ , or it has length ℓ and its j -th letter is q . Formally, $w \in \Sigma^*$ *satisfies* φ , denoted $w \models \varphi$, if $\varphi = q_{j:\ell}$ and $|w| \neq \ell$ or $|w| = \ell$ and $w[j] = q$; for the other cases, i.e., for $\varphi = \text{true}$, $\neg\varphi_1$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \wedge \varphi_2$, satisfaction is defined as usual. The language $\mathcal{L}(\varphi) \subseteq \Sigma^*$ of a formula is the set of configurations that satisfy φ . We also say that φ *denotes* the set $\mathcal{L}(\varphi)$. A formula is inductive if it denotes an inductive set.

► **Example 4.** In the dining philosophers, let $\varphi = (e_{1:4} \wedge b_{4:4}) \vee f_{2:4}$. We have

$$\mathcal{L}(\varphi) = \epsilon \mid \Sigma \mid \Sigma^2 \mid \Sigma^3 \mid (e \Sigma \Sigma b \mid \Sigma f \Sigma \Sigma) \mid \Sigma^5 \Sigma^* .$$

Observe that an expression like $(q_{1:1} \wedge r_{1:2})$ is not a formula, because it combines atomic propositions of two different lengths, which is not allowed. Notice also that $\neg q_{j:\ell}$ is equivalent to $\bigvee_{r \in \Sigma \setminus \{q\}} r_{j:\ell}$. Therefore, if we allowed negative conditions, we would still have the same class of expressible predicates on words of a given length (and we would not obtain formulas for the same predicates with fewer clauses.) Abusing language, if φ is a formula over AP_ℓ and $\mathcal{L}(\varphi)$ is an (inductive) invariant, then we also say that φ an (inductive) invariant. Observe that (inductive) invariants are closed under conjunction and disjunction.

Convention: From now on, “formula” means “positive formula in CNF”.

► **Definition 5.** Let $b \geq 0$. A b -formula is a formula with at most b clauses (with the convention that *true* is the only formula with 0 clauses). A set $S \subseteq \Sigma^*$ of configurations is b -bounded if for every length ℓ there exists a b -formula φ_ℓ over AP_ℓ such that $S \cap \Sigma^\ell = \mathcal{L}(\varphi_\ell)$.

Observe that, since one can always add tautological clauses to a formula without changing its language, a set S is b -bounded iff for every length ℓ there is a formula φ_ℓ with *exactly* b clauses.

► **Example 6.** In the dining philosophers, the 1-formulas $(t_{2i-1:\ell} \vee e_{2i-1:\ell})$ and $(f_{2i:\ell} \vee b_{2i:\ell})$ are inductive 1-invariants for every even $\ell \geq 1$ and every $i \in [1, \ell/2]$. It follows that the set I_0 of Example 3 is an intersection of (infinitely many) inductive 1-invariants. The same happens for I_1, \dots, I_5 . For example, I_1 is the intersection of all inductive 1-invariants of the form $(t_{i:\ell} \vee b_{i+1:\ell})$, for all $\ell \geq 1$ and all $i \in [1, \ell - 1]$; inductivity is shown by an easy case distinction.

We are now ready to define the sequence of inductive invariants we study in the paper:

► **Definition 7.** Let \mathcal{R} be a RTS. For every $b \geq 0$, we define $IndInv_b$ as the intersection of all inductive b -invariants of \mathcal{R} .

► **Proposition 8.** Let \mathcal{R} be a RTS. For every $b \geq 0$, $IndInv_b \supseteq Reach$ and $IndInv_b \supseteq IndInv_{b+1}$. Further, $Reach = \bigcap_{b=0}^{\infty} IndInv_b$.

Proof. $IndInv_b \supseteq Reach$ follows from the fact that, since inductive invariants are closed under intersection, $IndInv_b$ is an inductive invariant, and $Reach$ is the strongest inductive invariant. $IndInv_b \supseteq IndInv_{b+1}$ follows from the fact that, by definition, every b -invariant is also a $b+1$ -invariant. For the last part, observe that for every $\ell \geq 0$, the set $Reach \cap \Sigma^\ell$ is an inductive invariant for length ℓ . Let φ_ℓ be a formula over AP_ℓ such that $\mathcal{L}(\varphi_\ell) \cap \Sigma^\ell = Reach \cap \Sigma^\ell$, and let b_ℓ be its number of clauses. (Notice that φ_ℓ always exists, because every subset of Σ^ℓ can be expressed as a formula, and every formula can be put in conjunctive normal form.) Then φ_ℓ is a b_ℓ -bounded invariant, and so $\mathcal{L}(\varphi_\ell) \supseteq IndInv_{b_\ell}$ for every $\ell \geq 0$. So we have $Reach = \bigcap_{\ell=0}^{\infty} \mathcal{L}(\varphi_\ell) \supseteq \bigcap_{b=0}^{\infty} IndInv_b = Reach$, and we are done. ◀

Observe that, while $IndInv_b$ is always an inductive invariant, it is not necessarily b -bounded. The reason is that b -invariants are not closed under intersection. Indeed, the conjunction of two formulas with b clauses is not always equivalent to a formula with b clauses, one can only guarantee equivalence to a formula with $2b$ clauses.

► **Example 9.** The deadlocked configurations of the dining philosophers are

$$Dead = \overline{\Sigma^* f t f \Sigma^*} \cap \overline{\Sigma^* b e b \Sigma^*}.$$

We prove $IndInv_1 \cap Dead = \emptyset$, which implies that the dining philosophers are deadlock-free. Let C be the set of configurations of $((t \mid e)(f \mid b))^*$ containing no occurrence of ef , fe , or tbt as a cyclic word. In Example 6 we showed that C is an intersection of 1-invariants, which implies $IndInv_1 \subseteq C$. We prove $C \cap Dead = \emptyset$, which implies $IndInv_1 \cap Dead = \emptyset$. Let $w \in C$. If $|w| \leq 3$ the proof is an easy case distinction. Assume $|w| \geq 4$. We show that w contains an occurrence of ftf or beb , and so it is not a deadlock. If all philosophers are thinking at w , then, since w contains no occurrence of tbt , it contains an occurrence of ftf . If at least one philosopher is eating at w , then, since w contains no occurrence of eb or be , it contains an occurrence of beb .

Further, for the dining philosophers we have $Reach = IndInv_3$. Apart from some corner cases (e.g. an unsatisfiable invariant for every odd length), the reason is that the 3-formula

$$(t_{i:\ell} \vee b_{i+1:\ell}) \wedge (b_{i+1:\ell} \vee t_{i+2:\ell}) \wedge (t_{i:\ell} \vee f_{i+1:\ell} \vee t_{i+2:\ell})$$

is an inductive 3 invariant for every $\ell \geq 3$ and every $i \in [1, \ell - 2]$. The configurations satisfying this invariant and the inductive 1-invariants I_0, \dots, I_5 of Example 6 are the reachable configurations $Reach = (t(f \mid beb))^* \mid ebt((f \mid beb)t)^*b$.

It is not difficult to construct an (artificial) family \mathcal{R}_b of RTSs such that $IndInv_b \subset Reach(\mathcal{R}_b) = IndInv_{b+1}$.

► **Example 10.** Fix some $b > 0$. Consider a RTS with $\Sigma = \{0, 1\}$ and \mathcal{L}_T given by

$$\langle 0, 0 \rangle^{k_1} \langle 1, 0 \rangle \langle 0, 0 \rangle^{k_2} (\langle 0, 0 \rangle + \langle 0, 1 \rangle + \langle 1, 0 \rangle + \langle 1, 1 \rangle)^*$$

for every $k_1, k_2 \in \mathbb{N}$ such that $k_1 + k_2 = b - 1$. Then every transition of the RTS is of the form $u \cdot v \rightsquigarrow u' \cdot v'$, where $|u| = b = |u'|$, the word u contains exactly one 1, and the word u' contains only 0s. If we choose 0^* as set of initial configurations, then no transition is applicable to any initial configuration, and so $\text{Reach} = 0^*$. It is easy to check (see Appendix A of [21]) that $\text{IndInv}_b \supset \text{IndInv}_{b+1} = 0^* = \text{Reach}$. Also, one can easily check that if we set \mathcal{L}_T to

$$\langle 0, 0 \rangle^* \langle 1, 0 \rangle \langle 0, 0 \rangle^* (\langle 0, 0 \rangle + \langle 0, 1 \rangle + \langle 1, 0 \rangle + \langle 1, 1 \rangle)$$

then $\text{IndInv}_b \supset 0^* = \text{Reach}$ for every $b > 0$.

Finally, we show in the next section that IndInv_b is regular for every $b \geq 0$, which implies that any RTS \mathcal{R} such that Reach is not regular satisfies $\text{Reach} \neq \text{IndInv}_b$ for every $b \geq 0$.

3.3 IndInv_b is regular for every $b \geq 1$

We prove that IndInv_b is regular for every $b \geq 1$. For this, we first show how to encode b -formulas as words over the alphabet $(2^\Sigma)^b$, and then we prove the following two results:

1. The language of all b -formulas φ such that $\mathcal{L}(\varphi)$ is an inductive invariant is regular.
2. Given a regular language of b -formulas, the set of configurations that satisfy every formula in the language is regular.

Since IndInv_b contains the configurations that satisfy all the inductive b -invariants of \mathcal{R} , these two results imply that IndInv_b is regular.

Observe that a b -formula is an inductive invariant if it is satisfied by all initial configurations and is inductive. So we prove the second result in two steps. We first show that the set of b -formulas satisfied by all initial configurations is regular, and then that the set of all b -inductive formulas is regular.

Encoding b -formulas as b -powerwords. We introduce an encoding of b -formulas. We start with some examples. Assume \mathcal{R} is a RTS with $\Sigma = \{a, b, c\}$. We consider formulas over AP_3 , i.e., over the atomic propositions $\{a_{1:3}, a_{2:3}, a_{3:3}, b_{1:3}, b_{2:3}, b_{3:3}, c_{1:3}, c_{2:3}, c_{3:3}\}$.

We encode the 1-formula $(a_{1:3} \vee a_{2:3})$ as the word $\{a\}\{a\}\emptyset$ of length three over the alphabet 2^Σ . Intuitively, $\{a\}\{a\}\emptyset$ stands for the words of length 3 that have an a in their first or second position. Similarly, we encode $(a_{1:3} \vee b_{1:3} \vee b_{3:3})$ as $\{a, b\}\emptyset\{b\}$. Intuitively, $\{a, b\}\emptyset\{b\}$ stands for the set of words of length 3 that have a or b as first letter, or b as third letter. Since 2^Σ is the powerset of Σ , we call words over 2^Σ *powerwords*.

Consider now the 2-formula $(a_{1:3} \vee b_{1:3} \vee a_{2:3}) \wedge (b_{1:3} \vee b_{3:3} \vee c_{3:3})$. We put the encodings of its clauses “on top of each other”. Since the encodings of $(a_{1:3} \vee b_{1:3} \vee a_{2:3})$ and $(b_{1:3} \vee b_{3:3} \vee c_{3:3})$ are $\{a, b\}\{a\}\emptyset$ and $\{b\}\emptyset\{b, c\}$, respectively, we encode the formula as the word

$$\begin{bmatrix} \{a, b\} \\ \{b\} \end{bmatrix} \begin{bmatrix} \{a\} \\ \emptyset \end{bmatrix} \begin{bmatrix} \emptyset \\ \{b, c\} \end{bmatrix}$$

of length three over the alphabet $2^\Sigma \times 2^\Sigma = (2^\Sigma)^2$. We call such a word a *2-powerword*. Similarly, we encode a b -formula by a *b-powerword* of length three over the alphabet $(2^\Sigma)^b$.

In the following we overload φ to denote both a formula and its encoding as a b -powerword, and for example write

$$\varphi = \begin{bmatrix} X_{11} \\ \dots \\ X_{b1} \end{bmatrix} \cdots \begin{bmatrix} X_{1\ell} \\ \dots \\ X_{b\ell} \end{bmatrix} \quad \text{instead of} \quad \varphi = \bigwedge_{i=1}^b \bigvee_{j=1}^{\ell} \bigvee_{a \in X_{ij}} a_{i:\ell}$$

where $X_{ij} \subseteq \Sigma$. Intuitively, each row $X_{i1} \cdots X_{i\ell}$ encodes one clause of φ . We also write $\varphi = \varphi[1] \cdots \varphi[\ell]$, where $\varphi[i] \in (2^\Sigma)^b$ denotes the i -th letter of the b -powerword encoding φ . The satisfaction relation $w \models \varphi$ translates into a purely set-theoretical property:

► **Fact 11.** Let $w = w[1] \cdots w[\ell]$ be a configuration over Σ , and let $\varphi = \varphi[1] \cdots \varphi[\ell]$ be a b -formula, i.e., a b -powerword over $(2^\Sigma)^b$, where $\varphi[j] = \begin{bmatrix} X_{1j} \\ \dots \\ X_{bj} \end{bmatrix}$. We have $w \models \varphi$ iff for every $i \in [1, b]$ there exists $j \in [1, \ell]$ such that $w[j] \in X_{ij}$.

A DFA for the b -formulas satisfied by all initial configurations. Given a RTS \mathcal{R} and a bound $b \geq 0$, we let Init_b denote the set of all b -formulas satisfied by all initial configurations of \mathcal{R} . Recall that b -formulas are encoded as b -powerwords, and so Init_b is a language over the alphabet $(2^\Sigma)^b$. We show that Init_b is effectively regular:

► **Proposition 12.** Let $\mathcal{R} = \langle \Sigma, \mathcal{A}_I, \mathcal{A}_T \rangle$ be a RTS, and let n_I be the number of states of \mathcal{A}_I . For every $b \geq 1$, the language $\overline{\text{Init}_b}$ is recognized by a NFA with at most $n_I b$ states, and so Init_b is recognized by a DFA with at most $2^{b n_I}$ states.

Proof. $\overline{\text{Init}_b}$ contains the set of all b -formulas φ such that $w \not\models \varphi$ for some $w \in \mathcal{L}_I$. Let $\mathcal{A}_I = (Q_I, \Sigma, \delta_I, Q_{0I}, F_I)$. We consider only the case $b = 1$, the general case is handled in Appendix B of [21]. Let $\mathcal{B} = (Q_I, 2^\Sigma, \delta_B, Q_0, F_I)$ be the NFA with the same states, initial and final states as \mathcal{A} , and transition relation δ_B defined as follows for every $q \in Q_I$ and $X \in 2^\Sigma$:

$$q' \in \delta_B(q, X) \text{ iff there exists } a \in \Sigma \setminus X \text{ such that } q' \in \delta_I(q, a).$$

We show that \mathcal{B} recognizes $\overline{\text{Init}_1}$, i.e., that for every length $\ell \geq 0$, \mathcal{B} accepts a 1-formula φ iff there exists a configuration w such that $w \in \mathcal{L}_I$ and $w \not\models \varphi$. By Fact 11, this is the case iff there exists an accepting run $q_0 \xrightarrow{w[1]} q_1 \cdots q_{\ell-1} \xrightarrow{w[\ell]} q_\ell$ of \mathcal{A}_I such that $w[j] \notin \varphi[j]$ for every $j \in [1, \ell]$. By the definition of \mathcal{B} , this is the case iff $q_0 \xrightarrow{\varphi[1]} q_1 \cdots q_{\ell-1} \xrightarrow{\varphi[\ell]} q_\ell$ is an accepting run of \mathcal{B} . ◀

A DFA for the inductive b -formulas. For every $b \geq 0$, let Ind_b be the set of inductive b -formulas. We show that Ind_b is effectively regular:

► **Proposition 13.** Let $\mathcal{R} = \langle \Sigma, \mathcal{A}_I, \mathcal{A}_T \rangle$ be an RTS, and let n_T be the number of states of \mathcal{A}_T . $\overline{\text{Ind}_b}$ is recognized by a NFA with at most $n_T b 2^b$ states, and so Ind_b is recognized by a DFA with at most $2^{n_T b 2^b}$ states.

Proof. Let $\mathcal{A}_T = (Q_T, \Sigma, \delta_T, Q_{0T}, F_T)$. We consider only the case $b = 1$, for the general case see Appendix B of [21].

Let $\mathcal{C} = (Q_C, 2^\Sigma \times \Sigma, \delta_C, Q_{0C}, F_C)$ be a transducer accepting the words $\langle \varphi, w \rangle$ such that $\varphi \in (2^\Sigma)^*$ is a 1-formula and $w \models \varphi$, i.e., $w[j] \in \varphi[j]$ for at least one $j \in [1, \ell]$ (Fact 11). It is trivial to construct a transducer for this language with two states.

We define the NFA $\mathcal{B} = (Q_C \times Q_T, 2^\Sigma, \delta_B, Q_{0C} \times Q_{0T}, F_C \times F_T)$ over 2^Σ with transition relation δ_B as follows:

$(q', r') \in \delta_B((q, r), X)$ iff $\exists a_1 \in \Sigma, a_2 \in \Sigma \setminus X: q' \in \delta_C(q, \langle X, a_1 \rangle) \wedge r' \in \delta_T(r, \langle a_1, a_2 \rangle)$.

We show that \mathcal{B} recognizes $\overline{Ind_1}$. A 1-formula φ is not inductive iff there exist configurations w, u satisfying three conditions: $w \models \varphi$, i.e., $w[j] \in \varphi[j]$ for some $j \in [1, \ell]$; $\langle w, u \rangle \in \mathcal{L}(\mathcal{A}_t)$; and $u \not\models \varphi$, i.e., $u[j] \notin \varphi[j]$ for every $j \in [1, \ell]$ (Fact 11). By the definition of \mathcal{C} , this is the case iff there are accepting runs

$$q_0 \xrightarrow{\langle \varphi[1], w[1] \rangle} q_1 \cdots q_{\ell-1} \xrightarrow{\langle \varphi[\ell], w[\ell] \rangle} q_\ell \quad \text{and} \quad r_0 \xrightarrow{\langle w[1], u[1] \rangle} r_1 \cdots r_{\ell-1} \xrightarrow{\langle w[\ell], u[\ell] \rangle} r_\ell$$

of \mathcal{C} and \mathcal{A}_T , respectively. By the definition of \mathcal{B} , this the case iff

$$(q_0, r_0) \xrightarrow{\varphi[1]} (q_1, r_1) \cdots (q_{\ell-1}, r_{\ell-1}) \xrightarrow{\varphi[\ell]} (q_\ell, r_\ell)$$

is an accepting run of \mathcal{B} . ◀

A DFA for the set of configurations satisfying a regular set of b -formulas. Given a NFA \mathcal{A} over the alphabet $(2^\Sigma)^b$, i.e., a NFA recognizing a language of b -formulas, let $Sat(\mathcal{A})$ be the set of configurations satisfying all b -formulas of $\mathcal{L}(\mathcal{A})$.

► **Proposition 14.** *Let \mathcal{R} be a RTS over Σ , and let \mathcal{A} be a NFA over $(2^\Sigma)^b$ with m states. $\overline{Sat(\mathcal{A})}$ is recognized by a NFA with at most mb states, and so $Sat(\mathcal{A})$ is recognized by a DFA with at most 2^{mb} states.*

Proof. Let $\mathcal{A} = (Q, 2^\Sigma, \delta_A, Q_0, F)$. We consider only the case $b = 1$, for the general case see Appendix B of [21]. Let φ be a 1-formula of length ℓ . By Fact 11 we have $w \models \varphi$ iff $\bigvee_{j=1}^{\ell} w[j] \in \varphi[j]$

Consider the NFA $\mathcal{B} = (Q, \Sigma, \delta_B, Q_0, F)$ over Σ with the same states, initial and final states as \mathcal{A} , and transition relation defined as follows:

$$q' \in \delta_B(q, a) \text{ iff there exists } X \in 2^\Sigma \text{ such that } q' \in \delta_A(q, X) \text{ and } a \notin X.$$

We show that \mathcal{B} recognizes $\overline{Sat(\mathcal{A})}$. More precisely, we show that $w \notin Sat(\mathcal{A})$ iff $w \in \mathcal{L}(\mathcal{B})$ holds for every configuration w . Let $\ell \geq 0$, and let w be an arbitrary configuration of length ℓ . We have $w \notin Sat(\mathcal{A})$ iff there is an accepting run $q_0 \xrightarrow{\varphi[1]} q_1 \cdots q_{\ell-1} \xrightarrow{\varphi[\ell]} q_\ell$ of \mathcal{A} such that $w[j] \notin \varphi[j]$ for every $j \in [1, \ell]$. By the definition of \mathcal{B} , this is the case iff $q_0 \xrightarrow{\varphi[1]} q_1 \cdots q_{\ell-1} \xrightarrow{\varphi[\ell]} q_\ell$ is an accepting run of \mathcal{B} . ◀

Putting everything together. We combine the previous results to show that, given a RTS \mathcal{R} , the complement of $IndInv_b$ is recognized by a NFA whose number of states is exponential in \mathcal{R} and double exponential in b .

► **Theorem 15.** *Let $\mathcal{R} = \langle \Sigma, \mathcal{A}_I, \mathcal{A}_T \rangle$ be a RTS. Let n_I and n_T be the number of states of \mathcal{A}_I and \mathcal{A}_T , respectively, and let $K = n_I b + n_T 2^b$. For every $b \geq 0$, the set $\overline{IndInv_b}$ is recognized by a NFA with at most $2^K b$ states, and so $IndInv_b$ is recognized by a DFA with at most 2^{b2^K} states.*

Proof. Recall that $IndInv_b$ contains the configurations satisfying all b -formulas that are inductive invariants of \mathcal{R} . A b -formula is an inductive invariant iff it is inductive and it is satisfied by all initial configurations of \mathcal{R} . So $Init_b \cap Ind_b$ is the language of all b -formulas (equivalently, all b -powerwords) that are inductive invariants. By Propositions 12 and 13, this language is recognized by a DFA \mathcal{A} with at most $2^{n_I b} \cdot 2^{n_T 2^b} = 2^K$ states. A configuration w belongs to $IndInv_b$ iff it satisfies every formula of $Init_b \cap Ind_b$, i.e., $IndInv_b = Sat(\mathcal{A})$. By Proposition 14, $\overline{IndInv_b}$ is recognized by a NFA with $2^K b$ states. ◀

4 Deciding $IndInv_1 \cap \mathcal{U} = \emptyset$ is PSPACE-complete

Given an instance \mathcal{R}, \mathcal{U} of the safety verification problem and $b \geq 0$, if the set $IndInv_b$ satisfies $IndInv_b \cap \mathcal{L}(\mathcal{U}) = \emptyset$, then \mathcal{R} is safe. By Theorem 15, deciding whether $IndInv_b \cap \mathcal{L}(\mathcal{U}) = \emptyset$ holds is in EXSPACE for every fixed b . Indeed, the theorem and its proof show that there is a NFA recognizing $IndInv_b \cap \mathcal{L}(\mathcal{U})$ such that one can guess an accepting path of it, state by state, using exponential space. We do not know if there is a b such that the problem is EXSPACE-complete for every $b' \geq b$. In this section we show that for $b = 1$ the problem is actually PSPACE-complete.

4.1 Deciding $IndInv_1 \cap \mathcal{L}(\mathcal{U}) = \emptyset$ is in PSPACE

We give a non-deterministic polynomial space algorithm that decides $IndInv_1 \cap \mathcal{L}(\mathcal{U}) = \emptyset$. As a byproduct, we show that $IndInv_1$ is recognized by a NFA with a single exponential number of states. (Notice that we proved this for $\overline{IndInv_1}$ in Proposition 13, but not for $IndInv_1$.) All missing proofs and full versions of proof sketches can be found in the appendices of [21].

1-formulas have a special property: since the disjunction of two clauses is again a clause, the disjunction of two 1-formulas is again a 1-formula. This allows us to define the *separator* of a configuration w .

► **Definition 16.** *The separator of a configuration w , denoted Sep_w , is the union of all inductive 1-sets not containing w .*

We characterize membership of w in $IndInv_1$ in terms of its separator:

► **Lemma 17.** *For every configuration w , its separator Sep_w is an inductive 1-set. Further $w \in IndInv_1$ iff Sep_w is not an invariant.*

Proof. Since inductive sets are closed under union, Sep_w is inductive. Since the disjunction of two clauses is again a clause, the union of two 1-sets of configurations is also a 1-set, and so Sep_w is an inductive 1-set. For the last part, we prove that $w \notin IndInv_1$ iff Sep_w is an invariant. Assume first $w \notin IndInv_1$. Then some inductive 1-invariant does not contain w . Since, by definition, Sep_w contains this invariant, Sep_w is also an invariant. Assume now that Sep_w is an invariant. Then Sep_w is an inductive 1-invariant, and so $Sep_w \supseteq IndInv_1$. Since $w \notin Sep_w$, we get $w \notin IndInv_1$. ◀

Our plan for the rest of the section is as follows:

- We introduce the notion of a separation table for a configuration. (Definition 18)
- We show that, given a configuration w and a separation table τ for w , we can construct a 1-formula $\varphi_{Sep_w^\tau}$ such that $\mathcal{L}(\varphi_{Sep_w^\tau}) = Sep_w$. (Lemma 19)
- We use this result to define a transducer T_{sep} over $\Sigma \times 2^\Sigma$ that accepts a word $\langle w, \varphi \rangle$ iff $\varphi = \varphi_{Sep_w^\tau}$. (Proposition 21)
- We use T_{sep} and Proposition 12 to define a NFA over Σ that accepts a configuration w iff Sep_w is not an invariant, and so, by Lemma 17, iff $w \in IndInv_1$. (Theorem 22)

We present a characterization of Sep_w in terms of *tables*. Given a transition $s \rightsquigarrow t$, we call s and t the *source* and *target* of the transition, respectively. A *table* of length ℓ is a sequence $\tau = s_1 \rightsquigarrow t_1, \dots, s_n \rightsquigarrow t_n$ of transitions of \mathcal{R} (not necessarily distinct), all of length ℓ .¹ We define the *separation tables* of a configuration w .

¹ We call it table because we visualize $s_1, t_1, \dots, s_n, t_n$ as a matrix with $2n$ rows and ℓ columns.

► **Definition 18.** Let w be a configuration and let $\tau = s_1 \rightsquigarrow t_1, \dots, s_n \rightsquigarrow t_n$ be a table, both of length ℓ . For every $j \in [1, \ell]$, let $In(w, \tau)[j] = \{w[j], s_1[j], \dots, s_n[j]\}$ be the set of letters at position j of w and of the source configurations of the table.

- τ is consistent with w if for every $i \in [1, n], j \in [1, \ell]$, either $t_i[j] = w[j]$ or $t_i[j] = s_{i'}[j]$ for some $i' < i$.

(Intuitively: τ is consistent with w if for every position, the letter of the target is either the letter of w , or the letter of some earlier source.)

- τ is complete for w if every table $\tau' (s \rightsquigarrow t)$ consistent with w satisfies $s[j] \in In(w, \tau)$ for every $j \in [1, \ell]$.

(Intuitively: τ is complete for w if it cannot be extended by a transition that maintains consistency and introduces a new letter.)

A table is a separation table of w if it is consistent with and complete for w .

Observe that every configuration w has at least one separation table. If there are no transitions with target w , then the empty table with no transitions is a separation table. Otherwise, starting with any transition $s \rightsquigarrow w$, we repeatedly add a transition, maintaining consistency and introducing at least one new letter, until no such transition exists. This procedure terminates – there are only finitely many transitions between configurations of a fixed length – and yields a separation table.

The next lemma shows how to compute a 1-formula $\varphi_{Sep_w^\tau}$ such that $\mathcal{L}(\varphi_{Sep_w^\tau}) = Sep_w$ from any separation table τ of w .

► **Lemma 19.** Let τ be any separation table for a configuration w of length ℓ . Then Sep_w is the set of all configurations $z \in \Sigma^\ell$ such that $z[j] \notin In(w, \tau)[j]$ for some $j \in [1, \ell]$. In particular, Sep_w is the language of the 1-formula

$$\varphi_{Sep_w^\tau} := \bigvee_{j=1}^{\ell} \left(\bigvee_{a \notin In(w, \tau)[j]} a_{j:\ell} \right)$$

or, in the powerword encoding, of the formula

$$\varphi_{Sep_w^\tau} = \overline{In(w, \tau)[1]} \cdots \overline{In(w, \tau)[\ell]}.$$

Proof. It follows easily from the definitions that $\varphi_{Sep_w^\tau}$ denotes an inductive 1-set not containing w . To prove that it is the largest such set it suffices to show that for every $j \in [1, \ell]$ and every letter $x \in In(w, \tau)[j]$, the configuration w belongs to every inductive 1-set specified by a powerword containing x at position j . For this, we consider the tables $\tau_0, \tau_1, \dots, \tau_n = \tau$, where τ_i is the prefix of τ of length i , and prove by induction on k that the property holds for every τ_k . ◀

We construct a transducer over the alphabet $\Sigma \times 2^\Sigma$ that transduces a configuration w into the formula $\varphi_{Sep_w^\tau}$ of a table τ consistent with and complete for w . For this we need the consistency and completeness summaries of a table.

► **Definition 20.** Let $\tau = s_1 \rightsquigarrow t_1, \dots, s_n \rightsquigarrow t_n$ be a separation table for a configuration w . The consistency summary is the result of applying the following procedure to τ :

- Replace each row $s_i \rightsquigarrow v_i$ by the sequence of states of an accepting run of \mathcal{A}_T on it. (This produces a table with i rows and $\ell + 1$ columns, whose entries are states of \mathcal{A}_T .)
- In each column, keep the first occurrence of each state, removing the rest. (The result is a sequence of columns of possibly different lengths.)

The completeness summary is the sequence $(Q_0, Q'_0), (Q_1, Q'_1) \dots (Q_\ell, Q'_\ell)$ of pairs of sets of states of \mathcal{A}_T , defined inductively as follows for every $j \in [0, \ell]$:

- Q_0 is the set of initial states and Q'_0 is empty.
- Q_{j+1} is the set of states reachable from Q_j by means of letters $[a, b]$ such that $b \in \text{In}(w, \tau)$.
- Q'_{j+1} is the set of states reachable from Q'_j by means of letters $[a, b]$ such that $b \in \text{In}(w, \tau)$, or reachable from Q_j by means of letters $[a, b]$ such that $a \notin \text{In}(w, \tau)$ and $b \in \text{In}(w, \tau)$.

Observe that the consistency summary is a sequence $\alpha = \alpha[1] \dots \alpha[\ell]$, where $\alpha[i]$ is a sequence of *distinct* states of \mathcal{A}_T , i.e., an element of $Q_T^{n_T}$, and the completeness summary is a sequence $\beta = \beta[1] \dots \beta[\ell]$, where $\beta[i]$ is a pair of sets of states of \mathcal{A}_T , i.e., an element of $2^{Q_T} \times 2^{Q_T}$. We prove in Appendix C of [21]:

► **Proposition 21.** *There exists a transducer T_{sep} over the alphabet $\Sigma \times 2^\Sigma$ satisfying the following properties:*

- The states of T_{sep} are elements of $(Q_T \cup \{\square\})^{n_T} \times (2^{Q_T} \times 2^{Q_T})$, where n_T is the number of states of \mathcal{A}_T .
- There is a polynomial time algorithm that, given two states q, q' of T_{sep} and a letter $\langle a, X \rangle \in \Sigma \times 2^\Sigma$ decides whether the triple $(q, \langle a, X \rangle, q')$ is a transition of T_{sep} .
- T_{sep} recognizes a word $\langle w, \varphi \rangle$ over $\Sigma \times 2^\Sigma$ iff $\varphi = \varphi_{sep_w}$.

We can now use Theorem 12 to obtain our main result:

► **Theorem 22.** *Let $\mathcal{R} = \langle \Sigma, \mathcal{A}_I, \mathcal{A}_T \rangle$ be a RTS. There exists a NFA \mathcal{A}_1 over Σ satisfying the following properties:*

- The states of \mathcal{A}_1 are elements of $Q_T^{n_T} \times (2^{Q_T} \times 2^{Q_T}) \times Q_I$.
- There is a polynomial time algorithm that, given two states q, q' of \mathcal{A}_1 and a letter $a \in \Sigma$ decides whether the triple (q, a, q') is a transition of T_{sep} .
- $\mathcal{L}(\mathcal{A}_1) = \text{IndInv}_1$

Proof. Let T_{sep} be the transducer over the alphabet $\Sigma \times 2^\Sigma$ of Proposition 21. Let $\mathcal{A}_{\overline{\text{init}}}$ be a NFA recognizing $\overline{\text{Init}_1}$, i.e., the language of all 1-formulas satisfied by all initial configurations, or, in other words, all 1-formulas that are invariants. By Lemma 17, $w \in \text{IndInv}_1$ iff there exists a 1-formula φ such that $\langle w, \varphi \rangle \in \mathcal{L}(T_{sep})$ and $\varphi \in \mathcal{L}(\mathcal{A}_{\overline{\text{init}}})$. So there exists a NFA \mathcal{A}_1 for IndInv_1 whose states are the pairs $\langle q, r \rangle$ such that q is a state of T_{sep} and r a state of $\mathcal{A}_{\overline{\text{init}}}$. Since, by Proposition 12, $\mathcal{A}_{\overline{\text{init}}}$ has the same states as \mathcal{A}_I , the result follows. ◀

Observe that a state of \mathcal{A}_1 can be stored using space linear in \mathcal{A}_I and \mathcal{A}_T .

► **Corollary 23.** *Deciding $\text{IndInv}_1 \cap \mathcal{L}(\mathcal{U}) = \emptyset$ is in PSPACE.*

Proof. Guess a configuration w and an accepting run of \mathcal{A}_1 and \mathcal{U} on w , step by step. By Proposition 21, this can be done in polynomial space. Apply then $\text{NPSPACE} = \text{PSPACE}$. ◀

4.2 Deciding $\text{IndInv}_1 \cap \mathcal{L}(\mathcal{U}) = \emptyset$ is PSPACE-hard

Given a linearly bounded Turing machine, we construct a sequence \mathcal{R}_n of RTSs such that the instance of \mathcal{R}_n for some length $\Theta(t \times n)$ simulates the Turing machine on inputs of length n up to t steps. Moreover, we show that, for this RTS, $\text{IndInv}_1 = \text{Reach}(\mathcal{R}_n)$ holds. Therefore, we can reduce acceptance of the Turing machine to our problem (see Appendix C.1 of [21]).

► **Lemma 24.** *Deciding $\text{IndInv}_1 \cap \mathcal{L}(\mathcal{U}) = \emptyset$ is PSPACE-hard.*

System	$ \mathcal{L}_T $	$ \mathcal{L}_T $	$ IndInv_1 $	Properties	time (ms)
Bakery	5	9	8	deadlock	✓
				mutual exclusion	✓
Burns	5	9	6	deadlock	✓
				mutual exclusion	✓
Dijkstra	6	24	22	deadlock	✓
				mutual exclusion	✓
Dijkstra (ring)	6	17	17	deadlock	✓
				mutual exclusion	×
D. cryptographers	6	69	11	one cryptographer paid	✓
				no cryptographer paid	✓
Herman, linear	6	7	6	deadlock	×
				at least one token	✓
Herman, ring	6	7	7	deadlock	✓
				at least one token	✓
Israeli-Jafon	6	21	7	deadlock	✓
				at least one token	✓
Token passing	6	7	7	at most one token	✓
Lehmann-Rabin	5	15	13	deadlock	✓
LR phils.	6	14	15	deadlock	×
LR phils.(with b_ℓ and b_r)	5	14	9	deadlock	✓
Atomic D. phil.	5	12	20	deadlock	✓
Mux array	6	7	8	deadlock	✓
				mutual exclusion	×
Res. allocator	5	9	8	deadlock	✓
				mutual exclusion	×
Berkeley	5	19	9	deadlock	✓
				custom properties	$\frac{2}{3}$
Dragon	5	26	11	deadlock	✓
				custom properties	$\frac{6}{7}$
Firefly	5	18	7	deadlock	✓
				custom properties	$\frac{9}{4}$
Illinois	5	25	14	deadlock	✓
				custom properties	$\frac{9}{2}$
MESI	5	13	7	deadlock	✓
				custom properties	$\frac{2}{2}$
MOESI	5	13	10	deadlock	✓
				custom properties	$\frac{7}{7}$
Synapse	5	16	7	deadlock	✓
				custom properties	$\frac{2}{2}$

■ **Figure 1** Experimental results of using $IndInv_1$ as abstraction of the set of reachable configurations.

5 How large must the bound b be?

The index b needed to prove a property (i.e., the least b such that $IndInv_b$ implies the property) can be seen as a measure of how difficult it is for a human to understand the proof. We use the experimental setup of [12, 19, 13, 20], where systems are encoded as WS1S formulas and MONA [17] is used as computation engine, to show that $b = 1$ is enough for a substantial number of benchmarks used in the RMC literature. Note that our goal is to evaluate the complexity of invariants needed for systems from diverse domains, not to present a tool ready to verify complex systems.

Our set of benchmarks consists of problems studied in [14, 3, 12, 19, 13, 20]. In a first step, we use MONA to construct a minimal DFA for $IndInv_1$. For this, we write a WS1S formula $\Psi_1(w)$ expressing that, for every 1-formula φ , if φ is an inductive invariant, then w satisfies φ . MONA yields a minimal DFA for the configurations w satisfying Ψ_1 , which is precisely $IndInv_1$. We then construct the formula $\Psi_1(w) \wedge Unsafe(w)$, and use MONA to check if it is satisfiable². All files containing the MONA formulas and the results are provided in [18]. The results are shown in Figure 1. The first column gives the name of the example.

² The second formula $\Psi_1(w) \wedge Unsafe(w)$ being unsatisfiable suffices for verification purposes, but we use $\Psi_1(w)$ to obtain information on the size of the minimal DFA.

In the second and third column we give the number of states of the minimal DFA for \mathcal{L}_I and \mathcal{L}_T , which we also compute via MONA. In the next column we give the size of the minimal DFA for $IndInv_1$. The fifth column reports whether a property is implied by $IndInv_1$ (indicated by \checkmark) or not (indicated by \times). For the cache coherence protocols we replace \checkmark with k/m to state that k of m custom safety properties can be established. The last column gives the total time³. As we can see, $IndInv_1$ is strong enough to satisfy 46 out of 57 properties.

In a second step, we have studied some of the cases in which $IndInv_1$ is not strong enough. Direct computation of the automaton for $IndInv_2$ from a formula $\Psi_2(w)$ using MONA fails. (A direct computation based on the automata construction of Section 3 might yield better results, and will be part of our future work.) Using a combination of the automatic invariant computation method of [12, 19, 13, 20] and manual inspection of the returned invariants, we can report some results for some examples.

Examples for $IndInv_b$ with $b > 1$. Table 1 contains two versions of the dining philosophers in which philosophers take one fork at a time. All philosophers but one are right-handed, i.e., take their right fork first, and the remaining philosopher is left handed. If the forks “know” which philosopher has taken them (i.e., if they have states b_ℓ and b_r indicating that the left or the right philosopher has the fork), then deadlock-freedom can be proved using $IndInv_1$. If the states of the forks are just “free” and “busy”, then proving deadlock-freedom requires $IndInv_3$, and in fact $Reach = IndInv_3$ holds. We show how to establish this using the technique of [20] and some additional reasoning in Appendix A of [21].

The Berkeley and Dragon cache coherence protocols are considered as parameterized system in [16]. For both examples $IndInv_1$ is too coarse to establish all desired consistency assertions. In Appendix B we describe the formalization of both examples and show that $IndInv_2$ suffices to obtain the missing assertions.

6 Conclusion

We have introduced a regular model checking paradigm that approaches the set of reachable configurations from above. As already observed in [3, 14], such an approach does not require widening or acceleration techniques, as is the case when approaching from below. The main novelty with respect to [3, 14] is the discovery of a natural sequence of regular invariants converging to the set of reachable configurations.

Our new paradigm raises several questions. The first one is the exact computational complexity of checking emptiness of the intersection $IndInv_b$ and the unsafe configurations. We have shown PSPACE-completeness for $b = 1$, and we conjecture that the problem is already EXPSPACE-complete for all $b \geq 2$. We also think that the CEGAR techniques used in [20, 19] can be extended to the RMC setting, allowing one to compute intermediate regular invariants between $IndInv_b$ and $IndInv_{b+1}$. Another interesting research venue is the combination with acceleration or widening techniques, and the application of learning algorithms, like the one of [14]. Currently these techniques try to compute some inductive regular invariant, or perhaps one described by small automata, which may lead to invariants difficult to interpret by humans. A better approach might be to stratify the search, looking first for invariants for small values of b .

³ As reported by MONA.

References

- 1 Parosh Aziz Abdulla. Regular model checking. *Int. J. Softw. Tools Technol. Transf.*, 14(2):109–118, 2012.
- 2 Parosh Aziz Abdulla. Regular model checking: Evolution and perspectives. In *Model Checking, Synthesis, and Learning*, volume 13030 of *Lecture Notes in Computer Science*, pages 78–96. Springer, 2021.
- 3 Parosh Aziz Abdulla, Giorgio Delzanno, Noomene Ben Henda, and Ahmed Rezine. Regular model checking without transducers (on efficient verification of parameterized systems). In *TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 721–736. Springer, 2007.
- 4 Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Julien d’Orso. Regular model checking made simple and efficient. In *CONCUR*, volume 2421 of *Lecture Notes in Computer Science*, pages 116–130. Springer, 2002.
- 5 Parosh Aziz Abdulla, Bengt Jonsson, Marcus Nilsson, and Mayank Saksena. A survey of regular model checking. In *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 35–48. Springer, 2004.
- 6 Parosh Aziz Abdulla, A. Prasad Sistla, and Muralidhar Talupur. Model checking parameterized systems. In *Handbook of Model Checking*, pages 685–725. Springer, 2018.
- 7 Bernard Boigelot, Axel Legay, and Pierre Wolper. Iterating transducers in the large (extended abstract). In *CAV*, volume 2725 of *Lecture Notes in Computer Science*, pages 223–235. Springer, 2003.
- 8 Ahmed Bouajjani, Peter Habermehl, Adam Rogalewicz, and Tomas Vojnar. Abstract regular (tree) model checking. *Int. J. Softw. Tools Technol. Transf.*, 14(2):167–191, 2012.
- 9 Ahmed Bouajjani, Peter Habermehl, and Tomas Vojnar. Abstract regular model checking. In *CAV*, volume 3114 of *Lecture Notes in Computer Science*, pages 372–386. Springer, 2004.
- 10 Ahmed Bouajjani, Bengt Jonsson, Marcus Nilsson, and Tayssir Touili. Regular model checking. In *CAV*, volume 1855 of *Lecture Notes in Computer Science*, pages 403–418. Springer, 2000.
- 11 Ahmed Bouajjani and Tayssir Touili. Widening techniques for regular tree model checking. *Int. J. Softw. Tools Technol. Transf.*, 14(2):145–165, 2012.
- 12 Marius Bozga, Javier Esparza, Radu Iosif, Joseph Sifakis, and Christoph Welzel. Structural invariants for the verification of systems with parameterized architectures. In *TACAS (1)*, volume 12078 of *Lecture Notes in Computer Science*, pages 228–246. Springer, 2020.
- 13 Marius Bozga, Radu Iosif, and Joseph Sifakis. Checking deadlock-freedom of parametric component-based systems. *J. Log. Algebraic Methods Program.*, 119:100621, 2021.
- 14 Yu-Fang Chen, Chih-Duo Hong, Anthony W. Lin, and Philipp Rummer. Learning to prove safety over parameterised concurrent systems. In *FMCAD*, pages 76–83. IEEE, 2017.
- 15 Dennis Dams, Yassine Lakhnech, and Martin Steffen. Iterating transducers. In *CAV*, volume 2102 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 2001.
- 16 Giorgio Delzanno. Constraint-based verification of parameterized cache coherence protocols. *Formal Methods Syst. Des.*, 23(3):257–301, 2003.
- 17 Jacob Elgaard, Nils Klarlund, and Anders Moller. MONA 1.x: new techniques for WS1S and WS2S. In *Proc. 10th International Conference on Computer-Aided Verification, CAV ’98*, volume 1427 of *LNCS*, pages 516–520. Springer-Verlag, June/July 1998.
- 18 Javier Esparza, Mikhail Raskin, and Christoph Welzel. Repository of examples. <https://doi.org/10.5281/zenodo.6483615>, April 2022. doi:10.5281/zenodo.6483615.
- 19 Javier Esparza, Mikhail A. Raskin, and Christoph Welzel. Abduction of trap invariants in parameterized systems. In *GandALF*, volume 346 of *EPTCS*, pages 1–17, 2021.
- 20 Javier Esparza, Mikhail A. Raskin, and Christoph Welzel. Computing parameterized invariants of parameterized petri nets. In *Petri Nets*, volume 12734 of *Lecture Notes in Computer Science*, pages 141–163. Springer, 2021.
- 21 Javier Esparza, Mikhail A. Raskin, and Christoph Welzel. Regular model checking upside-down: An invariant-based approach. *CoRR*, abs/2205.03060, 2022. arXiv:2205.03060.

- 22 Bengt Jonsson and Marcus Nilsson. Transitive closures of regular relations for verifying infinite-state systems. In *TACAS*, volume 1785 of *Lecture Notes in Computer Science*, pages 220–234. Springer, 2000.
- 23 Axel Legay. Extrapolating (omega-)regular model checking. *Int. J. Softw. Tools Technol. Transf.*, 14(2):119–143, 2012.
- 24 Kenneth L. McMillan and Oded Padon. Ivy: A multi-modal verification tool for distributed algorithms. In *CAV (2)*, volume 12225 of *Lecture Notes in Computer Science*, pages 190–202. Springer, 2020.
- 25 Oded Padon, Kenneth L. McMillan, Aurojit Panda, Mooly Sagiv, and Sharon Shoham. Ivy: safety verification by interactive generalization. In *PLDI*, pages 614–630. ACM, 2016.
- 26 Amir Pnueli, Sitvanit Ruah, and Lenore D. Zuck. Automatic deductive verification with invisible invariants. In *TACAS*, volume 2031 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2001.

A Dining philosophers with one left-handed philosopher

We sketch the formalization of the case in which the states of the forks are only “free” and “busy”. Consider a RTS with $\Sigma = \{f, b, t, h, e\}$. The state h represents a philosopher who already grabbed the first fork and waits for the second one. All other states are used as before. The philosopher at index 1 takes first the fork at index 2 and then the fork at index n , while any other philosopher $i > 1$ first takes the fork at index $i - 1$ and then the fork at index $i + 1$.

In [20] the absence of deadlocks in this example is shown via only a few inductive assertions. These assertions can be equivalently expressed as 3-invariants. Moreover, these assertions are actually enough to completely characterize *Reach* in this example. To this end, observe that, analogously to Example 2, *Reach* is completely characterized by the absence of a few *invalid* patterns. These patterns separate into three cases: First, a philosopher should use some fork, but this fork is still considered free. Second, two philosophers are in states that require the same fork. Third, no adjacent philosopher currently uses some fork, yet this fork is busy. More formally, we get

- $\Sigma (\Sigma \Sigma)^* f (h | e) \Sigma (\Sigma \Sigma)^*, (\Sigma \Sigma)^+ e f (\Sigma \Sigma)^*, (e | h) f (\Sigma \Sigma)^*, e (\Sigma \Sigma)^* f,$
- $(\Sigma \Sigma)^+ e \Sigma (h | e) \Sigma (\Sigma \Sigma)^*, (e | h) \Sigma (e | h) \Sigma (\Sigma \Sigma)^*, e \Sigma (\Sigma \Sigma)^* e \Sigma,$
- $t b t \Sigma (\Sigma \Sigma)^*, (t | h) \Sigma (\Sigma \Sigma)^* (t | h) b, (\Sigma \Sigma)^+ (t | h) b t \Sigma (\Sigma \Sigma)^*,$

The absence of these patterns can be established with the following languages of inductive 1-invariants and inductive 3-invariants:

$$\begin{aligned} & [\{e\}] [\emptyset] ([\emptyset] [\emptyset])^* [\{e\}] [\{f\}] \\ & [\{e, h\}] [\{f\}] [\{e, h\}] [\emptyset] ([\emptyset] [\emptyset])^* \\ & [\emptyset] [\emptyset] ([\emptyset] [\emptyset])^* [\{e\}] [\{f\}] [\{e, h\}] [\emptyset] ([\emptyset] [\emptyset])^* \end{aligned} \quad \begin{aligned} & \begin{bmatrix} \{t, h\} \\ \{t, h\} \\ \emptyset \end{bmatrix} \begin{bmatrix} [\emptyset] \\ \emptyset \\ \emptyset \end{bmatrix} \left(\begin{bmatrix} [\emptyset] \\ \emptyset \\ \emptyset \end{bmatrix} \begin{bmatrix} [\emptyset] \\ \emptyset \\ \emptyset \end{bmatrix} \right)^* \begin{bmatrix} \emptyset \\ \{t, h\} \\ \{b\} \end{bmatrix} \begin{bmatrix} [\emptyset] \\ \emptyset \\ \emptyset \end{bmatrix} \\ & \begin{bmatrix} \{t\} \\ \{t\} \\ \emptyset \end{bmatrix} \begin{bmatrix} \{b\} \\ \emptyset \\ \{b\} \end{bmatrix} \begin{bmatrix} [\emptyset] \\ \emptyset \\ \emptyset \end{bmatrix} \begin{bmatrix} \{t\} \\ \emptyset \\ \emptyset \end{bmatrix} \left(\begin{bmatrix} [\emptyset] \\ \emptyset \\ \emptyset \end{bmatrix} \begin{bmatrix} [\emptyset] \\ \emptyset \\ \emptyset \end{bmatrix} \right)^* \\ & \begin{bmatrix} [\emptyset] \\ \emptyset \\ \emptyset \end{bmatrix} \begin{bmatrix} [\emptyset] \\ \emptyset \\ \emptyset \end{bmatrix} \left(\begin{bmatrix} [\emptyset] \\ \emptyset \\ \emptyset \end{bmatrix} \begin{bmatrix} [\emptyset] \\ \emptyset \\ \emptyset \end{bmatrix} \right)^* \begin{bmatrix} \{t, h\} \\ \{t, h\} \\ \emptyset \end{bmatrix} \begin{bmatrix} \emptyset \\ \{b\} \\ \{t\} \end{bmatrix} \begin{bmatrix} \{t\} \\ \emptyset \\ \emptyset \end{bmatrix} \begin{bmatrix} [\emptyset] \\ \emptyset \\ \emptyset \end{bmatrix} \left(\begin{bmatrix} [\emptyset] \\ \emptyset \\ \emptyset \end{bmatrix} \begin{bmatrix} [\emptyset] \\ \emptyset \\ \emptyset \end{bmatrix} \right)^* \end{aligned}$$

Consequently, *IndInv*₃ and *Reach* coincide for this example. This immediately implies that *IndInv*₃ proves deadlock-freedom since the system actually is deadlock-free.

However, *IndInv*₂ is insufficient to prove deadlock-freedom: assume there exists some inductive 2-invariant I that invalidates that $D = h b t f e b$ can be reached. Then, I must separate all elements from *Reach* and all configurations D' with $D' \rightsquigarrow^* D$ because it is inductive. In particular, $D' = t b e b e b$ and the reachable configuration $t b h b e b$. Hence, one clause of I contains $h_{3,6}$. Consider the following pair of configurations: $D'' = t f h f t f$

and $C = t b h f t f$. I must separate D'' from C since $D'' \rightsquigarrow^* D$ while $C \in Reach$. Since $D'' \models h_{3.6}$ this separation is based on the second clause of I which must contain $b_{2.6}$. This means $t b h f e b \models I$. Since I is inductive and $t b h f e b \rightsquigarrow t b e b e b \rightsquigarrow t f t f e b \rightsquigarrow h b t f e b = D$ the assumption that I exists is folly. Consequently, D cannot be excluded via inductive 2-invariants.

B $IndInv_2$ for cache coherence protocols Berkeley and Dragon

For both protocols we follow the specification of [16].

Berkeley

In the Berkeley cache coherence protocol, each cell is in one of four different states: invalid (i), unowned (u), exclusive (e), and shared (s). Initially, all cells are invalid. Consequently, the language of initial configurations is i^* . For the transitions we consider a few different events. The first one is that the memory is read and the corresponding cell does provide some value of it; i.e., the cell is *not* in the state i . In this case nothing changes:

$$\left(\begin{array}{c} [i] \\ [i] \end{array} \middle| \begin{array}{c} [u] \\ [u] \end{array} \middle| \begin{array}{c} [e] \\ [e] \end{array} \middle| \begin{array}{c} [s] \\ [s] \end{array} \right)^* \left(\begin{array}{c} [u] \\ [u] \end{array} \middle| \begin{array}{c} [e] \\ [e] \end{array} \middle| \begin{array}{c} [s] \\ [s] \end{array} \right) \left(\begin{array}{c} [i] \\ [i] \end{array} \middle| \begin{array}{c} [u] \\ [u] \end{array} \middle| \begin{array}{c} [e] \\ [e] \end{array} \middle| \begin{array}{c} [s] \\ [s] \end{array} \right)^*$$

If, on the other hand, a value is read from some cell that is in state i , then this memory cell fetches the information without claiming ownership; i.e., moves into the state u . Every other memory cell observes this process. Thus, cells that previously were in e move to s to account for the fact that another memory cell holds the same information.

$$\left(\begin{array}{c} [i] \\ [i] \end{array} \middle| \begin{array}{c} [u] \\ [u] \end{array} \middle| \begin{array}{c} [e] \\ [s] \end{array} \middle| \begin{array}{c} [s] \\ [s] \end{array} \right)^* \left(\begin{array}{c} [i] \\ [u] \end{array} \right) \left(\begin{array}{c} [i] \\ [i] \end{array} \middle| \begin{array}{c} [u] \\ [u] \end{array} \middle| \begin{array}{c} [e] \\ [s] \end{array} \middle| \begin{array}{c} [s] \\ [s] \end{array} \right)^*$$

If a value is written to a cell that was invalid before, then this cell claims exclusive ownership; that is, all other cells are invalidated.

$$\left(\begin{array}{c} [i] \\ [i] \end{array} \middle| \begin{array}{c} [u] \\ [i] \end{array} \middle| \begin{array}{c} [e] \\ [i] \end{array} \middle| \begin{array}{c} [s] \\ [i] \end{array} \right)^* \left(\begin{array}{c} [i] \\ [e] \end{array} \right) \left(\begin{array}{c} [i] \\ [i] \end{array} \middle| \begin{array}{c} [u] \\ [i] \end{array} \middle| \begin{array}{c} [e] \\ [i] \end{array} \middle| \begin{array}{c} [s] \\ [i] \end{array} \right)^*$$

If a cell already has exclusive ownership of this information there is nothing to be done. If the cell has only shared ownership of the value, all other cells *that claim shared ownership* are invalidated.

$$\left(\begin{array}{c} [i] \\ [i] \end{array} \middle| \begin{array}{c} [u] \\ [i] \end{array} \middle| \begin{array}{c} [e] \\ [e] \end{array} \middle| \begin{array}{c} [s] \\ [i] \end{array} \right)^* \left(\begin{array}{c} [u] \\ [e] \end{array} \middle| \begin{array}{c} [s] \\ [e] \end{array} \right) \left(\begin{array}{c} [i] \\ [i] \end{array} \middle| \begin{array}{c} [u] \\ [i] \end{array} \middle| \begin{array}{c} [e] \\ [e] \end{array} \middle| \begin{array}{c} [s] \\ [i] \end{array} \right)^*$$

Finally, the cache can decide to drop data at any moment in time. Thus, any cell might move into the state i .

$$\left(\begin{array}{c} [i] \\ [i] \end{array} \middle| \begin{array}{c} [u] \\ [u] \end{array} \middle| \begin{array}{c} [e] \\ [e] \end{array} \middle| \begin{array}{c} [s] \\ [s] \end{array} \right)^* \left(\begin{array}{c} [u] \\ [i] \end{array} \middle| \begin{array}{c} [e] \\ [i] \end{array} \middle| \begin{array}{c} [s] \\ [i] \end{array} \right) \left(\begin{array}{c} [i] \\ [i] \end{array} \middle| \begin{array}{c} [u] \\ [u] \end{array} \middle| \begin{array}{c} [e] \\ [e] \end{array} \middle| \begin{array}{c} [s] \\ [s] \end{array} \right)^*$$

We pose now the question whether a configuration can be reached where two different cells claiming exclusive access to some data. The corresponding set \mathcal{U} corresponds to $\Sigma^* e \Sigma^* e \Sigma^*$. As shown in Table 1, $IndInv_1$ does not prove this property. Let us see why. Assume there is an inductive 1-invariant I which invalidates the bad word $b = e e$; that is, $b \not\models I$. Observe

now that we can reach b in one step from $b' = u e$, and $b'' = e u$. Consequently, I cannot be satisfied by b' or b'' either. Otherwise, since I is inductive, we already get $b \models I$. This means, I must not contain $e_{1:2}$, $e_{2:2}$, $u_{1:2}$, or $u_{2:2}$. This, however, makes I unsatisfiable for the actually reachable configuration $u u$.

Using an adapted version of the semi-automatic approach of [20] and some additional reasoning led us to the following language of inductive 2-invariants which exclude all configurations from \mathcal{U} :

$$\begin{bmatrix} \emptyset \\ \emptyset \end{bmatrix}^* \begin{bmatrix} \{i, s, u\} \\ \{i\} \end{bmatrix} \begin{bmatrix} \emptyset \\ \emptyset \end{bmatrix}^* \begin{bmatrix} \{i\} \\ \{i, s, u\} \end{bmatrix} \begin{bmatrix} \emptyset \\ \emptyset \end{bmatrix}^* .$$

Since $IndInv_2$ is the strongest inductive 2-invariant, this shows that $IndInv_2$ is strong enough to prove the property.

Dragon

The Dragon protocol distinguishes five states. As before, we have states for invalid cells (i), cells that maintain an exclusive copy of the data (e) and cells that have a (potentially) shared copy of the data (s). In contrast to before, the Dragon protocol does not invalidate other copies of some data when it is updated. Instead we introduce two new states which mirror e and s but, additionally, indicate that the data might have changed. We refer to these states as \hat{e} and \hat{s} respectively. Regardless, we initialize all cells as invalid; i.e., we have the initial language i^* .

Assume a read from a “valid” cell; that is, some cell that is not in state i . In that case, nothing changes:

$$\left(\begin{bmatrix} i \\ i \end{bmatrix} \begin{bmatrix} e \\ e \end{bmatrix} \begin{bmatrix} s \\ s \end{bmatrix} \begin{bmatrix} \hat{e} \\ \hat{e} \end{bmatrix} \begin{bmatrix} \hat{s} \\ \hat{s} \end{bmatrix} \right)^* \left(\begin{bmatrix} e \\ e \end{bmatrix} \begin{bmatrix} s \\ s \end{bmatrix} \begin{bmatrix} \hat{e} \\ \hat{e} \end{bmatrix} \begin{bmatrix} \hat{s} \\ \hat{s} \end{bmatrix} \right) \left(\begin{bmatrix} i \\ i \end{bmatrix} \begin{bmatrix} e \\ e \end{bmatrix} \begin{bmatrix} s \\ s \end{bmatrix} \begin{bmatrix} \hat{e} \\ \hat{e} \end{bmatrix} \begin{bmatrix} \hat{s} \\ \hat{s} \end{bmatrix} \right)^* .$$

If a read occurs from an invalid cell – while all cells are invalid – the accessed cell becomes an exclusive reference:

$$\begin{bmatrix} i \\ i \end{bmatrix}^* \begin{bmatrix} i \\ e \end{bmatrix} \begin{bmatrix} i \\ i \end{bmatrix}^* .$$

If not all cells are invalid but a read occurs for an invalid cell then this cell obtains a shared copy to the data. Moreover, all exclusive references; i.e., cells in states e or \hat{e} , move to their shared counterparts (s and \hat{s} respectively).

$$\left(\begin{bmatrix} i \\ i \end{bmatrix} \begin{bmatrix} e \\ s \end{bmatrix} \begin{bmatrix} s \\ s \end{bmatrix} \begin{bmatrix} \hat{e} \\ \hat{s} \end{bmatrix} \begin{bmatrix} \hat{s} \\ \hat{s} \end{bmatrix} \right)^* \begin{bmatrix} i \\ s \end{bmatrix} \left(\begin{bmatrix} i \\ i \end{bmatrix} \begin{bmatrix} e \\ s \end{bmatrix} \begin{bmatrix} s \\ s \end{bmatrix} \begin{bmatrix} \hat{e} \\ \hat{s} \end{bmatrix} \begin{bmatrix} \hat{s} \\ \hat{s} \end{bmatrix} \right)^* .$$

Writing a cell in state \hat{e} does not change anything. On the other hand, writing a cell in state e moves that cell into the state \hat{e} :

$$\left(\begin{bmatrix} i \\ i \end{bmatrix} \begin{bmatrix} e \\ e \end{bmatrix} \begin{bmatrix} s \\ s \end{bmatrix} \begin{bmatrix} \hat{e} \\ \hat{e} \end{bmatrix} \begin{bmatrix} \hat{s} \\ \hat{s} \end{bmatrix} \right)^* \begin{bmatrix} \hat{e} \\ \hat{e} \end{bmatrix} \left(\begin{bmatrix} i \\ i \end{bmatrix} \begin{bmatrix} e \\ e \end{bmatrix} \begin{bmatrix} s \\ s \end{bmatrix} \begin{bmatrix} \hat{e} \\ \hat{e} \end{bmatrix} \begin{bmatrix} \hat{s} \\ \hat{s} \end{bmatrix} \right)^*$$

and

$$\left(\begin{bmatrix} i \\ i \end{bmatrix} \begin{bmatrix} e \\ e \end{bmatrix} \begin{bmatrix} s \\ s \end{bmatrix} \begin{bmatrix} \hat{e} \\ \hat{e} \end{bmatrix} \begin{bmatrix} \hat{s} \\ \hat{s} \end{bmatrix} \right)^* \begin{bmatrix} e \\ \hat{e} \end{bmatrix} \left(\begin{bmatrix} i \\ i \end{bmatrix} \begin{bmatrix} e \\ e \end{bmatrix} \begin{bmatrix} s \\ s \end{bmatrix} \begin{bmatrix} \hat{e} \\ \hat{e} \end{bmatrix} \begin{bmatrix} \hat{s} \\ \hat{s} \end{bmatrix} \right)^* .$$

A write-operation on a cell that is the only one in state s or \hat{s} results in a change to \hat{e} . If there are other cells in either state, one moves two \hat{s} while all others move to s .

$$\left(\begin{array}{c|c|c} [i] & [e] & [\hat{e}] \\ \hline [i] & [e] & [\hat{e}] \end{array} \right)^* \begin{array}{c} [\hat{s}] \\ [\hat{e}] \end{array} \left(\begin{array}{c|c|c} [i] & [e] & [\hat{e}] \\ \hline [i] & [e] & [\hat{e}] \end{array} \right)^*,$$

$$\left(\begin{array}{c|c|c} [i] & [e] & [\hat{e}] \\ \hline [i] & [e] & [\hat{e}] \end{array} \right)^* \begin{array}{c} [s] \\ [\hat{e}] \end{array} \left(\begin{array}{c|c|c} [i] & [e] & [\hat{e}] \\ \hline [i] & [e] & [\hat{e}] \end{array} \right)^*$$

and

$$\left(\begin{array}{c|c|c|c|c} [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \\ \hline [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \end{array} \right)^* \left(\begin{array}{c|c} [s] & [\hat{s}] \\ \hline [s] & [\hat{s}] \end{array} \right) \left(\begin{array}{c|c|c|c|c} [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \\ \hline [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \end{array} \right)^* \left(\begin{array}{c|c} [s] & [\hat{s}] \\ \hline [s] & [\hat{s}] \end{array} \right) \left(\begin{array}{c|c|c|c|c} [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \\ \hline [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \end{array} \right)^*$$

$$\left(\begin{array}{c|c|c|c|c} [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \\ \hline [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \end{array} \right)^* \left(\begin{array}{c|c} [s] & [\hat{s}] \\ \hline [s] & [\hat{s}] \end{array} \right) \left(\begin{array}{c|c|c|c|c} [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \\ \hline [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \end{array} \right)^* \left(\begin{array}{c|c} [s] & [\hat{s}] \\ \hline [s] & [\hat{s}] \end{array} \right) \left(\begin{array}{c|c|c|c|c} [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \\ \hline [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \end{array} \right)^*.$$

If a value is written to a previously invalid cell, then either this cell moves to \hat{e} (assuming all other cells are i as well), while the occurrence of another cell with this value causes the written cell to become \hat{s} and all other cells to move to state s .

$$\begin{array}{c} [i]^* \\ [i] \end{array} \begin{array}{c} [i] \\ [\hat{e}] \end{array} \begin{array}{c} [i]^* \\ [i] \end{array}$$

and

$$\left(\begin{array}{c|c|c|c|c} [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \\ \hline [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \end{array} \right)^* \begin{array}{c} [i] \\ [\hat{s}] \end{array} \left(\begin{array}{c|c|c|c|c} [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \\ \hline [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \end{array} \right)^* \left(\begin{array}{c|c} [\hat{e}] & [s] \\ \hline [s] & [\hat{s}] \end{array} \right) \left(\begin{array}{c|c|c|c|c} [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \\ \hline [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \end{array} \right)^*$$

$$\left(\begin{array}{c|c|c|c|c} [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \\ \hline [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \end{array} \right)^* \left(\begin{array}{c|c} [\hat{e}] & [s] \\ \hline [s] & [\hat{s}] \end{array} \right) \left(\begin{array}{c|c|c|c|c} [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \\ \hline [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \end{array} \right)^* \begin{array}{c} [i] \\ [\hat{s}] \end{array} \left(\begin{array}{c|c|c|c|c} [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \\ \hline [i] & [e] & [\hat{e}] & [s] & [\hat{s}] \end{array} \right)^*.$$

Finally, any cell might drop its content at any point.

$$\left(\begin{array}{c|c|c|c|c} [i] & [e] & [s] & [\hat{e}] & [\hat{s}] \\ \hline [i] & [e] & [s] & [\hat{e}] & [\hat{s}] \end{array} \right)^* \left(\begin{array}{c|c|c|c} [e] & [\hat{e}] & [\hat{s}] & [s] \\ \hline [i] & [i] & [i] & [i] \end{array} \right) \left(\begin{array}{c|c|c|c|c} [i] & [e] & [s] & [\hat{e}] & [\hat{s}] \\ \hline [i] & [e] & [s] & [\hat{e}] & [\hat{s}] \end{array} \right)^*$$

We are interested now to establish that the language $\Sigma^* \hat{e} \Sigma^* \hat{e} \Sigma^*$ cannot be reached. The proof that $IndInv_1$ is insufficient to exclude all configurations of $\Sigma^* \hat{e} \Sigma^* \hat{e} \Sigma^*$ is straightforward: Observe that both $s \hat{e}$ and $\hat{e} s$ can reach $\hat{e} \hat{e}$ in one step. In consequence, analogously to the argument used for the Berkeley protocol, any inductive 1-invariant cannot distinguish between the reachable $s s$ and the unreachable $\hat{e} \hat{e}$.

On the other hand, the language

$$\left[\emptyset \right]^* \begin{array}{c} \{\hat{s}, i, s\} \\ \{i\} \end{array} \left[\emptyset \right]^* \begin{array}{c} \{i\} \\ \{\hat{s}, i, s\} \end{array} \left[\emptyset \right]^*$$

of inductive 2-invariants induces an abstraction disjoint from $\Sigma^* \hat{e} \Sigma^* \hat{e} \Sigma^*$. Consequently, $IndInv_2$ does as well.

On an Invariance Problem for Parameterized Concurrent Systems

Marius Bozga

Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000, France

Lucas Bueri

Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000, France

Radu Iosif

Univ. Grenoble Alpes, CNRS, Grenoble INP, VERIMAG, 38000, France

Abstract

We consider concurrent systems consisting of replicated finite-state processes that synchronize via joint interactions in a network with user-defined topology. The system is specified using a resource logic with a multiplicative connective and inductively defined predicates, reminiscent of Separation Logic [19]. The problem we consider is if a given formula in this logic defines an invariant, namely whether any model of the formula, following an arbitrary firing sequence of interactions, is transformed into another model of the same formula. This property, called *havoc invariance*, is quintessential in proving the correctness of reconfiguration programs that change the structure of the network at runtime. We show that the havoc invariance problem is many-one reducible to the entailment problem $\phi \models \psi$, asking if any model of ϕ is also a model of ψ . Although, in general, havoc invariance is found to be undecidable, this reduction allows to prove that havoc invariance is in 2EXP, for a general fragment of the logic, with a 2EXP entailment problem.

2012 ACM Subject Classification Software and its engineering \rightarrow Formal software verification

Keywords and phrases parameterized verification, invariant checking, resource logics, reconfigurable systems, tree automata

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.24

Related Version *Full Version*: <https://arxiv.org/abs/2204.12117>

1 Introduction

The parameterized verification problem asks to decide whether a system consisting of an arbitrary number of finite-state processes that communicate via synchronized (joint) actions satisfies a specification, such as deadlock freedom, mutual exclusion or a temporal logic property e.g., every request is eventually answered. The literature in this area has a wealth of decidability and complexity results (see [3] for a survey) classified according to the communication type (e.g., rendez-vous, broadcast) and the network topology e.g., rings where every process interacts with its left/right neighbours, cliques where each two process may interact, stars with a controller interacting with unboundedly many workers, etc.

As modern computing systems are dynamically adaptive, recent effort has been put into designing *reconfigurable systems*, whose network topologies change at runtime (see [12] for a survey) in order to address maintenance (e.g., replacement of faulty and obsolete components by new ones, firmware updates, etc.) and internal traffic issues (e.g., re-routing to avoid congestion in a datacenter [18]). Unfortunately the verification of dynamic reconfigurable systems (i.e., proving the absence of design errors) remains largely unexplored. Consequently, such systems are prone to bugs that may result in e.g., denial of services or data corruption¹.

¹ Google reports on a cascading cloud failure due to reconfiguration: <https://status.cloud.google.com/incident/appengine/19007>.



Proving correctness of parameterized reconfigurable networks is tackled in [1], where a Hoare-style program logic is proposed to write proofs of reconfiguration programs i.e., programs that dynamically add and remove processes and interactions from the network during runtime. The assertion language used by these proofs is a logic that describes sets of configurations defining the network topology and the local states of the processes. The logic views processes and interactions as resources that can be joined via a *separating conjunction*, in the spirit of Separation Logic [19]. The separating conjunction supports *local reasoning*, which is the ability of describing reconfigurations *only with respect to those components and interactions that are involved in the mutation*, while disregarding the rest of the system's configuration. Moreover, the separating conjunction allows to concisely describe networks of unbounded size, that share a similar architectural style (e.g., pipelines, rings, stars, trees) by means of inductively defined predicates.

Due to the interleaving of reconfigurations and interactions between components, the annotations of the reconfiguration program form a valid proof under so-called *havoc invariance* assumptions, stating global properties about the configurations, that remain, moreover, *unchanged under the ongoing interactions in the system*. These assumptions are needed to apply the sequential composition rule that infers a Hoare triple $\{\phi\} P; Q \{\psi\}$ from two premisses $\{\phi\} P \{\theta\}$ and $\{\theta\} Q \{\psi\}$, where P and Q are reconfiguration actions that add and/or remove processes and communication channels. Essentially, because the states of the processes described by the intermediate assertion θ might change between the end of P and the beginning of Q , this rule is sound provided that θ is a havoc invariant formula.

This paper contributes to the automated generation of reconfiguration proofs, by a giving a procedure that discharges the havoc invariance side conditions. The challenge is that a formula of the configuration logic (that contains inductively defined predicates) describes an infinite set of configurations of arbitrary sizes. The main result is that the havoc invariance problem is effectively many-one reducible to the entailment problem $\phi \models \psi$, that asks if every model of a formula ϕ is a model of another formula ψ . Here ψ is the formula whose havoc invariance is being checked and ϕ defines the set of configurations γ' obtained from a model γ of ψ , by executing one interaction from γ . The reduction is polynomial if certain parameters are bounded by a constant (i.e., the arity of the predicates, the size of interactions and the number of predicate atoms is an inductive rule), providing a 2EXP upper bound for a fragment of the logic with a decidable (2EXP) entailment problem [4, §6]. Having a polynomial reduction motivates, moreover, future work on the definition of fragments of lower (e.g., polynomial) entailment complexity (see e.g., [9] for a fragment of Separation Logic with a polynomial entailment problem), that are likely to yield efficient decision procedures for the havoc invariance problem as well. In addition, we provide a 2EXP-hard lower bound for the havoc invariance problem in this fragment of the logic (i.e., assuming predicates of unbounded arity) and show that havoc invariance is undecidable, when unrestricted formulæ are considered as input.

Related Work. Specifying parameterized concurrent systems by inductive definitions is reminiscent of *network grammars* [20, 16, 13], that use inductive rules to describe systems with linear (pipeline, token-ring) architectures obtained by composition of an unbounded number of processes. In contrast, we use predicates of unrestricted arities to describe network topologies that can be, in general, more complex than trees. Moreover, we write inductive definitions using a resource logic, suitable also for writing Hoare logic proofs of reconfiguration programs, based on local reasoning [8].

Verification of network grammars against safety properties (unreachability of error configurations) requires the synthesis of *network invariants* [21], computed by rather costly fixpoint iterations [17] or by abstracting (forgetting the particular values of indices in) the composition of a small bounded number of instances [14]. In previous work, we have developed an invariant synthesis method based on *structural invariants*, that are synthesized with little computational effort and prove to be efficient in many practical examples [5, 6].

The havoc invariance problem considered in this paper is, however, different from safety checking and has not been addressed before, to the best of our knowledge. An explanation is that verification of reconfigurable systems has received fairly scant attention, relying mostly on runtime verification [7, 10, 15, 11], instead of deductive verification, reported in [1]. In [1] we addressed havoc invariance with a set of inference rules used to write proofs manually, whereas the goal of this paper is to discharge such conditions automatically.

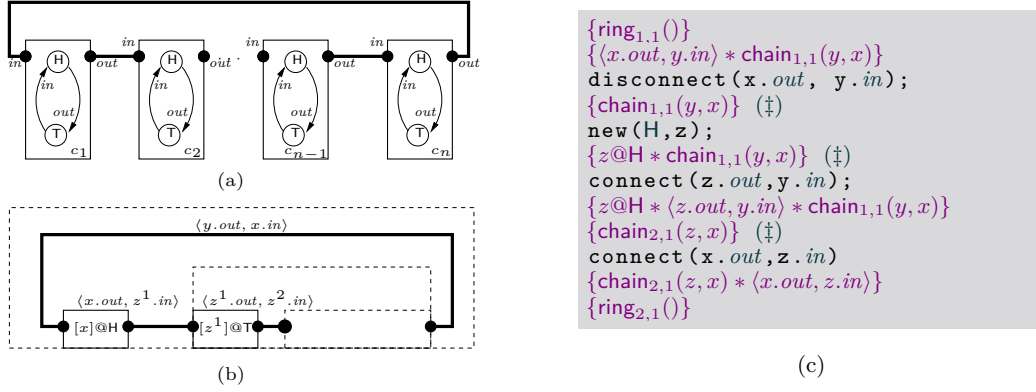
1.1 A Motivating Example

Consider, for instance, a system consisting of a finite but unbounded number of processes, called *components* in the following. The components execute the same machine with states T and H, denoting whether the component has a token (T) or a hole (H). The components are placed in a ring, each component having exactly one left and one right neighbour, as in Fig. 1 (a). A component without a token may receive one, by executing a transition $H \xrightarrow{in} T$, simultaneously with its left neighbour, that executes the transition $T \xrightarrow{out} H$, as in Fig. 1 (a). Note that there can be more than one token, moving independently in the system, such that no token overtakes another token. The configurations of the token ring system are described by the following inductive rules:

$$\begin{aligned} \text{ring}_{h,t}() &\leftarrow \exists x \exists y . \langle x.out, y.in \rangle * \text{chain}_{h,t}(y, x) \\ \text{chain}_{h,t}(x, y) &\leftarrow \exists z . [x]@q * \langle x.out, z.in \rangle * \text{chain}_{h',t'}(z, y), \text{ for both } q \in \{H, T\} \\ \text{chain}_{0,1}(x, x) &\leftarrow [x]@T \quad \text{chain}_{1,0}(x, x) \leftarrow [x]@H \quad \text{chain}_{0,0}(x, x) \leftarrow [x] \\ \text{where } h' &\stackrel{\text{def}}{=} \begin{cases} \max(h-1, 0) & , \text{ if } q = H \\ h & , \text{ if } q = T \end{cases} \quad \text{and } t' \stackrel{\text{def}}{=} \begin{cases} \max(t-1, 0) & , \text{ if } q = T \\ t & , \text{ if } q = H \end{cases} \end{aligned}$$

The predicate $\text{ring}_{h,t}()$ describes a ring with at least h (resp. t) components in state H (resp. T). The ring consists of an interaction between the ports *out* and *in* of two components x and y , respectively, described by $\langle x.out, y.in \rangle$ and a separate chain of components between x and y , described by $\text{chain}_{h,t}(y, x)$. Inductively, a chain consists of a component $[x]@q$ in state $q \in \{H, T\}$, an interaction $\langle x.out, z.in \rangle$ and a separate $\text{chain}_{h',t'}(z, y)$, where h' and t' are the least numbers of components in state H and T, respectively, after the removal of the component x . Fig. 1 (b) depicts the unfolding of the inductive definition of $\text{ring}_{h,t}()$ with the existentially quantified variables z from the above rules α -renamed to z^1, z^2 , etc.

A *reconfiguration action* is an atomic creation or deletion of a component or interaction. A *reconfiguration sequence* is a finite sequence of reconfiguration actions that takes as input a mapping of program variables to components and executes the actions from the sequence, in interleaving with the interactions in the system. For instance, the reconfiguration sequence from Fig. 1 (c) takes as input the mapping of x and y to two adjacent components in the token ring, removes the interaction $\langle x.out, y.in \rangle$ by executing $\text{disconnect}(x.out, y.in)$ and creates a new component in state H (by executing $\text{new}(x, H)$) that is connected in between x and y via two new interactions created by executing $\text{connect}(z.out, y.in)$ and $\text{connect}(x.out, z.in)$, respectively. Fig. 1 (c) shows a proof (with annotations in curly braces) of the fact that the outcome of the reconfiguration of a ring of components is a ring whose least number of components in state H is increased from one to two. This proof is split into several subgoals:



■ **Figure 1** Inductive Specification and Reconfiguration of a Token Ring.

1. Entailments required to apply the consequence rule of Hoare logic e.g., $\text{ring}_{1,1}() \models \exists x \exists y . \langle x.out, y.in \rangle * \text{chain}_{1,1}(y, x)$. The entailment problem has been addressed in [4, §6], with the definition of a general fragment of the configuration logic, for which the entailment problem is decidable in double exponential time.
2. Hoare triples that describe the effect of the atomic reconfiguration actions e.g., $\{\langle x.out, y.in \rangle * \text{chain}_{1,1}(y, x)\} \text{disconnect}(x.out, y.in) \{\text{chain}_{1,1}(y, x)\}$. These are obtained by applying the frame rule to the local² specifications of the atomic actions. The local specification of reconfiguration actions and the frame rule for local actions are described in [1, §4.2].
3. *Havoc invariance* proofs for the annotations marked with (‡) in Fig. 1 (c). For instance, the formula $\text{chain}_{1,1}(y, x)$ is havoc invariant because the interactions in a chain of components will only move tokens to the right without creating more or losing any, hence there will be the same number of components in state H (T) no matter which interactions are fired.

2 Definitions

We denote by \mathbb{N} the set of positive integers, including zero. For a set A , we denote $A^1 \stackrel{\text{def}}{=} A$, $A^{i+1} \stackrel{\text{def}}{=} A^i \times A$, for all $i \geq 0$, where \times denotes the Cartesian product, and $A^+ \stackrel{\text{def}}{=} \bigcup_{i \geq 1} A^i$. The cardinality of a finite set A is denoted by $\|A\|$. By writing $A \subseteq_{\text{fin}} B$ we mean that A is a finite subset of B . Given integers i and j , we write $[i, j]$ for the set $\{i, i+1, \dots, j\}$, assumed to be empty if $i > j$. For a function $f : A \rightarrow B$, we denote by $f[a_i \leftarrow b_i]_{i \in [1, n]}$ the function that maps a_i into b_i for each $i \in [1, n]$ and agrees with f everywhere else.

2.1 Configurations

We model a parallel system as a hypergraph, whose vertices are *components* (i.e., the nodes of the network) and hyperedges are *interactions* (i.e., describing the way the components communicate with each other). The components are taken from a countably infinite set \mathbb{C} , called the *universe*. We consider that each component executes its own copy of the same *behavior*, represented as a finite-state machine $\mathbb{B} = (\mathcal{P}, \mathcal{Q}, \rightarrow)$, where \mathcal{P} is a finite set of

² A Hoare triple $\{\phi\} P \{\psi\}$ is *local* if it mentions only those components and interactions added or deleted by P . Local specifications are plugged into a global context by the *frame rule* that infers $\{\phi * F\} P \{\psi * F\}$ from $\{\phi\} P \{\psi\}$ if the variables modified by P are not free in F .

ports, \mathcal{Q} is a finite set of states and $\rightarrow \subseteq \mathcal{Q} \times \mathcal{P} \times \mathcal{Q}$ is a transition relation. Intuitively, each transition $q \xrightarrow{p} q'$ of the behavior \mathbb{B} is triggered by a visible event, represented by the port p . The universe \mathbb{C} and the behavior $\mathbb{B} = (\mathcal{P}, \mathcal{Q}, \rightarrow)$ are considered to be fixed in the following.

A *configuration* is a snapshot of the system, describing the topology of the network (i.e., the set of present components and interactions) together with the local state of each component, formally defined below (see also [4]):

- **Definition 1.** A configuration is a tuple $\gamma = (\mathcal{C}, \mathcal{I}, \varrho)$, where:
- $\mathcal{C} \subseteq_{fin} \mathbb{C}$ is a finite set of components, that are present in the configuration,
 - $\mathcal{I} \subseteq_{fin} (\mathbb{C} \times \mathcal{P})^+$ is a finite set of interactions, where each interaction is a sequence $(c_i, p_i)_{i \in [1, n]} \in (\mathbb{C} \times \mathcal{P})^n$ that binds together the ports p_1, \dots, p_n of the pairwise distinct components c_1, \dots, c_n , respectively. The ordered sequence of ports (p_1, \dots, p_n) is called an interaction type and we denote by \mathcal{P}^+ the set of interaction types.
 - $\varrho : \mathbb{C} \rightarrow \mathcal{Q}$ is a state map associating each (possibly absent) component, a state of the behavior \mathbb{B} , such that the set $\{c \in \mathbb{C} \mid \varrho(c) = q\}$ is infinite, for each $q \in \mathcal{Q}$.

We denote by Γ the set of configurations.

The last condition requires that there is an infinite pool of components in each state $q \in \mathcal{Q}$; since \mathbb{C} is infinite and \mathcal{Q} is finite, this condition is feasible.

- **Example 2.** The configurations of the system from Fig. 1 (a) are $(\{c_1, \dots, c_n\}, \{(c_i, out, c_{(i \bmod n)+1}, in) \mid i \in [1, n]\}, \varrho)$, where $\varrho : \mathbb{C} \rightarrow \{\mathbf{H}, \mathbf{T}\}$ is a state map. The ring topology is given by components $\{c_1, \dots, c_n\}$ and interactions $\{(c_i, out, c_{(i \bmod n)+1}, in) \mid i \in [1, n]\}$. ◀

Note that Def. 1 allows configurations with interactions that involve absent components i.e., not from the set \mathcal{C} of present components in the given configuration. The following definition distinguishes such configurations:

- **Definition 3.** A configuration $\gamma = (\mathcal{C}, \mathcal{I}, \varrho)$ is said to be tight if and only if for any interaction $(c_i, p_i)_{i \in [1, n]} \in \mathcal{I}$ we have $\{c_i \mid i \in [1, n]\} \subseteq \mathcal{C}$ and loose otherwise.

For instance, every configuration of the system from Fig. 1 (a) is tight and becomes loose if a component is deleted.

2.2 Configuration Logic

Let \mathbb{V} and \mathbb{A} be countably infinite sets of *variables* and *predicates*, respectively. For each predicate $A \in \mathbb{A}$, we denote its arity by $\#A$. The formulæ of the *Configuration Logic* (CL) are described inductively by the following syntax:

$$\phi := \text{emp} \mid [x] \mid \langle x_1.p_1, \dots, x_n.p_n \rangle \mid x@q \mid x = y \mid x \neq y \mid A(x_1, \dots, x_{\#A}) \mid \phi * \phi \mid \exists x . \phi$$

where $x, y, x_1, \dots \in \mathbb{V}$, $q \in \mathcal{Q}$ and $A \in \mathbb{A}$. A formula $[x]$, $\langle x_1.p_1, \dots, x_n.p_n \rangle$, $x@q$ and $A(x_1, \dots, x_{\#A})$ is called a *component*, *interaction*, *state* and *predicate* atom, respectively. We use the shorthand $[x]@q \stackrel{\text{def}}{=} [x] * x@q$. Intuitively, a formula $[x]@q * [y]@q' * \langle x.out, y.in \rangle * \langle x.in, y.out \rangle$ describes a configuration consisting of two distinct components, denoted by the values of x and y , in states q and q' , respectively, and two interactions binding the *out* port of one to the *in* port of the other component.

A formula with no occurrences of predicate atoms (resp. existential quantifiers) is called *predicate-free* (resp. *quantifier-free*). A qpf formula is both predicate- and quantifier-free. A variable is *free* if it does not occur in the scope of a quantifier and $\text{fv}(\phi)$ is the set of free variables of ϕ . A *substitution* $\phi[x_i/y_i]_{i \in [1, n]}$ replaces simultaneously every free occurrence of x_i by y_i in ϕ , for all $i \in [1, n]$. The size of a formula ϕ is the total number of occurrences of symbols needed to write it down, denoted by $\text{size}(\phi)$.

The only connective of the logic is the *separating conjunction* $*$. Intuitively, $\phi_1 * \phi_2$ means that ϕ_1 and ϕ_2 hold separately, on disjoint parts of the same configuration. Its formal meaning is coined by the following definition of *composition* of configurations:

► **Definition 4.** *The composition of two configurations $\gamma_i = (\mathcal{C}_i, \mathcal{I}_i, \varrho)$, for $i = 1, 2$, such that $\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset$ and $\mathcal{I}_1 \cap \mathcal{I}_2 = \emptyset$, is defined as $\gamma_1 \bullet \gamma_2 \stackrel{\text{def}}{=} (\mathcal{C}_1 \cup \mathcal{C}_2, \mathcal{I}_1 \cup \mathcal{I}_2, \varrho)$. The composition $\gamma_1 \bullet \gamma_2$ is undefined if $\mathcal{C}_1 \cap \mathcal{C}_2 \neq \emptyset$ or $\mathcal{I}_1 \cap \mathcal{I}_2 \neq \emptyset$.*

► **Example 5.** Let $\gamma_i = (\{c_i\}, \{(c_i, \text{out}, c_{3-i}, \text{in})\}, \varrho)$ be configurations, for $i = 1, 2$. Then $\gamma_1 \bullet \gamma_2 = (\{c_1, c_2\}, \{(c_1, \text{out}, c_2, \text{in}), (c_2, \text{out}, c_1, \text{in})\}, \varrho)$. \lrcorner

The meaning of the predicates is given by a set of inductive definitions:

► **Definition 6.** *A set of inductive definitions (SID) Δ consists of rules of the form $A(x_1, \dots, x_{\#A}) \leftarrow \phi$, where $x_1, \dots, x_{\#A}$ are pairwise distinct variables, called parameters, such that $\text{fv}(\phi) \subseteq \{x_1, \dots, x_{\#A}\}$. We say that the rule $A(x_1, \dots, x_{\#A}) \leftarrow \phi$ defines A and denote by $\text{def}_\Delta(A)$ the set of rules from Δ that define A and by $\text{Def}(\Delta) \stackrel{\text{def}}{=} \{A \mid \text{def}_\Delta(A) \neq \emptyset\}$ the set of predicates defined by Δ .*

Note that having distinct parameters in a rule is without loss of generality, as e.g., a rule $A(x_1, x_1) \leftarrow \phi$ can be equivalently written as $A(x_1, x_2) \leftarrow x_1 = x_2 * \phi$. As a convention, we shall always use the names $x_1, \dots, x_{\#A}$ for the parameters of a rule that defines A . An example of a SID is given in §1.1.

The size of a SID is $\text{size}(\Delta) \stackrel{\text{def}}{=} \sum_{A(x_1, \dots, x_{\#A}) \leftarrow \phi \in \Delta} \text{size}(\phi) + \#A + 1$. Other parameters, relevant for complexity evaluation, are the maximal

- (1) arity $\#\Delta \stackrel{\text{def}}{=} \max\{\#A \mid A(x_1, \dots, x_{\#A}) \leftarrow \phi \in \Delta\}$ of a defined predicate,
- (2) size of an interaction type $N(\Delta) \stackrel{\text{def}}{=} \max\{n \mid \langle y_1.p_1, \dots, y_n.p_n \rangle \text{ occurs in } \Delta\}$, and
- (3) number of predicate atoms $H(\Delta) \stackrel{\text{def}}{=} \max\{h \mid A(x_1, \dots, x_{\#A}) \leftarrow \exists y_1 \dots \exists y_m . \phi * \bigstar_{\ell=1}^h B_\ell(\mathbf{z}_\ell), \phi \text{ is a qpf formula}\}$.

The semantics of CL formulæ is defined by a satisfaction relation $\gamma \models_\Delta^\nu \phi$ between configurations and formulæ. This relation is parameterized by a *store* $\nu : \mathbb{V} \rightarrow \mathbb{C}$ mapping the free variables of a formula into components from the universe (possibly absent from γ) and an SID Δ . The definition of the satisfaction relation is by induction on the structure of formulæ, where $\gamma = (\mathcal{C}, \mathcal{I}, \varrho)$ is a configuration (Def. 1):

$\gamma \models_\Delta^\nu \text{emp}$	\iff	$\mathcal{C} = \emptyset$ and $\mathcal{I} = \emptyset$
$\gamma \models_\Delta^\nu [x]$	\iff	$\mathcal{C} = \{\nu(x)\}$ and $\mathcal{I} = \emptyset$
$\gamma \models_\Delta^\nu \langle x_1.p_1, \dots, x_n.p_n \rangle$	\iff	$\mathcal{C} = \emptyset$ and $\mathcal{I} = \{(\nu(x_1), p_1), \dots, (\nu(x_n), p_n)\}$
$\gamma \models_\Delta^\nu x @ q$	\iff	$\gamma \models_\Delta^\nu \text{emp}$ and $\varrho(\nu(x)) = q$
$\gamma \models_\Delta^\nu x \sim y$	\iff	$\gamma \models_\Delta^\nu \text{emp}$ and $\nu(x) \sim \nu(y)$, for all $\sim \in \{=, \neq\}$
$\gamma \models_\Delta^\nu A(y_1, \dots, y_{\#A})$	\iff	$\gamma \models_\Delta^\nu \phi[x_1/y_1, \dots, x_{\#A}/y_{\#A}]$, for some rule $A(x_1, \dots, x_{\#A}) \leftarrow \phi$ from Δ
$\gamma \models_\Delta^\nu \phi_1 * \phi_2$	\iff	there exist γ_1 and γ_2 , such that $\gamma = \gamma_1 \bullet \gamma_2$ and $\gamma_i \models_\Delta^\nu \phi_i$, for all $i = 1, 2$
$\gamma \models_\Delta^\nu \exists x . \phi$	\iff	$\gamma \models_\Delta^{\nu[x \leftarrow c]} \phi$, for some $c \in \mathbb{C}$

If $\gamma \models_\Delta^\nu \phi$, we say that the pair (γ, ν) is a Δ -model of ϕ . If ϕ is a predicate-free formula, the satisfaction relation does not depend on the SID, written $\gamma \models^\nu \phi$. A formula ϕ is *satisfiable* if and only if it has a model. A formula ϕ Δ -entails a formula ψ , written $\phi \models_\Delta \psi$, if and only if any Δ -model of ϕ is a Δ -model of ψ . Two formulæ are Δ -equivalent, written $\phi \equiv_\Delta \psi$ if and only if $\phi \models_\Delta \psi$ and $\psi \models_\Delta \phi$. A formula ϕ is Δ -tight if γ is tight (Def. 3), for any Δ -model (γ, ν) of ϕ . We omit mentioning Δ whenever it is clear from the context or not needed.

2.3 The Havoc Invariance Problem

This paper is concerned with the *havoc invariance* problem i.e., the problem of deciding whether the set of models of a given CL formula is closed under the execution of a sequence of interactions. The execution of an interaction $(c_i, p_i)_{i \in [1, n]}$ synchronizes transitions labeled by the ports p_1, \dots, p_n from the behaviors (i.e., replicas of the state machine \mathbb{B}) of c_1, \dots, c_n , respectively. This joint execution of several transitions in different components of the system is formally described by the *step* relation below:

► **Definition 7.** The step relation $\Rightarrow \subseteq \Gamma \times (\mathbb{C} \times \mathcal{P})^+ \times \Gamma$ is defined as:

$$(\mathcal{C}, \mathcal{I}, \varrho) \xrightarrow{(c_i, p_i)_{i \in [1, n]}} (\mathcal{C}, \mathcal{I}, \varrho') \text{ if and only if } (c_i, p_i)_{i \in [1, n]} \in \mathcal{I} \text{ and } \varrho' = \varrho[c_i \leftarrow q'_i]_{i \in [1, n]}$$

where $\varrho(c_i) = q_i$ and $q_i \xrightarrow{p_i} q'_i$ is a transition of \mathbb{B} , for all $i \in [1, n]$

The havoc relation \rightsquigarrow^* is the reflexive and transitive closure of the relation $\rightsquigarrow \subseteq \Gamma^2$: $(\mathcal{C}, \mathcal{I}, \varrho) \rightsquigarrow (\mathcal{C}, \mathcal{I}, \varrho')$ if and only if $(\mathcal{C}, \mathcal{I}, \varrho) \xrightarrow{(c_i, p_i)_{i \in [1, n]}} (\mathcal{C}, \mathcal{I}, \varrho')$, for some interaction $(c_i, p_i)_{i \in [1, n]} \in \mathcal{I}$.

► **Example 8.** Let $\gamma_i = (\{c_1, c_2, c_3\}, \{(c_i, \text{out}, c_{i \bmod 3+1}, \text{in}) \mid i \in [1, 3]\}, \varrho_i)$, for $i \in [1, 3]$ be configurations, where $\varrho_1(c_1) = \varrho_1(c_2) = \text{H}$, $\varrho_1(c_3) = \text{T}$, $\varrho_2(c_1) = \text{T}$, $\varrho_2(c_2) = \varrho_2(c_3) = \text{H}$, $\varrho_3(c_1) = \varrho_3(c_3) = \text{H}$, $\varrho_3(c_2) = \text{T}$. Then we have $\gamma_i \rightsquigarrow^* \gamma_j$, for all $i, j \in [1, 3]$. ◻

Two interactions $(c_1, p_1, \dots, c_n, p_n)$ and $(c_{i_1}, p_{i_1}, \dots, c_{i_n}, p_{i_n})$ such that $\{i_1, \dots, i_n\} = [1, n]$, are equivalent from the point of view of the step relation, since the set of executed transitions is the same; nevertheless, we chose to distinguish them in the following, for reasons of simplicity. Note, moreover, that the havoc relation does not change the component or the interaction set of a configuration, only its state map.

► **Definition 9.** Given an SID Δ and a predicate A , the problem $\text{HavocInv}[\Delta, A]$ asks whether for all $\gamma, \gamma' \in \Gamma$ and each store ν , such that $\gamma \models_{\Delta}^{\nu} A(x_1, \dots, x_{\#A})$ and $\gamma \rightsquigarrow^* \gamma'$, it is the case that $\gamma' \models_{\Delta}^{\nu} A(x_1, \dots, x_{\#A})$?

► **Example 10.** Consider a model $\gamma = (\{c_1, \dots, c_n\}, \{(c_i, \text{out}, c_{(i \bmod n)+1}, \text{in}) \mid i \in [1, n]\}, \varrho)$ of the formula $\text{ring}_{1,1}()$ i.e., having the property that $\varrho(c_i) = \text{H}$ and $\varrho(c_j) = \text{T}$ for at least two indices $i \neq j \in [1, n]$, where the SID that defines $\text{ring}_{1,1}()$ is given in §1.1. Similar to Example 8, in any configuration $\gamma' = (\{c_1, \dots, c_n\}, \{(c_i, \text{out}, c_{(i \bmod n)+1}, \text{in}) \mid i \in [1, n]\}, \varrho')$ such that $\gamma \rightsquigarrow^* \gamma'$, we have $\varrho'(c_k) = \text{H}$ and $\varrho'(c_\ell) = \text{T}$, for some $k \neq \ell \in [1, n]$, hence γ' is a model of $\text{ring}_{1,1}()$, meaning that $\text{ring}_{1,1}()$ is havoc invariant. Examples of formulæ that are not havoc invariant include e.g., $[x]@T * \langle x.\text{out}, y.\text{in} \rangle * [y]@H$. ◻

Without loss of generality, we consider the havoc invariance problem only for single predicate atoms. This is because, for any formula ϕ , such that $\text{fv}(\phi) = \{x_1, \dots, x_n\}$, one may consider a fresh predicate symbol (i.e., not in the SID) A_ϕ and add the rule $A_\phi(x_1, \dots, x_n) \leftarrow \phi$ to the SID. Then ϕ is havoc invariant if and only if $A_\phi(x_1, \dots, x_n)$ is havoc invariant.

3 From Havoc Invariance to Entailment

We describe a many-one reduction of the havoc invariance (Def. 9) to the entailment problem, following three steps. Given an instance $\text{HavocInv}[\Delta, A]$ of the havoc invariance problem, the SID Δ is first translated into a tree automaton recognizing trees labeled with predicate-free formulæ, that symbolically encode the set of Δ -models of the predicate atom $A(x_1, \dots, x_{\#A})$.

Second, we define a structure-preserving tree transducer that simulates the effect of executing exactly one interaction from such a model. Third, we compute the image of the language recognized by the first tree automaton via the transducer, as a second tree automaton, which is translated back into another SID $\bar{\Delta}$ defining one or more predicates $\bar{A}_1, \dots, \bar{A}_p$, among other. Finally, we prove that $\text{HavocInv}[\Delta, A]$ has a positive answer if and only if each of the entailments $\{\bar{A}_i(x_1, \dots, x_{\#A}) \models_{\Delta \cup \bar{\Delta}} A(x_1, \dots, x_{\#A})\}_{i=1}^p$ produced by the reduction, holds.

For the sake of self-containment, we recall below the definitions of trees, tree automata and (structure-preserving) tree transducers. Let $(\Sigma, \#)$ be a ranked alphabet, where each symbol $\alpha \in \Sigma$ has an associated arity $\#\alpha \geq 0$. A *tree* over Σ is a finite partial function $t : \mathbb{N}^* \rightarrow_{\text{fin}} \Sigma$, whose domain $\text{dom}(t) \subseteq_{\text{fin}} \mathbb{N}^*$ is both *prefix-closed* i.e., $u \in \text{dom}(t)$, for all $u, v \in \mathbb{N}^*$, such that $u \cdot v \in \text{dom}(t)$, and *complete* i.e., $\{n \in \mathbb{N} \mid u \cdot n \in \text{dom}(t)\} = [1, \#\alpha]$, for all $u \in \text{dom}(t)$. Given $u \in \text{dom}(t)$, the *subtree* of t rooted at u is the tree $t|_u$, such that $\text{dom}(t|_u) \stackrel{\text{def}}{=} \{w \mid u \cdot w \in \text{dom}(t)\}$ and $t|_u(w) \stackrel{\text{def}}{=} t(u \cdot w)$. We denote by $\mathbb{T}(\Sigma)$ the set of trees over a ranked alphabet Σ .

A *tree automaton* (TA) is a tuple $\mathcal{A} = (\Sigma, \mathcal{S}, \mathcal{F}, \delta)$, where Σ is a ranked alphabet, \mathcal{S} is a finite set of states, $\mathcal{F} \subseteq \mathcal{S}$ is a set of final states and δ is a set of transitions $\alpha(s_1, \dots, s_{\#\alpha}) \rightarrow s$; when $\#\alpha = 0$, we write $\alpha \rightarrow s$ instead of $\alpha() \rightarrow s$. A *run* of \mathcal{A} over a tree t is a function $\pi : \text{dom}(t) \rightarrow \mathcal{S}$, such that, for all $u \in \text{dom}(t)$, we have $\pi(u) = s$ if $(t(u))(\pi(u \cdot 1), \dots, \pi(u \cdot \#\alpha)) \rightarrow s \in \delta$. Given a state $q \in \mathcal{S}$, a run π of \mathcal{A} is *q-accepting* if and only if $\pi(\epsilon) = q$, in which case \mathcal{A} is said to *q-accept* t . We denote by $\mathcal{L}_q(\mathcal{A})$ the set of trees *q-accepted* by \mathcal{A} and let $\mathcal{L}(\mathcal{A}) \stackrel{\text{def}}{=} \bigcup_{q \in \mathcal{F}} \mathcal{L}_q(\mathcal{A})$. A language L is *recognizable* if and only if there exists a TA \mathcal{A} , such that $L = \mathcal{L}(\mathcal{A})$.

A *tree transducer* (TT) is a tree automaton over an alphabet of pairs $\mathcal{T} = (\Sigma^2, \mathcal{S}, \mathcal{F}, \delta)$, such that $\#\alpha = \#\beta = n$, for each transition $(\alpha, \beta)(s_1, \dots, s_n) \rightarrow s \in \delta$. Intuitively, a transition of the transducer reads a symbol α from the input tree and writes another symbol β to the output tree, at the same position. Clearly, any tree $t : \mathbb{N}^* \rightarrow_{\text{fin}} \Sigma^2$ with labels from the set of pairs $\{(\alpha, \beta) \in \Sigma^2 \mid \#\alpha = \#\beta\}$ can be viewed as a pair of trees (t_1, t_2) over Σ , such that $\text{dom}(t_1) = \text{dom}(t_2) = \text{dom}(t)$. In order to define the image of a tree language via a transducer, we define

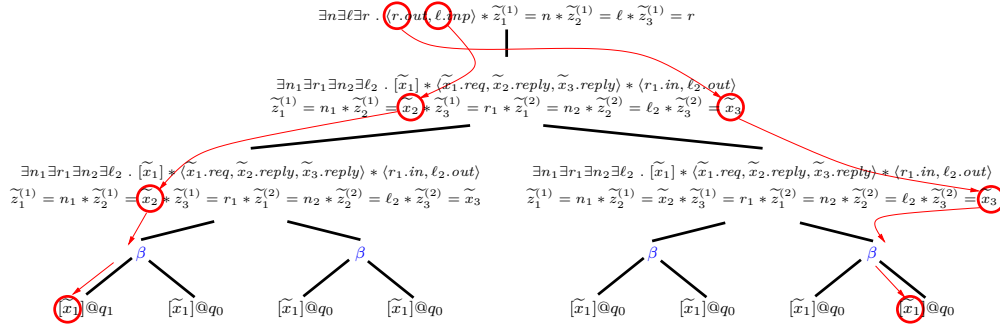
- (i) *projection* $L \downarrow_i \stackrel{\text{def}}{=} \{t_i \mid (t_1, t_2) \in L\}$, for all $i = 1, 2$, where $L \subseteq \mathbb{T}(\Sigma^2)$, and
- (ii) *cylindrification* $L \uparrow^i \stackrel{\text{def}}{=} \{(t_1, t_2) \mid t_i \in L\}$, for all $i = 1, 2$, where $L \subseteq \mathbb{T}(\Sigma)$.

The *image* of a language $L \subseteq \mathbb{T}(\Sigma)$ via a transducer \mathcal{T} is the language $\mathcal{T}(L) \stackrel{\text{def}}{=} (L \uparrow^1 \cap \mathcal{L}(\mathcal{T})) \downarrow_2$. It is manifest that $\mathcal{T}(L)$ is recognizable whenever L is recognizable.

3.1 From SID to Tree Automata and Back

We define a two-way connection between SIDs and TAs, as follows:

1. Given a finite SID Δ we define a TA \mathcal{A}_Δ , whose states q_A are named after the predicates A that occur in Δ and whose alphabet consists of the predicate-free formulæ from the rules of Δ , with variables mapped to canonical names, together with a tuple of arities, needed for later bookkeeping. Each tree $t \in \mathcal{L}_{q_A}(\mathcal{A}_\Delta)$ defines a unique predicate-free formula $\Phi(t)$, such that the Δ -models of a predicate atom $A(x_1, \dots, x_{\#A})$ are exactly the models of some $\Phi(t)$, for $t \in \mathcal{L}_{q_A}(\mathcal{A}_\Delta)$.
2. Conversely, given a TA \mathcal{A} over an alphabet of formulæ annotated with arities, the tuple of arities associated with each alphabet symbol allows to define a SID $\Delta_{\mathcal{A}}$, whose predicates A_q are named after the states q of the TA, such that the models of the formulæ $\Phi(t)$, such that $t \in \mathcal{L}_q(\mathcal{A})$ are exactly the $\Delta_{\mathcal{A}}$ -models of the predicate atom $A_q(x_1, \dots, x_{\#A_q})$.



■ **Figure 2** Tree Labeled with Formulae Encoding a System from Example 12.

Let us fix a countably infinite set of variables $\widetilde{\text{Var}} \stackrel{\text{def}}{=} \{\tilde{x}_i \mid i \geq 1\} \cup \{\tilde{z}_i^{(\ell)} \mid i, \ell \geq 1\}$, with the understanding that \tilde{x}_i are canonical names for the variables from the left-hand side and $\tilde{z}_i^{(\ell)}$ are canonical names for the variables occurring in the ℓ -th predicate atom from the right-hand side of a rule. An *alphabet symbol* $\alpha = \langle \psi, a_0, \dots, a_h \rangle$ consists of a predicate-free formula ψ and a tuple of positive integers $a_0, \dots, a_h \in \mathbb{N}$, such that $\text{fv}(\psi) = \{\tilde{x}_i \mid i \in [1, a_0]\} \cup \{\tilde{z}_i^{(\ell)} \mid \ell \in [1, h], i \in [1, a_\ell]\}$. We take the arity of such a symbol to be $\#\alpha \stackrel{\text{def}}{=} h$ and denote by $\widetilde{\Sigma}$ the (infinite) set of alphabet symbols. Trees labeled with symbols from $\widetilde{\Sigma}$ define predicate-free *characteristic formulae*, as follows:

► **Definition 11.** Given a tree $t \in \mathbb{T}(\widetilde{\Sigma})$, where $t(\epsilon) = \langle \exists y_1 \dots \exists y_m . \phi, a_0, \dots, a_h \rangle$ with ϕ a qpf formula, and a node $u \in \mathbb{N}^*$, we define the qpf characteristic formula:

$$\Psi^u(t) \stackrel{\text{def}}{=} \phi[\tilde{x}_j/x_j^u]_{j \in [1, a_0]} [\tilde{z}_j^{(\ell)}/x_j^{u, \ell}]_{\ell \in [1, h], j \in [1, a_\ell]} [y_j/y_j^u]_{j \in [1, m]} * \bigstar_{\ell \in [1, h]} \Psi^{u, \ell}(t_\ell)$$

Assuming that $t(v) = \langle \exists y_1 \dots \exists y_{m^v} . \phi^v, a_0^v, \dots, a_h^v \rangle$, for all $v \in \text{dom}(t)$, we consider also the predicate-free formula $\Phi^u(t) = (\exists x_j^{u, v})_{v \in \text{dom}(t) \setminus \{\epsilon\}, j \in [1, a_0^v]} (\exists y_j^{u, v})_{v \in \text{dom}(t), j \in [1, m^v]} . \Psi^u(t)$.

► **Example 12.** We consider a system whose components form a tree, in which each parent sends a request (*req*) to and receives replies (*reply*) from both its children. In addition, the leaves of the tree form a ring, with the *out* port of each leaf connected to the *in* port of its right neighbour. The system is described by the following inductive definitions:

$$\text{Root}() \leftarrow \exists n \exists \ell \exists r . \langle r.out, \ell.in \rangle * \text{Node}(n, \ell, r) \quad (1)$$

$$\begin{aligned} \text{Node}(n, \ell, r) \leftarrow \exists n_1 \exists r_1 \exists n_2 \exists \ell_2 . [n] * \langle n.req, n_1.reply, n_2.reply \rangle * \langle r_1.in, \ell_2.out \rangle * \\ \text{Node}(n_1, \ell, r_1) * \text{Node}(n_2, \ell_2, r) \end{aligned} \quad (2)$$

$$\text{Node}(n, \ell, r) \leftarrow [n] @_{q_0} \quad \text{Node}(n, \ell, r) \leftarrow [n] @_{q_1} \quad (3)$$

Fig. 2 shows a tree $t \in \mathbb{T}(\widetilde{\Sigma})$ describing an instance of the system, where $\widetilde{\Sigma} = \{\alpha, \beta, \gamma_0, \gamma_1\}$:

$$\begin{aligned} \alpha &\stackrel{\text{def}}{=} \langle \exists n \exists \ell \exists r . \langle r.out, \ell.in \rangle * \tilde{z}_1^{(1)} = n * \tilde{z}_2^{(1)} = \ell * \tilde{z}_3^{(1)} = r, 0, 3 \rangle \\ \beta &\stackrel{\text{def}}{=} \langle \exists n_1 \exists r_1 \exists n_2 \exists \ell_2 . [n] * \langle n.req, \ell.reply, r.reply \rangle * \langle r_1.in, \ell_2.out \rangle * \\ &\quad \tilde{z}_1^{(1)} = n_1 * \tilde{z}_2^{(1)} = \ell * \tilde{z}_3^{(1)} = r_1 * \tilde{z}_1^{(2)} = n_2 * \tilde{z}_2^{(2)} = \ell * \tilde{z}_3^{(2)} = r, 3, 3, 3 \rangle \\ \gamma_0 &\stackrel{\text{def}}{=} \langle [\tilde{x}_1] @_{q_0}, 3 \rangle \quad \gamma_1 \stackrel{\text{def}}{=} \langle [\tilde{x}_1] @_{q_1}, 3 \rangle \end{aligned}$$

For simplicity, Fig. 2 shows only the formulae, not the arity lists of the alphabet symbols. ◻

24:10 On an Invariance Problem for Parameterized Concurrent Systems

The models of the characteristic formula $\Psi^\epsilon(\mathfrak{t})$ of a tree $\mathfrak{t} \in \mathbb{T}(\tilde{\Sigma})$ define walks in the tree that correspond to chains of equalities between variables. Formally, a *walk* in \mathfrak{t} is a sequence of nodes $u_1, \dots, u_n \in \text{dom}(\mathfrak{t})$, such that u_i is either the parent or a child of u_{i+1} , for all $i \in [1, n-1]$. Note that a walk can visit the same node of the tree several times. In particular, if the characteristic formula $\Psi^\epsilon(\mathfrak{t})$ is tight (i.e., has only tight models in the sense of Def. 3) there exist equality walks between the node containing an interaction atom and the nodes where these variables are instantiated by component atoms. For instance, walks between the root containing $\langle r.out, \ell.in \rangle$ and the left- and right-most leaves, labeled with component atoms that associate elements of \mathbb{C} to the variables ℓ and r are shown in Fig. 2.

► **Lemma 13.** *Let $\mathfrak{t} \in \mathbb{T}(\tilde{\Sigma})$ be a tree, such that $\Psi^\epsilon(\mathfrak{t})$ is tight, (γ, ν) be a model of $\Psi^\epsilon(\mathfrak{t})$ and y^v, z^w be two variables that occur in a component and interaction atom of $\Psi^\epsilon(\mathfrak{t})$, respectively. Then $\nu(y^v) = \nu(z^w)$ if and only if there exists a walk u_1, \dots, u_n in \mathfrak{t} and variables $y^v = x_{i_1}^{u_1}, \dots, x_{i_n}^{u_n} = z^w$, such that either $x_{i_j}^{u_j}$ and $x_{i_{j+1}}^{u_{j+1}}$ are the same variable, or the equality atom $x_{i_j}^{u_j} = x_{i_{j+1}}^{u_{j+1}}$ occurs in $\Psi^\epsilon(\mathfrak{t})$, for all $j \in [1, n-1]$.*

Let Δ be a fixed and finite SID in the following. We build a TA \mathcal{A}_Δ that recognizes the Δ -models of each predicate atom defined by Δ , in the sense of Lemma 16 below.

► **Definition 14.** *We associate each rule $r : \mathbf{A}(x_1, \dots, x_{\#\mathbf{A}}) \leftarrow \exists y_1 \dots \exists y_m \cdot \phi * \bigstar_{\ell \in [1, h]} \mathbf{B}_\ell(z_1^\ell, \dots, z_{\#\mathbf{B}_\ell}^\ell) \in \Delta$, where ϕ is a qpf formula, with the alphabet symbol:*

$$\alpha_r \stackrel{\text{def}}{=} \left\langle \exists y_1 \dots \exists y_m \cdot \left(\phi * \bigstar_{\ell \in [1, h], i \in [1, \#\mathbf{B}_\ell]} \tilde{z}_i^{(\ell)} = z_i^\ell \right) [x_j / \tilde{x}_j]_{j \in [1, \#\mathbf{A}], \#\mathbf{A}, \#\mathbf{B}_1, \dots, \#\mathbf{B}_h} \right\rangle$$

Let $\mathcal{A}_\Delta \stackrel{\text{def}}{=} (\Sigma_\Delta, \mathcal{S}_\Delta, \delta_\Delta)$ be a TA, where $\Sigma_\Delta \stackrel{\text{def}}{=} \{\alpha_r \mid r \in \Delta\}$, $\mathcal{S}_\Delta \stackrel{\text{def}}{=} \{q_{\mathbf{A}} \mid \mathbf{A} \in \text{Def}(\Delta)\}$ and $\delta_\Delta \stackrel{\text{def}}{=} \{\alpha_r(q_{\mathbf{B}_1}, \dots, q_{\mathbf{B}_h}) \rightarrow q_{\mathbf{A}} \mid r \in \Delta\}$.

► **Example 15** (contd. from Example 12). The TA corresponding to the SID in Example 12 is $\mathcal{A}_\Delta = (\tilde{\Sigma}, \mathcal{S}_\Delta, \delta_\Delta)$, where $\tilde{\Sigma} = \{\alpha, \beta, \gamma_0, \gamma_1\}$, $\mathcal{S}_\Delta = \{q_{\text{Root}}, q_{\text{Node}}\}$ and $\delta_\Delta = \{\alpha(q_{\text{Node}}) \rightarrow q_{\text{Root}}, \beta(q_{\text{Node}}, q_{\text{Node}}) \rightarrow q_{\text{Node}}, \gamma_0 \rightarrow q_{\text{Node}}, \gamma_1 \rightarrow q_{\text{Node}}\}$. ◻

The following lemma proves that the predicate-free formulæ corresponding (in the sense of Def. 11) to the trees recognized by \mathcal{A}_Δ in a state $q_{\mathbf{A}}$ define the Δ -models of the predicate atom $\mathbf{A}(x_1, \dots, x_{\#\mathbf{A}})$:

► **Lemma 16.** *For any predicate $\mathbf{A} \in \text{Def}(\Delta)$, configuration γ , store ν and node $u \in \mathbb{N}^*$, we have $\gamma \models_\Delta^\nu \mathbf{A}(x_1^u, \dots, x_{\#\mathbf{A}}^u)$ if and only if $\gamma \models^\nu \Phi^u(\mathfrak{t})$, for some tree $\mathfrak{t} \in \mathcal{L}_{q_{\mathbf{A}}}(\mathcal{A}_\Delta)$.*

Conversely, given a tree automaton $\mathcal{A} = (\Sigma, \mathcal{S}, \delta)$, we construct a SID $\Delta_{\mathcal{A}}$ that defines the models of the predicate-free formulæ corresponding (Def. 11) to the trees recognized by \mathcal{A} (Lemma 19). We assume that the alphabet Σ consists of symbols $\langle \psi, a_0, \dots, a_h \rangle$ of arity h , where ψ is a predicate-free formula with free variables $\text{fv}(\psi) = \{\tilde{x}_i \mid i \in [1, a_0]\} \cup \{\tilde{z}_i^{(\ell)} \mid \ell \in [1, h], i \in [1, a_\ell]\}$ and that the transitions of the TA meet the requirement:

► **Definition 17.** *A TA \mathcal{A} is SID-compatible iff for any transitions $\langle \psi, a_0, \dots, a_h \rangle(q_1, \dots, q_h) \rightarrow q_0$ and $\langle \psi', a'_0, \dots, a'_h \rangle(q'_1, \dots, q'_h) \rightarrow q'_0$ of \mathcal{A} , we have $q_i = q'_i$ only if $a_i = a'_i$, for all $i \in [0, h]$.*

Let us fix a SID-compatible TA $\mathcal{A} = (\Sigma, \mathcal{S}, \delta)$ for the rest of this section.

► **Definition 18.** *The SID $\Delta_{\mathcal{A}}$ has a rule:*

$$\mathbf{A}_{q_0}(x_1, \dots, x_{a_0}) \leftarrow \exists y_1^1 \dots \exists y_{a_h}^h \cdot \phi[\tilde{x}_i / x_i]_{i \in [1, a_0]} [\tilde{z}_i^{(\ell)} / y_i^\ell]_{\ell \in [1, h], i \in [1, a_\ell]} * \bigstar_{\ell \in [1, h]} \mathbf{A}_{q_\ell}(y_1^\ell, \dots, y_{a_\ell}^\ell)$$

for each transition $\langle \phi, a_0, \dots, a_h \rangle(q_1, \dots, q_h) \rightarrow q_0$ of \mathcal{A} and those rules only.

The following lemma states that $\Delta_{\mathcal{A}}$ defines the set of models of the characteristic formulæ (Def. 11) of the trees recognized by \mathcal{A} .

► **Lemma 19.** *For any state $q \in \mathcal{S}$, configuration γ , store ν and node $u \in \mathbb{N}^*$, we have $\gamma \models_{\Delta_{\mathcal{A}}}^{\nu} A_q(x_1^u, \dots, x_{\#A_q}^u)$ if and only if $\gamma \models^{\nu} \Phi^u(\mathfrak{t})$, for some tree $\mathfrak{t} \in \mathcal{L}_q(\mathcal{A})$.*

3.2 Encoding Havoc Steps by Tree Transducers

The purpose of this section is the definition of a transducer that simulates one havoc step. Before giving its definition, we note that the havoc invariance problem can be equivalently defined by considering the transformation induced by a single havoc step, instead of an arbitrary sequence of steps. The following lemma can be taken as an equivalent definition:

► **Lemma 20.** *HavocInv $[\Delta, A]$ has a positive answer if and only if, for all $\gamma, \gamma' \in \Gamma$ and each store ν , such that $\gamma \models_{\Delta}^{\nu} A(x_1, \dots, x_{\#A})$ and $\gamma \rightsquigarrow \gamma'$, it is the case that $\gamma' \models_{\Delta}^{\nu} A(x_1, \dots, x_{\#A})$.*

We fix a SID Δ for the rest of this section and recall the existence of a fixed finite-state behavior $\mathbb{B} = (\mathcal{P}, \mathcal{Q}, \rightarrow)$ with ports \mathcal{P} , states \mathcal{Q} and transitions $q \xrightarrow{p} q' \in \mathcal{Q} \times \mathcal{P} \times \mathcal{Q}$. We define a transducer \mathcal{T}_{τ} parameterized by a given interaction type $\tau = (p_1, \dots, p_n) \in \mathcal{P}^+$. The havoc step transducer is the automata-theoretic union of the typed transducers over the set of interaction types that occurs in Δ .

Given a tree $\mathfrak{t} \in \mathbb{T}(\tilde{\Sigma})$, an interaction-typed transducer \mathcal{T}_{τ}

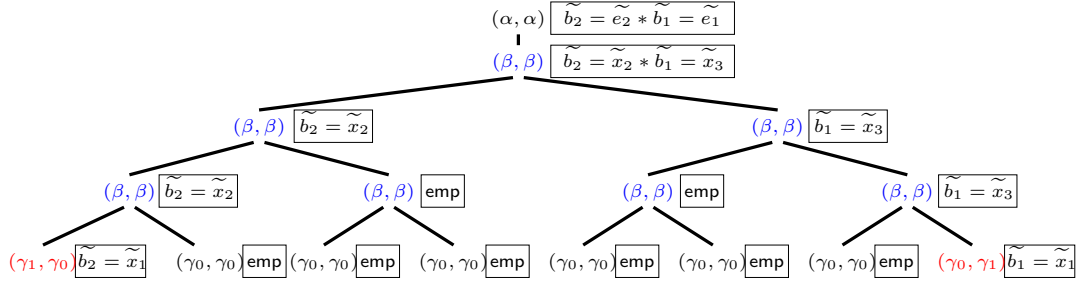
- (1) guesses an interaction atom $\langle z_1.p_1, \dots, z_n.p_n \rangle$ that occurs in some label of \mathfrak{t} ,
- (2) tracks the equality walks (Lemma 13) between each variable z_i and the component atom $[x_i]@q_i$ that defines the store value of z_i and its current state, and
- (3) replaces each state component atom $[x_i]@q_i$ by $[x_i]@q'_i$, where $q_i \xrightarrow{p_i} q'_i$ is a transition from \mathbb{B} , for each $i \in [1, n]$.

The output of the transducer is a tree $\mathfrak{t}' \in \mathbb{T}(\tilde{\Sigma})$, that symbolically encodes the effect of executing some interaction of type τ over \mathfrak{t} . The main challenge in defining \mathcal{T}_{τ} is that the equality walks between an interaction atom $\langle z_1.p_1, \dots, z_n.p_n \rangle$ and the component atoms instantiating the variables z_1, \dots, z_n may visit a tree node more than once. To capture this, the transducer will guess at once the equalities summarizing the different fragments of the walk that lie in the currently processed subtree of \mathfrak{t} . Accordingly, the states of \mathcal{T}_{τ} are conjunctions of equalities, with special variables \tilde{b}_i (resp. \tilde{e}_i) indicating whether a component (resp. interaction) atom has already been encountered in the current subtree, intuitively marking the *beginning* (resp. *end*) of the walk.

For an interaction type $\tau = (p_1, \dots, p_n)$, let $\widetilde{\text{TVar}}_{\tau} \stackrel{\text{def}}{=} \{\tilde{x}_i \mid i \in [1, \#(\Delta)]\} \cup \{\tilde{b}_i, \tilde{e}_i \mid i \in [1, n]\}$ and let $\text{Eq}(\widetilde{\text{TVar}}_{\tau})$ be the set of separating conjunctions of equality atoms i.e., $\varphi \stackrel{\text{def}}{=} *_{i \in I} x_i = y_i$, such that $\text{fv}(\varphi) \subseteq \widetilde{\text{TVar}}_{\tau}$. Note that $\exists x . \varphi$, for $\varphi \in \text{Eq}(\widetilde{\text{TVar}}_{\tau})$, is equivalent to a formula from $\text{Eq}(\widetilde{\text{TVar}}_{\tau})$ obtained by eliminating the quantifier: either x occurs in an atom $x = y$ for a variable y distinct from x then $(\exists x . \varphi) \equiv \varphi[x/y]$, or $x \notin \text{fv}(\varphi)$, in which case $(\exists x . \varphi) \equiv \varphi$.

► **Definition 21.** *The transducer $\mathcal{T}_{\tau} \stackrel{\text{def}}{=} (\Sigma_{\Delta}^2, \mathcal{S}_{\tau}, \mathcal{F}_{\tau}, \delta_{\tau})$, where $\tau = (p_1, \dots, p_n)$, is as follows:*

- $\mathcal{S}_{\tau} = \{\varphi \in \text{Eq}(\widetilde{\text{TVar}}_{\tau}) \mid \varphi \not\models (\tilde{b}_i = \tilde{b}_j), \varphi \not\models (\tilde{e}_i = \tilde{e}_j), \varphi \not\models (\tilde{b}_i = \tilde{e}_j), \text{ for any } i \neq j\}$,
- $\mathcal{F}_{\tau} = \{\varphi \in \mathcal{S}_{\tau} \mid \varphi \models *_{i \in [1, n]} (\tilde{b}_i = \tilde{e}_i)\}$, and
- δ_{τ} contains transitions of the form $(\alpha, \alpha')(\varphi_1, \dots, \varphi_h) \rightarrow_{\tau} \varphi$ where:
 - $\alpha = (\exists y_1 \dots \exists y_m . \psi, a_0, \dots, a_h)$ and $\alpha' = (\exists y_1 \dots \exists y_m . \psi', a_0, \dots, a_h)$, where ψ and ψ' are qpf formulæ such that $\text{fv}(\psi) = \text{fv}(\psi') \subseteq \widetilde{\text{TVar}}_{\tau} \cup \{y_1, \dots, y_m\}$,



■ **Figure 3** Tree Transducer for the Interactions of Type (out, in) in the System from Fig. 2.

- there exists a set $I = \{i_1, \dots, i_r\} \subseteq [1, n]$, variables $\xi_1, \dots, \xi_r \in \text{fv}(\psi)$ and transitions $q_1 \xrightarrow{p_{i_1}} q'_1, \dots, q_r \xrightarrow{p_{i_r}} q'_r$ in \mathbb{B} , such that $\psi = (\bigstar_{k \in [1, r]} [\xi_k] @ q_k) * \eta$ and $\psi' = (\bigstar_{k \in [1, r]} [\xi_k] @ q'_k) * \eta$, for some qpf formula η ,
- there exists a set $J \in \{\emptyset, [1, n]\}$, such that ψ contains an interaction atom $\langle \zeta_1.p_1, \dots, \zeta_n.p_n \rangle$ if $J = [1, n]$,
- the sets I and $\{i \in [1, n] \mid \tilde{b}_i \in \text{fv}(\varphi_\ell)\}_{\ell \in [1, h]}$ are pairwise disjoint,
- at most one of the sets $J, \{i \in [1, n] \mid \tilde{e}_i \in \text{fv}(\varphi_\ell)\}_{\ell \in [1, h]}$ is not empty,
- φ is the result of eliminating the quantifiers from the separating conjunction of equalities:

$$\exists \tilde{z}_1^{(1)} \dots \exists \tilde{z}_{a_h}^{(h)} \exists y_1 \dots \exists y_m \cdot \bigstar_{\ell \in [1, h]} \varphi_\ell[\tilde{x}_j / \tilde{z}_j^{(\ell)}]_{j \in [1, a_\ell]} * \bigstar_{k \in [1, r]} \tilde{b}_{i_k} = \xi_k * \bigstar_{\ell \in J} \tilde{e}_\ell = \zeta_\ell * \psi_{eq}$$

where ψ_{eq} is the separating conjunction of the equality atoms from ψ .

► **Example 22.** (contd. from Examples 12 and 15) Fig. 3 shows a run of the transducer $\mathcal{T}_{(out, in)}$, that describes the symbolic execution of the interaction corresponding to the $\langle r.out, \ell.in \rangle$ interaction atom from the root of the tree in Fig. 2. The states of the transducer are separating conjunctions of equality atoms, enclosed within square boxes. The transducer replaces the component atoms $\gamma_1 = \langle [\tilde{x}_1] @ q_1, 3 \rangle$ with $\gamma_0 = \langle [\tilde{x}_1] @ q_0, 3 \rangle$ (resp. γ_0 with γ_1) in the left-most (resp. right-most) leaf of the tree. \lrcorner

Let $L \subseteq \mathbb{T}(\tilde{\Sigma})$ be an arbitrary language. The following lemmas prove that the transducer \mathcal{T}_τ from Def. 21 correctly simulates a havoc step produced by an interaction of type τ .

► **Lemma 23.** For each tree $\mathfrak{t} \in L$, such that $\Phi^\epsilon(\mathfrak{t})$ is tight, configurations $\gamma = (\mathcal{C}, \mathcal{I}, \varrho), \gamma' \in \Gamma$ and store ν , such that $\gamma \models^\nu \Phi^\epsilon(\mathfrak{t})$ and $\gamma \xrightarrow{(c_i, p_i)_{i \in [1, n]}} \gamma'$, for some $c_1, \dots, c_n \in \mathcal{C}$ and $(c_i, p_i)_{i \in [1, n]} \in \mathcal{I}$, there exists a tree $\mathfrak{t}' \in \mathcal{T}_{(p_1, \dots, p_n)}(L)$, such that $\gamma' \models^\nu \Phi^\epsilon(\mathfrak{t}')$.

Note that the condition of $\Phi^\epsilon(\mathfrak{t})$ having only tight models is necessary to avoid interactions $(c_i, p_i)_{i \in [1, n]}$ that fire by “accident” i.e., when the interaction is created by an atom $\langle \zeta_1.p_1, \dots, \zeta_n.p_n \rangle$, with the components c_1, \dots, c_n created by component atoms $[\xi_1] @ q_1, \dots, [\xi_n] @ q_n$, such that the equality $\xi_i = \zeta_i$ is not the consequence of $\Phi^\epsilon(\mathfrak{t})$, for some $i \in [1, n]$. The effect of such interactions is not captured by the transducer introduced by Def. 21. Tightness is, moreover, a necessary condition of Lemma 13, that ensures the existence of equality walks between the variables occurring in an interaction atom and those of the atoms creating the components to which these variables are mapped, in a model of $\Phi^\epsilon(\mathfrak{t})$.

► **Lemma 24.** For each tree $\mathfrak{t}' \in \mathcal{T}_{(p_1, \dots, p_n)}(L)$, configuration $\gamma' \in \Gamma$ and store ν , such that $\gamma' \models^\nu \Phi^\epsilon(\mathfrak{t}')$, there exists a configuration $\gamma = (\mathcal{C}, \mathcal{I}, \varrho)$ and a tree $\mathfrak{t} \in L$, such that $\gamma \models^\nu \Phi^\epsilon(\mathfrak{t})$ and $\gamma \xrightarrow{(c_i, p_i)_{i \in [1, n]}} \gamma'$, for some $c_1, \dots, c_n \in \mathcal{C}$ and $(c_i, p_i)_{i \in [1, n]} \in \mathcal{I}$.

3.3 The Main Result

We establish the main result of this section, which is a many-one reduction of the havoc invariance to the entailment problem. The result is sharpened by proving that the reduction

- (i) preserves the class of the SID (see Def. 25 below), and
- (ii) is polynomial when several parameters of the SID are bounded by constants and simply exponential otherwise.

In particular, a class-preserving polynomial reduction ensures that the decidability and complexity upper bounds of the entailment problem carry over to the havoc invariance problem.

► **Definition 25.** For two predicate-free formulæ ϕ and ψ , we write $\phi \simeq \psi$ if and only if they become equivalent when dropping the state atoms from both. For an arity-preserving equivalence relation $\sim \subseteq \mathbb{A} \times \mathbb{A}$ (i.e., $\#A = \#B$, for all $A \sim B$), for any two rules r_1 and r_2 , we write $r_1 \approx r_2$ if and only if $r_1 = A(x_1, \dots, x_{\#A}) \leftarrow \exists y_1 \dots \exists y_m . \phi * \bigstar_{\ell \in [1, h]} B_\ell(\mathbf{z}_\ell)$, $r_2 = A'(x_1, \dots, x_{\#A'}) \leftarrow \exists y'_1 \dots \exists y'_p . \psi * \bigstar_{\ell \in [1, h]} B'_\ell(\mathbf{u}_\ell)$, $\exists y_1 \dots \exists y_m . \phi \simeq \exists y'_1 \dots \exists y'_p . \psi$, $A \sim A'$ and $B_\ell \sim B'_\ell$, for all $\ell \in [1, h]$. For two SIDs Δ_1 and Δ_2 , we write $\Delta_1 \preceq \Delta_2$ if and only if for each rule $r_1 \in \Delta_1$ there exists a rule $r_2 \in \Delta_2$, such that $r_1 \approx r_2$. We denote by $\Delta_1 \approx \Delta_2$ the conjunction of $\Delta_1 \preceq \Delta_2$ and $\Delta_2 \preceq \Delta_1$.

If $A_1 \sim A_2$ and $\Delta_1 \approx \Delta_2$ then Δ_1 -models of $A_1(x_1, \dots, x_{\#A_1})$ differ from the Δ_2 -models of $A_2(x_1, \dots, x_{\#A_1})$ only by a renaming of the states occurring within state atoms. This is because any derivation of the satisfaction relation $\gamma \models_{\Delta_1}^\nu A_1(x_1, \dots, x_{\#A_1})$ can be mimicked (modulo the state atoms that may change) by a derivation of $\gamma \models_{\Delta_2}^\nu A_2(x_1, \dots, x_{\#A_2})$, and viceversa. We are now in the position of stating the main result of this section:

► **Theorem 26.** Assuming that $A(x_1, \dots, x_{\#A})$ is a Δ -tight formula, each instance $\text{HavocInv}[\Delta, A]$ of the havoc invariance problem can be reduced to a set $\{\bar{A}_i(x_1, \dots, x_{\#A}) \models_{\Delta \cup \bar{\Delta}} A(x_1, \dots, x_{\#A})\}_{i=1}^p$ of entailments, where $\Delta \approx \bar{\Delta}$, for an arity-preserving equivalence relation $\sim \subseteq \mathbb{A} \times \mathbb{A}$, such that $\bar{A}_i \sim A$, for all $i \in [1, p]$. The reduction is polynomial, if $\#(\Delta)$, $N(\Delta)$ and $H(\Delta)$ are bounded by constants and simply exponential, otherwise.

4 Decidability and Complexity

We prove the undecidability of the havoc invariance problem (Def. 9) using a reduction from the universality of context-free languages, a textbook undecidable problem [2].

► **Theorem 27.** The $\text{HavocInv}[\Delta, A]$ problem is undecidable.

The undecidability proof for the havoc invariance problem uses an argument similar to the one used to prove undecidability of the entailment problem [4, Theorem 4]. We leverage further from this similarity and carve a fragment of CL with a decidable havoc invariance problem, based on the reduction from Theorem 26. For self-containment reasons, we recall the definition of a CL fragment for which the entailment problem is decidable (see [4, §6] for more details and proofs). This definition relies on three, easily checkable, syntactic restrictions on the rules of the SID and a decidable semantic restriction on the models of a predicate atom defined by the SID. The syntactic restrictions use the notion of profile:

► **Definition 28.** The profile of a SID Δ is the pointwise greatest function $\lambda_\Delta : \mathbb{A} \rightarrow \text{pow}(\mathbb{N})$, mapping each predicate A into a subset of $[1, \#A]$, such that, for each rule $A(x_1, \dots, x_{\#A}) \leftarrow \phi$ from Δ , each atom $B(y_1, \dots, y_{\#B})$ from ϕ and each $i \in \lambda_\Delta(B)$, there exists $j \in \lambda_\Delta(A)$, such that x_j and y_i are the same variable.

24:14 On an Invariance Problem for Parameterized Concurrent Systems

The profile identifies the parameters of a predicate that are always replaced by a variable $x_1, \dots, x_{\#A}$ in each unfolding of $A(x_1, \dots, x_{\#A})$, according to the rules in Δ ; it is computed by a greatest fixpoint iteration, in polynomial time.

► **Definition 29.** A rule $A(x_1, \dots, x_{\#A}) \leftarrow \exists y_1 \dots \exists y_m \cdot \phi * \bigstar_{\ell=1}^h B_\ell(z_1^\ell, \dots, z_{\#B_\ell}^\ell)$, where ϕ is a qpf formula, is said to be:

1. progressing (P) if and only if $\phi = [x_1] * \psi$, where ψ consists of interaction atoms involving x_1 and (dis-)equalities, such that $\bigcup_{\ell=1}^h \{z_1^\ell, \dots, z_{\#B_\ell}^\ell\} = \{x_2, \dots, x_{\#A}\} \cup \{y_1, \dots, y_m\}$,
2. connected (C) if and only if, for each $\ell \in [1, h]$ there exists an interaction atom in ψ that contains both z_1^ℓ and a variable from $\{x_1\} \cup \{x_i \mid i \in \lambda_\Delta(A)\}$,
3. equationally-restricted (e-restricted or R) if and only if, for every disequality $x \neq y$ from ϕ , we have $\{x, y\} \cap \{x_i \mid i \in \lambda_\Delta(A)\} \neq \emptyset$.

A SID Δ is progressing (P), connected (C) and e-restricted (R) if and only if each rule in Δ is progressing, connected and e-restricted, respectively.

► **Example 30.** For example, the rules for the $\text{chain}_{h,t}(x_1, x_2)$ predicates from the SID in §1.1 are PCR, but not the rules for $\text{ring}_{h,t}()$ predicates, that are neither progressing nor connected. The latter can be replaced with the following PCR rules:

$$\overline{\text{ring}}_{h,t}(x) \leftarrow \exists y \exists z \cdot [x]@q * \langle x.out, z.in \rangle * \text{chain}_{h',t'}(z, y) * \langle y.out, x.in \rangle, \text{ for all } h, t \in \mathbb{N}$$

Similarly, rule (2) for the *Node* predicate is PCR, but not rules (1) and (3), from Example 12. In order to obtain a SID that is PCR, these rules can be replaced with, respectively:

$$\begin{aligned} \text{Root}(n) &\leftarrow \exists n_1 \exists \ell_1 \exists r_1 \exists n_2 \exists \ell_2 \exists r_2 \cdot [n] * \langle n.req, n_1.reply, n_2.reply \rangle * \langle r_1.in, \ell_2.out \rangle * \\ &\quad \text{Node}(n_1, \ell_1, r_1) * \text{Node}(n_2, \ell_2, r_2) \end{aligned}$$

$$\text{Node}(n, \ell, r) \leftarrow [n] * \langle n.req, \ell.reply, r.reply \rangle * \langle \ell.in, r.out \rangle * \text{Leaf}(\ell) * \text{Leaf}(r) \quad \text{Leaf}(n) \leftarrow [n] \quad \lrcorner$$

A first property is that PCR SIDs define only tight configurations (Def. 3), a prerequisite for the reduction from Theorem 26:

► **Lemma 31.** Let Δ be a PCR SID and let $A \in \text{Def}(\Delta)$ be a predicate. Then, for any Δ -model (γ, ν) of $A(x_1, \dots, x_{\#A})$, the configuration γ is tight.

The last restriction for the decidability of entailments relates to the degree of the models of a predicate atom. The degree of a configuration is defined in analogy with the degree of a graph as the maximum number of interactions involving a component:

► **Definition 32.** The degree of a configuration $\gamma = (\mathcal{C}, \mathcal{I}, \varrho)$ is defined as $\delta(\gamma) \stackrel{\text{def}}{=} \max_{c \in \mathcal{C}} \delta_c(\gamma)$, where $\delta_c(\gamma) \stackrel{\text{def}}{=} \|\{(c_1, p_1, \dots, c_n, p_n) \in \mathcal{I} \mid c = c_i, i \in [1, n]\}\|$.

For instance, the configuration of the system from Fig. 1 (a) has degree two. The *degree boundedness* problem $\text{DegreeBound}[\Delta, A]$ asks, given a predicate A and a SID Δ , if the set $\{\delta(\gamma) \mid \gamma \models_\Delta \exists x_1 \dots \exists x_{\#A} \cdot A(x_1, \dots, x_{\#A})\}$ is finite. This problem is decidable [4, Theorem 3]. The entailment problem $A(x_1, \dots, x_{\#A}) \models_\Delta \exists x_{\#A+1} \dots \exists x_{\#B} \cdot B(x_1, \dots, x_{\#B})$ is known to be decidable for PCR SIDs Δ , provided, moreover, that $\text{DegreeBound}[\Delta, A]$ holds:

► **Theorem 33** ([4]). *The entailment problem*

$$A(x_1, \dots, x_{\#A}) \models_\Delta \exists x_{\#A+1} \dots \exists x_{\#B} \cdot B(x_1, \dots, x_{\#B}),$$

where Δ is PCR and $\text{DegreeBound}[\Delta, A]$ has a positive answer, is in 2EXP, if $\#(\Delta)$ and $N(\Delta)$ are bounded by constants and in 4EXP, otherwise.

Back to the havoc invariance problem, we give first a lower bound using a reduction from the entailment problem $A(x_1, \dots, x_{\#A}) \models_{\Delta} \exists x_{\#A+1} \dots \exists x_{\#B} . B(x_1, \dots, x_{\#B})$, where Δ is a PCR SID and $\text{HavocInv}[\Delta, A]$ has a positive answer. To the best of our efforts, we could not prove that the entailment problem is 2EXP-hard under the further assumption that $\#(\Delta)$ is bounded by a constant, which leaves the question of a matching lower bound for the havoc invariance problem open, in this case.

► **Lemma 34.** *The $\text{HavocInv}[\Delta, A]$ problem for PCR SIDs Δ , such that $\text{DegreeBound}[\Delta, A]$ has a positive answer, is 2EXP-hard.*

The main result of this section is a consequence of Theorems 26 and 33. In the absence of a constant bound on the parameters $\#(\Delta)$, $N(\Delta)$ and $H(\Delta)$, the entailment resulting from the reduction (Theorem 26) is of simply exponential size in the input and the time complexity of solving the entailments is 4EXP (Theorem 33), yielding a 5EXP upper bound:

► **Theorem 35.** *The $\text{HavocInv}[\Delta, A]$ problem, for PCR SIDs such that $\text{DegreeBound}[\Delta, A]$ has a positive answer is in 2EXP, if $\#(\Delta)$, $N(\Delta)$ and $H(\Delta)$ are bounded by constants and in 5EXP, otherwise.*

5 Conclusions

We have considered a logic for describing sets of configurations of parameterized concurrent systems, with user-defined network topology. The havoc invariance problem asks whether a given formula in the logic is invariant under the execution of the system starting from each configuration that is a model of a formula. An algorithm for this problem uses a many-one reduction to the entailment problem, thus leveraging from earlier results on the latter problem. We study the decidability and complexity of the havoc invariance problem and show that a doubly-exponential algorithm exists for a fairly general fragment of the logic, that encompasses all our examples. This result is relevant for automating the generation of correctness proofs for reconfigurable systems, that change the network topology at runtime.

References

- 1 Emma Ahrens, Marius Bozga, Radu Iosif, and Joost-Pieter Katoen. Local reasoning about parameterized reconfigurable distributed systems. *CoRR*, abs/2107.05253, 2021. [arXiv:2107.05253](#).
- 2 Yehoshua Bar-Hillel, Micha Perles, and Eli Shamir. On formal properties of simple phrase structure grammars. *Sprachtypologie und Universalienforschung*, 14:143–172, 1961.
- 3 Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.
- 4 Marius Bozga, Lucas Bueri, and Radu Iosif. Decision problems in a logic for reasoning about reconfigurable distributed systems. In *International Joint Conference on Automated Reasoning, to appear*, 2022. [arXiv:2202.09637](#).
- 5 Marius Bozga, Javier Esparza, Radu Iosif, Joseph Sifakis, and Christoph Welzel. Structural invariants for the verification of systems with parameterized architectures. In *Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020*, volume 12078 of *LNCS*, pages 228–246. Springer, 2020.
- 6 Marius Bozga and Radu Iosif. Specification and safety verification of parametric hierarchical distributed systems. In *Formal Aspects of Component Software - 17th International Conference, FACS 2021, Virtual Event, October 28-29, 2021, Proceedings*, volume 13077 of *Lecture Notes in Computer Science*, pages 95–114. Springer, 2021.

- 7 Antonio Bucchiarone and Juan P. Galeotti. Dynamic software architectures verification using dynalloy. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, 10, 2008.
- 8 Cristiano Calcagno, Peter W. O’Hearn, and Hongseok Yang. Local action and abstract separation logic. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pages 366–378. IEEE Computer Society, 2007. doi:10.1109/LICS.2007.30.
- 9 Byron Cook, Christoph Haase, Joël Ouaknine, Matthew J. Parkinson, and James Worrell. Tractable reasoning in a fragment of separation logic. In *CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 235–249. Springer, 2011.
- 10 Julien Dormoy, Olga Kouchnarenko, and Arnaud Lanoix. Using temporal logic for dynamic reconfigurations of components. In Luís Soares Barbosa and Markus Lumpe, editors, *Formal Aspects of Component Software - 7th International Workshop, FACS 2010*, volume 6921 of *Lecture Notes in Computer Science*, pages 200–217. Springer, 2010.
- 11 Antoine El-Hokayem, Marius Bozga, and Joseph Sifakis. A temporal configuration logic for dynamic reconfigurable systems. In Chih-Cheng Hung, Jiman Hong, Alessio Bechini, and Eunjee Song, editors, *SAC ’21: The 36th ACM/SIGAPP Symposium on Applied Computing, Virtual Event, Republic of Korea, March 22-26, 2021*, pages 1419–1428. ACM, 2021. doi:10.1145/3412841.3442017.
- 12 Klaus-Tycho Foerster and Stefan Schmid. Survey of reconfigurable data center networks: Enablers, algorithms, complexity. *SIGACT News*, 50(2):62–79, 2019.
- 13 Dan Hirsch, Paolo Inverardi, and Ugo Montanari. Graph grammars and constraint solving for software architecture styles. In *Proceedings of the Third International Workshop on Software Architecture, ISAW ’98*, pages 69–72, New York, NY, USA, 1998. Association for Computing Machinery. doi:10.1145/288408.288426.
- 14 Yonit Kesten, Amir Pnueli, Elad Shahar, and Lenore D. Zuck. Network invariants in action. In *CONCUR 2002 - Concurrency Theory, 13th International Conference*, volume 2421 of *LNCS*, pages 101–115. Springer, 2002.
- 15 Arnaud Lanoix, Julien Dormoy, and Olga Kouchnarenko. Combining proof and model-checking to validate reconfigurable architectures. *Electron. Notes Theor. Comput. Sci.*, 279(2):43–57, 2011.
- 16 Daniel Le Metayer. Describing software architecture styles using graph grammars. *IEEE Transactions on Software Engineering*, 24(7):521–533, 1998. doi:10.1109/32.708567.
- 17 David Lesens, Nicolas Halbwachs, and Pascal Raymond. Automatic verification of parameterized linear networks of processes. In *The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 346–357. ACM Press, 1997.
- 18 Mohammad Noormohammadpour and Cauligi S. Raghavendra. Datacenter traffic control: Understanding techniques and tradeoffs. *IEEE Commun. Surv. Tutorials*, 20(2):1492–1525, 2018.
- 19 John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, pages 55–74. IEEE Computer Society, 2002. doi:10.1109/LICS.2002.1029817.
- 20 Ze’ev Shtadler and Orna Grumberg. Network grammars, communication behaviors and automatic verification. In Joseph Sifakis, editor, *Automatic Verification Methods for Finite State Systems, International Workshop*, volume 407 of *LNCS*, pages 151–165. Springer, 1989.
- 21 Pierre Wolper and Vinciane Lovinfosse. Verifying properties of large sets of processes with network invariants. In *Automatic Verification Methods for Finite State Systems, International Workshop*, volume 407 of *LNCS*, pages 68–80. Springer, 1989.

Towards Concurrent Quantitative Separation Logic

Ira Fesefeldt   

Software Modeling and Verification Group, RWTH Aachen University, Germany

Joost-Pieter Katoen   

Software Modeling and Verification Group, RWTH Aachen University, Germany

Thomas Noll   

Software Modeling and Verification Group, RWTH Aachen University, Germany

Abstract

In this paper, we develop a novel verification technique to reason about programs featuring concurrency, pointers and randomization. While the integration of concurrency and pointers is well studied, little is known about the combination of all three paradigms. To close this gap, we combine two kinds of separation logic – Quantitative Separation Logic and Concurrent Separation Logic – into a new separation logic that enables reasoning about lower bounds of the probability to realise a postcondition by executing such a program.

2012 ACM Subject Classification Theory of computation → Program verification; Theory of computation → Concurrent algorithms; Mathematics of computing → Probabilistic reasoning algorithms

Keywords and phrases Randomization, Pointers, Heap-Manipulating, Separation Logic, Concurrency

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.25

Related Version *Extended Version*: <https://arxiv.org/abs/2207.02822> [16]

1 Introduction and Related Work

In this paper, we aim to provide support for formal reasoning about concurrent imperative programs that are extended by two important features: dynamic data structures and randomisation. In other words, it deals with the analysis and verification of concurrent probabilistic pointer programs. This problem is of practical interest as many concurrent algorithms operating on data structures use randomisation to reduce the level of interaction between threads. For example, probabilistic skip lists [48] work well in the concurrent setting [17] because threads can independently manipulate nodes in the list without much synchronisation. In contrast, scalability of traditional balanced tree structures is difficult to achieve, since re-balancing operations may require locking access to large parts of the data structure. Bloom filters are another example of a probabilistic data structure supporting parallel access [8]. A further aspect is that stochastic modelling naturally arises when analysing faulty behaviour of (concurrent) software systems, as we later demonstrate in Section 5.

However, the combination of these features poses severe challenges when it comes to implementing and reasoning about concurrent randomised algorithms that operate on dynamic data structures. To give a systematic overview of related approaches, we mention that a number of program logics for reasoning about concurrent software have been developed [13, 14, 18, 30, 32, 43]. Next, we will address the programming-language extensions in isolation and then consider their integration. An overview is shown in Figure 1.

Pointers. Pointers constitute an essential concept in modern programming languages, and are used for implementing dynamic data structures like lists, trees etc. However, many software bugs can be traced back to the erroneous use of pointers by e.g. dereferencing null pointers or accidentally pointing to wrong parts of the heap, creating the need for



© Ira Fesefeldt, Joost-Pieter Katoen, and Thomas Noll;
licensed under Creative Commons License CC-BY 4.0

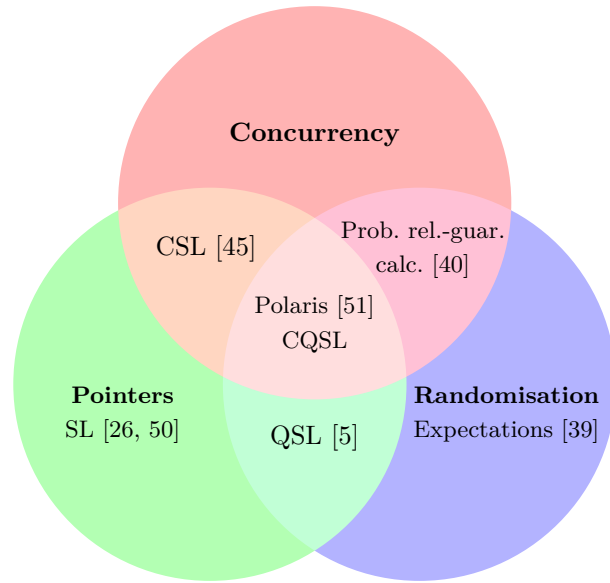
33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 25; pp. 25:1–25:24



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Overview of programming language features and formal approaches (CQSL denotes our concurrent extension of QSL).

computer-aided verification methods. The most popular formalism for reasoning about such programs is Separation Logic (SL) [26, 50], which supports Hoare-style verification of imperative, heap-manipulating and, possibly, concurrent programs. Its assertion language extends first-order logic with connectives that enable concise specifications of how program memory, or other resources, can be split-up and combined. In this way, SL supports local reasoning about the resources employed by programs. Consequently, program parts can be verified by considering only those resources they actually access – a crucial property for building scalable tools including automated verifiers [7, 28, 42, 47], static analysers [6, 10, 20], and interactive theorem provers [31].

The notion of resources, and in particular their controlled access, becomes even more important in a concurrent setting. Therefore, SL has been extended to Concurrent Separation Logic (CSL) [45] to enable reasoning about resource ownership, where the resource typically is dynamically allocated memory (i.e., the heap). The popularity of CSL is evident by the number of its extensions [9]. Of particular importance to our work is [53], which presents a soundness result for CSL that is formulated in an inductive manner, matching the “small-step” operational style of semantics. Here, we will employ a similar technique that also takes quantitative aspects (probabilities) into account.

Randomisation. Probabilistic programs (i.e., programs with the ability to sample from probability distributions) are increasingly popular for implementing efficient randomised algorithms [41] and describing uncertainty in systems [11, 19], among other similar tasks. In such applications, the purely qualitative (true vs. false) approach of classical logic is obviously not sufficient. The method advocated by us is based on weakest precondition reasoning as established in a classical setting by Dijkstra [12]. It has been extended to provide semantic foundations for probabilistic programs by Kozen [34, 35] and McIver & Morgan [39]. The latter also coined the term “weakest preexpectation” for random variables that take over the role of logical formulae when doing quantitative reasoning about probabilistic programs – the quantitative analogue of weakest preconditions. Their relation to operational models is

studied in [21]. Moreover, weakest preexpectation reasoning has been shown to be useful for obtaining bounds on the expected resource consumption [44] and, especially, the expected run-time [33] of probabilistic programs.

However, verification techniques that support reasoning about both randomisation and dynamic data structures are rare – a surprising situation given that randomised algorithms typically rely on such data structures. One notable exception is the extension of SL to Quantitative Separation Logic (QSL) [4, 5], which marries SL and weakest preexpectations. QSL has successfully been applied to the verification of randomised algorithms, and QSL expectations have been formalised in Isabelle/HOL [24]. The present work builds on these results by additionally taking concurrency into account.

A prior program logic designed for reasoning about programs that are both concurrent and randomised but do not maintain dynamic data structures is the probabilistic rely-guarantee calculus developed by McIver et al. [40], which extends Jones’s original rely-guarantee logic [30] by probabilistic constructs.

Later, Tassarotti & Harper [51] address the full setting of concurrent probabilistic pointer programs by combining CSL with probabilistic relational Hoare logic [3] to obtain Polaris, a Concurrent Separation Logic with support for probabilistic reasoning. Verification is thus understood as establishing a relation between a program to be analysed and a program which is known to be well-behaved. Programs which do not almost surely terminate, however, are outside the scope of their approach. In contrast, the goal of our method is to directly measure quantitative program properties on source-code level using weakest liberal preexpectations defined by a set of proof rules, including possibly non-almost surely terminating programs. Since the weakest liberal preexpectation includes non-termination probability, we can use invariants to bound the weakest liberal preexpectation of loops from below.

The main contributions of this paper are:

- the definition of a concurrent heap-manipulating probabilistic guarded command language (chpGCL) and its operational semantics in terms of Markov Decision Processes (MDP);
- a formal framework for reasoning about quantitative properties of chpGCL programs, which is obtained by extending classical weakest liberal preexpectations by resource invariants;
- a sound proof system that supports backward reasoning about such preexpectations; and
- the demonstration of our verification method on a (probabilistic) producer-consumer example.

The remainder of this paper is organised as follows. Section 2 introduces QSL as an assertion language for quantitative reasoning about (both sequential and concurrent) probabilistic pointer programs. In Section 3, we present the associated programming language (chpGCL) together with an operational semantics. Next, in Section 4 we develop a calculus for reasoning about lower bounds of weakest liberal preexpectations. Its usage is demonstrated in Section 5, and in Section 6 we conclude and explain further research directions. The paper is accompanied by an extended version providing elaborated proofs and an appendix providing additional details about the examples.

2 Quantitative Separation Logic

To reason about probability distributions over states of a program, we use Quantitative Separation Logic (QSL) [5, 37]. QSL is an extension of classical (or *qualitative*) Separation Logic in the sense that instead of mapping stack/heap pairs to booleans in order to gain a set characterization of states, we assign probabilities to stack/heap pairs.

► **Definition 2.1 (Stack).** Let Vars be a fixed set of variables. A stack $s: \text{Vars} \rightarrow \mathbb{Z}$ is a mapping from variable symbols to values. We denote the set of all stacks by Stacks .

When evaluating an (arithmetic or boolean) expression e with respect to a stack s , we write $e(s)$. In this sense, expressions are mappings from stacks to values. The stack that agrees with a stack s except for the value of x , which is mapped to v , is denoted as $s[x := v]$.

► **Definition 2.2 (Heaps).** A heap $h: L \rightarrow \mathbb{Z}$ is a mapping from a finite subset of locations $L \subset \mathbb{N}_{>0}$ to values. We denote the set of all heaps by Heaps .

We furthermore write $\text{dom}(h)$ for the domain of h , $h_1 \perp h_2$ if and only if $\text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset$, and for disjoint heaps $h_1 \perp h_2$ we define the disjoint union of heaps h_1 and h_2 as

$$(h_1 \star h_2)(\ell) = \begin{cases} h_1(\ell) & \text{if } \ell \in \text{dom}(h_1) \\ h_2(\ell) & \text{if } \ell \in \text{dom}(h_2) \\ \text{undef} & \text{else .} \end{cases}$$

A pair of a stack and a heap is a state of the program. The stack is used to describe the variables of the program. The heap describes the addressable memory of the program.

► **Definition 2.3 (Program States).** A program state $\sigma \in \text{Stacks} \times \text{Heaps}$ is a pair consisting of a stack and a heap. The set of all states is denoted by States .

Expectations are random variables that map states to non-negative reals. In this paper, we only consider one-bounded expectations. These do not map states to arbitrary non-negative reals, but only to reals between 0 and 1. The nomenclature of calling these expectations rather than random variables is due to the weakest preexpectation calculus being used to derive expectations.

► **Definition 2.4 (Expectations).** A (one-bounded) expectation $X: \text{States} \rightarrow [0, 1]$ is a mapping from program states to probabilities. We write $\mathbb{E}_{\leq 1}$ for the set of all (one-bounded) expectations. We call an expectation φ qualitative if for all $(s, h) \in \text{States}$ we have that $\varphi(s, h) \in \{0, 1\}$. We define the partial order $(\mathbb{E}_{\leq 1}, \leq)$ as the pointwise application of less than or equal, i.e., $X \leq Y$ if and only if $\forall (s, h) \in \text{States} X(s, h) \leq Y(s, h)$.

We use capital letters for regular (one-bounded) expectations and Greek letters for qualitative expectations. As in [5], we choose to not give a specific syntax for QSL since the weakest liberal preexpectation of a given postexpectation – for which we provide more detail in Section 3 – may not be expressible in a given syntax. Instead, we prefer to interpret expectations as *extensional objects* that can be combined via various connectives. These connectives include (but are not limited to) the pointwise-applied connectives of addition, multiplication, exponentiation, maximum and minimum. As it is common in quantitative logics, the maximum/minimum is the quantitative extension of disjunction/conjunction, respectively. However, multiplication can be chosen as the quantitative extension of conjunction as well. We denote the substitution of a variable x by the expression e in the expectation X as $X[x := e]$ and define it as $X[x := e](s, h) = X[s[x := e(s)], h)$. When dealing with state predicates, we use Iverson brackets [27] to cast boolean values into integers:

$$[b](s, h) = \begin{cases} 1 & \text{if } (s, h) \in b \\ 0 & \text{else} \end{cases}$$

Note that we could also define predicates as mappings from states to 0 or 1. We refrain from this, since (1) usage of Iverson brackets is standard in weakest preexpectation reasoning and (2) we may use QSL inside of Iverson brackets.

For a state (s, h) , the empty heap predicate **emp** holds if and only if $\text{dom}(h) = \emptyset$, the points-to predicate $e \mapsto e_0, \dots, e_n$ holds if and only if $\text{dom}(h) = \{e(s) + 0, \dots, e(s) + n\}$ and $\forall i \in \{0, \dots, n\} h(e(s) + i) = e_i(s)$, the allocated predicate $e \mapsto -$ holds if and only if $\text{dom}(h) = \{e(s)\}$, and the equality predicate $e = e'$ holds if and only if $e(s) = e'(s)$.

We also use quantitative extensions of two separation connectives – the separating conjunction and the magic wand. The quantitative extension of the separating conjunction, which we call *separating multiplication*, maximises the value of the product of its arguments applied to separated heaps:

$$(X \star Y)(s, h) = \sup \{ X(s, h_1) \cdot Y(s, h_2) \mid h_1 \star h_2 = h \}$$

The definition of separating multiplication is similar to the classical separating conjunction: the existential quantifier is replaced by a supremum and the conjunction by a multiplication. Note that the set over which the supremum ranges is never empty.

The (guarded) quantitative magic wand is defined for a qualitative first argument and a quantitative second argument. We minimise the value of the second argument applied to the original heap joined with a heap that evaluates the first argument to 1, i.e., for qualitative expectation φ and expectation Y we have:

$$(\varphi \multimap Y)(s, h) = \inf \{ Y(s, h'') \mid \varphi(s, h') = 1, h'' = h \star h' \}$$

If the set is empty, the infimum evaluates to the greatest element of all probabilities, which is 1. Although it is also possible to allow expectations in both arguments (cf. [5]), we restrict ourselves to the guarded version of the magic wand. This restriction allows us to exploit the superdistributivity of multiplication, i.e., $\varphi \multimap (X \cdot Y) \geq (\varphi \multimap X) \cdot (\varphi \multimap Y)$.

► **Example 2.5.** To illustrate separating operations and lower bounding in QSL, we consider $X = [x \mapsto -] \star ([x \mapsto y] \multimap (0.5 \cdot [x \mapsto -]))$, $Y = 0.5 \cdot [x \mapsto -]$ and $Z = 0.5 \cdot [x \mapsto y]$. Let us consider the semantics of X in more detail. X is non-zero only for states that allocate exactly x . In this case, after changing the value pointed to by x to y , 0.5 is returned if x is still allocated (which obviously holds). Thus, the combination of separating multiplication and magic wand realises a change of value: First a pointer is removed from the heap by using separating multiplication, and afterwards we add it back with a different value using the magic wand. Note that $[x \mapsto y]$ is qualitative, which is required for our version of the magic wand. Then we have $X = Y$ and $Z \leq X$.

3 Programming Language and Operational Semantics

Our programming language is a concurrent extension of the heap-manipulating and probabilistic guarded command language [5]. Our language features both deterministic and probabilistic control flow, atomic regions, concurrent threads operating on shared memory, variable-based assignments, and heap manipulations. Although our language allows arbitrary shared memory, we will later only be able to reason about shared memory in the heap. Conditional choice without an else branch is considered syntactic sugar. Atomic regions consist of programs without memory allocation or concurrency. However, probabilistic choice is admitted. Programs that satisfy this restriction are called *tame*.

The reason to restrict the program fragment within atomic regions is that non-tame statements introduce non-determinism (as addresses to be allocated and schedulings of concurrent programs are chosen non-deterministically), which would increase the semantics'

complexity while providing only little benefit (we refer to [1] regarding the handling of non-tame probabilistic programs in atomic regions). If an atomic region loops with a certain probability p , we instead transition to a non-terminating program with probability p .

► **Definition 3.1** (Concurrent Heap-Manipulating Probabilistic Guarded Command Language). *The concurrent heap-manipulating probabilistic guarded command language chpGCL is generated by the grammar*

$C \longrightarrow$	\downarrow	<i>(terminated program)</i>
	diverge	<i>(non-terminating program)</i>
	$x := e$	<i>(assignment)</i>
	$\{C\} [e_p] \{C\}$	<i>(prob. choice)</i>
	$C; C$	<i>(seq. composition)</i>
	atomic $\{C\}$	<i>(atomic region)</i>
	if $(b) \{C\}$ else $\{C\}$	<i>(conditional choice)</i>
	while $(b) \{C\}$	<i>(loop)</i>
	$C \parallel C$	<i>(concurrency)</i>
	$x := \mathbf{new}(e_0, \dots, e_n)$	<i>(allocation)</i>
	free (e) ,	<i>(disposal)</i>
	$x := \langle e \rangle$	<i>(lookup)</i>
	$\langle e \rangle := e'$	<i>(mutation)</i>

where x is a variable, $e, e', e_i: \mathbf{Stacks} \rightarrow \mathbb{Z}$ are arithmetic expressions, $e_p: \mathbf{Stacks} \rightarrow [0, 1]$ is a probabilistic arithmetic expression and $b \subseteq \mathbf{Stacks}$ is a guard.

► **Example 3.2.** We consider as running example a little program with two threads synchronizing over a randomised value:

$$\langle r \rangle := -1; \quad \left\{ \langle r \rangle := 0 \right\} [0.5] \left\{ \langle r \rangle := 1 \right\} \parallel \begin{cases} y := \langle r \rangle; \\ \mathbf{while}(y = -1) \{ y := \langle r \rangle \}; \end{cases}$$

We first initialise our resource r with some integer that stands for an undefined value (here -1). The first thread now either assigns 0 or 1 with probability 0.5 to r . As soon as r has a new value, the second thread receives this value and terminates as r is not -1 any more.

We define the operational semantics of our programming language chpGCL in the form of a *Markov Decision Process* (MDP for short). An MDP allows the use of both *non-determinism*, which we need for interleaving multiple threads, and *probabilities*, which are used for encoding probabilistic program commands. A transition between states is thus always annotated with two parameters: (1) an action that is taken non-deterministically and (2) a probability to transition to a state given the aforementioned action.

► **Definition 3.3** (Markov Decision Process). *A Markov Decision Process $M = (U, \mathbf{Act}, \mathbb{P})$ consists of a countable set of states U , a mapping from states to enabled actions $\mathbf{Act}: U \rightarrow 2^A$ for a countable set of actions A , and a transition probability function $\mathbb{P}: (U \times A) \rightarrow U \rightarrow [0, 1]$ where for all $\sigma \in U$ and $a \in \mathbf{Act}(\sigma)$ we require $\sum_{\sigma' \in U} \mathbb{P}(\sigma, a)(\sigma') = 1$. We also use the shorthand notation $\sigma \xrightarrow[p]{a} \sigma'$ for $\mathbb{P}(\sigma, a)(\sigma') = p$ in case $p > 0$.*

$$\begin{array}{c}
\frac{}{x := e, (s, h) \xrightarrow[\text{assign}]{1} \downarrow, (s[x := e(s)], h)} \text{ASSIGN} \\
\frac{e(s) \in \text{dom}(h)}{x := \langle e \rangle, (s, h) \xrightarrow[\text{lookup}]{1} \downarrow, (s[x := h(e(s))], h)} \text{LOOKUP} \quad \frac{e(s) \notin \text{dom}(h)}{x := \langle e \rangle, (s, h) \xrightarrow[\text{lookup-abt}]{1} \text{abort}} \text{LOOKUP-ABT} \\
\frac{e(s) \in \text{dom}(h)}{\langle e \rangle := e', (s, h) \xrightarrow[\text{mutation}]{1} \downarrow, (s, h[e(s) := e'(s)])} \text{MUT} \quad \frac{e(s) \notin \text{dom}(h)}{\langle e \rangle := e', (s, h) \xrightarrow[\text{mutation-abt}]{1} \text{abort}} \text{MUT-ABT} \\
\frac{e(s) \in \text{dom}(h) \quad h' = h \setminus \{e(s) \mapsto h(e(s))\}}{\text{free}(e), (s, h) \xrightarrow[\text{free}]{1} \downarrow, (s, h')} \text{FREE} \quad \frac{e(s) \notin \text{dom}(h)}{\text{free}(e), (s, h) \xrightarrow[\text{free-abt}]{1} \text{abort}} \text{FREE-ABT} \\
\frac{\ell + 0, \dots, \ell + n \in \mathbb{N}_{>0} \setminus \text{dom}(h) \quad e_0(s) = v_0, \dots, e_n(s) = v_n \quad h' = h \star \{\ell + 0 \mapsto v_1\} \star \dots \star \{\ell + n \mapsto v_n\}}{x := \text{new}(e_0, \dots, e_n), (s, h) \xrightarrow[\text{alloc-}\ell]{1} \downarrow, (s[x := \ell], h')} \text{ALLOC}
\end{array}$$

■ **Figure 2** Operational semantics of basic commands in `chpGCL`.

$$\begin{array}{c}
\frac{C_1, (s, h) \xrightarrow[a]{p} C'_1, (s', h')}{C_1; C_2, (s, h) \xrightarrow[a]{p} C'_1; C_2, (s', h')} \text{SEQ} \quad \frac{C_2, (s, h) \xrightarrow[a]{p} C'_2, (s', h')}{\downarrow; C_2, (s, h) \xrightarrow[a]{p} C'_2, (s', h')} \text{SEQ-END} \\
\frac{C_1, (s, h) \xrightarrow[a]{p} \text{abort}}{C_1; C_2, (s, h) \xrightarrow[a]{p} \text{abort}} \text{SEQ-ABT} \quad \frac{C_2, (s, h) \xrightarrow[a]{p} \text{abort}}{\downarrow; C_2, (s, h) \xrightarrow[a]{p} \text{abort}} \text{SEQ-END-ABT} \\
\frac{s \in b}{\text{if}(b) \{C_1\} \text{ else } \{C_2\}, (s, h) \xrightarrow[\text{if-t}]{1} C_1, (s, h)} \text{IF-T} \quad \frac{s \notin b}{\text{if}(b) \{C_1\} \text{ else } \{C_2\}, (s, h) \xrightarrow[\text{if-f}]{1} C_2, (s, h)} \text{IF-F} \\
\frac{s \in b}{\text{while}(b) \{C_1\}, (s, h) \xrightarrow[\text{loop-t}]{1} C_1; \text{while}(b) \{C_1\}, (s, h)} \text{WHILE-T} \\
\frac{s \notin b}{\text{while}(b) \{C_1\}, (s, h) \xrightarrow[\text{loop-f}]{1} \downarrow, (s, h)} \text{WHILE-F} \quad \frac{}{\text{diverge}, (s, h) \xrightarrow[\text{div}]{1} \text{diverge}, (s, h)} \text{DIV} \\
\frac{e_p(s) = p}{\{C_1\} [e_p] \{C_2\}, (s, h) \xrightarrow[\text{prob}]{p} C_1, (s, h)} \text{PROB-L} \quad \frac{e_p(s) = p}{\{C_1\} [e_p] \{C_2\}, (s, h) \xrightarrow[\text{prob}]{1-p} C_2, (s, h)} \text{PROB-R}
\end{array}$$

■ **Figure 3** Operational semantics of non-concurrent control-flow operations in `chpGCL`.

We define the operational semantics of `chpGCL` as an MDP. A state in this MDP consists of a `chpGCL` program to be executed and a program state (s, h) . The meaning of basic commands, i.e., assignments, heap mutations, heap lookups, memory allocation and disposal, is defined by the inference rules shown in Figure 2. An action is enabled if and only if an inference rule for this action exists. We use an `abort` keyword to indicate that a memory access error happened and terminate at this state. We consider aborted runs as undesired runs. The condition $\sum_{\sigma' \in U} \mathbb{P}(\sigma, a)(\sigma') = 1$ holds for all states in a `chpGCL` program. States with program \downarrow or `abort` have no enabled actions, thus the condition holds trivially for all enabled actions a ; non-probabilistic programs only have actions with trivial distributions; states with probabilistic choice only have a single action with a biased coin-flip distribution; and other programs are composed of these.

Control-flow statements include while loops, conditional choice, sequential composition and probabilistic choice, and we define their operational semantics in Figure 3. For the sake of brevity, we do not include a command to sample from a distribution.

The remaining control-flow statements handle concurrency, i.e., the concurrent execution of two threads and the atomic execution of regions. An atomic region may only terminate with a certain probability. The notation $C, (s, h) \xrightarrow[p]{*} \dots$ denotes that program C does not terminate on state (s, h) with probability p . As mentioned before, we will only allow

$$\begin{array}{c}
\frac{C, (s, h) \xrightarrow{p}^* \downarrow, (s', h') \quad C \text{ is tame}}{\text{atomic}\{C\}, (s, h) \xrightarrow[p]{\text{atomic}} \downarrow, (s', h')} \text{ ATOM-END} \qquad \frac{C, (s, h) \xrightarrow{p}^* \text{abort} \quad C \text{ is tame}}{\text{atomic}\{C\}, (s, h) \xrightarrow[p]{\text{atomic}} \text{abort}} \text{ ATOM-ABT} \\
\frac{C, (s, h) \xrightarrow{p}^* \dots \quad C \text{ is tame}}{\text{atomic}\{C\}, (s, h) \xrightarrow[p]{\text{atomic}} \text{diverge}, (s, h)} \text{ ATOM-LOOP} \\
\frac{C_1, (s, h) \xrightarrow[p]{C_{1,a}} C'_1, (s', h')}{C_1 \parallel C_2, (s, h) \xrightarrow[p]{C_{1,a}} C'_1 \parallel C_2, (s', h')} \text{ CON-L} \qquad \frac{C_2, (s, h) \xrightarrow[p]{C_{2,a}} C'_2, (s', h')}{C_1 \parallel C_2, (s, h) \xrightarrow[p]{C_{2,a}} C_1 \parallel C'_2, (s', h')} \text{ CON-R} \\
\frac{C_1, (s, h) \xrightarrow[p]{C_{1,a}} \text{abort}}{C_1 \parallel C_2, (s, h) \xrightarrow[p]{C_{1,a}} \text{abort}} \text{ CON-L-ABT} \qquad \frac{C_2, (s, h) \xrightarrow[p]{C_{2,a}} \text{abort}}{C_1 \parallel C_2, (s, h) \xrightarrow[p]{C_{2,a}} \text{abort}} \text{ CON-R-ABT} \\
\frac{}{\downarrow \parallel \downarrow, (s, h) \xrightarrow[\text{con-end}]{1} \downarrow, (s, h)} \text{ CON-END}
\end{array}$$

■ **Figure 4** Operational semantics of concurrent control-flow operations in chpGCL.

tame programs inside atomic regions. A tame program does not require any (scheduling) actions since its Markov model is fully probabilistic. To formally define the syntax used in the inference rules for atomic regions, we first need to introduce *schedulers*, which are used to resolve non-determinism in an MDP. There are various classes of schedulers, and indeed we will later allow the use of different classes. However, we do require that all schedulers are deterministic and may have a history. This especially rules out any randomised scheduler, which would be an interesting topic, but is out of scope for the results presented here. Our schedulers use finite sequences of MDP states as histories.

► **Definition 3.4 (Scheduler).** *A scheduler is a mapping $\mathfrak{s}: U^+ \rightarrow A$ from histories of states to enabled actions, i.e., $\mathfrak{s}(\sigma_1 \dots \sigma_n) \in \text{Act}(\sigma_n)$. We denote the set of all schedulers by \mathbb{S} .*

For final states σ' (i.e., with program \downarrow or **abort**) and an MDP $(U, \text{Act}, \mathbb{P})$, we define

$$\text{reach}(n, \sigma_1, \mathfrak{s}, \sigma') = \sum \left[\prod_{i=1}^{m-1} \mathbb{P}(\sigma_i, \mathfrak{s}(\sigma_1 \dots \sigma_i))(\sigma_{i+1}) \right. \\
\left. \left| \sigma_1 \dots \sigma_m \in U^m, \sigma_m = \sigma', m \leq n \right. \right], \quad (1)$$

$$\sigma \xrightarrow[\mathfrak{s}]{p}^* \sigma' \quad \text{iff} \quad p = \lim_{n \rightarrow \infty} \text{reach}(n, \sigma, \mathfrak{s}, \sigma'), \quad (2)$$

$$\sigma \xrightarrow[\mathfrak{s}]{1-p}^* \dots \quad \text{iff} \quad p = \sum_{\sigma' \text{ final}} \lim_{n \rightarrow \infty} \text{reach}(n, \sigma, \mathfrak{s}, \sigma'). \quad (3)$$

For a function f and a predicate b , we write $[f(x) \mid x \in b]$ for the bag consisting of the values $f(x)$ with $x \in b$. We use notation (1) to calculate the probability to reach the final state σ' from σ_1 in at most n steps w.r.t \mathfrak{s} . We unroll the MDP here into the Markov Chain induced by \mathfrak{s} after at most n steps (cf. [2, Definition 10.92]). With notation (2), we define the reachability probability of a final state and with notation (3), we define the probability of non-termination. We avoid reasoning about uncountable sets of paths in case of non-termination by taking the probability to not reach a final state, i.e., a state with program \downarrow or **abort**. A scheduler \mathfrak{s} is unique if for every state $\sigma \in U$ there is at most one enabled action \mathfrak{s} can map to, i.e., $|\text{Act}(\sigma)| \leq 1$. In that case, we usually omit the corresponding transition label.

To reason about the operational semantics using QSL, we use weakest *liberal* preexpectations [5, 38], which take the greatest lower bound of the expected value with respect to a postexpectation together with the probability of non-termination for all schedulers that we

want to consider. We allow subsets of schedulers $S \subseteq \mathbb{S}$ in order to apply fairness conditions. Later, we only consider the complete set of schedulers. In that case, we omit the superscript from the function wlp , which is defined in the following.

► **Definition 3.5** (Weakest Liberal Preexpectation). *For a program C and an expectation X , we define the weakest liberal preexpectation with respect to a set of schedulers $\emptyset \neq S \subseteq \mathbb{S}$ as*

$$\text{wlp}^S \llbracket C \rrbracket (X)(s, h) = \inf \left\{ \sum \left[p \cdot X(s', h') \mid C, (s, h) \xrightarrow[\mathfrak{s}]{p}^* \downarrow, (s', h') \right] + p_{div} \right. \\ \left. \mid \mathfrak{s} \in S \text{ and } C, (s, h) \xrightarrow[\mathfrak{s}]{p_{div}}^* \dots \right\}.$$

► **Example 3.6.** For program C in Example 3.2, we evaluate (without proof) $\text{wlp} \llbracket C \rrbracket ([y = 0]) = \text{wlp}^{\mathbb{S}} \llbracket C \rrbracket ([y = 0]) = 0.5 \star [r \mapsto -]$. That is, if r is allocated, then the likelihood of C terminating without aborting in a state in which y equals 0 is 0.5, and zero otherwise. We will prove that this is a lower bound in Example 4.2.

4 Weakest Safe Liberal Preexpectations

For sequential probabilistic programs, a backwards expectation transformer can be defined to compute wlp [5]. This is not feasible for concurrent programs due to the non-locality of shared memory. Instead, we drop exact computation in our approach and reason about lower bounds of wlp by using inference rules similar to Hoare triples. To support shared memory, we furthermore introduce a modified version of wlp – the weakest *resource-safe* liberal preexpectation. The general idea as inspired by [53] is to prove that the shared memory is invariant with respect to a qualitative expectation, which we call a *resource invariant*. In other words, the shared memory is proven to be *safe* with respect to the resource invariant. We archive this by enforcing that at every point in the program’s execution (except for executions in atom regions), some part of the heap is satisfied by the resource invariant. In Example 4.2 we use the resource invariant $\max \{ [r \mapsto 0], [r \mapsto -1] \}$ to prove the lower bound from Example 3.6. We enforce that the program states do not include the shared memory any more, the transitions however are taken with any possible shared memory.

► **Definition 4.1** (Weakest Resource-Safe Liberal Preexpectation). *We first consider the expectation after one step with respect to a mapping from programs to expectations, that is, for a program C and a mapping $t: \text{chpGCL} \rightarrow \mathbb{E}_{\leq 1}$, we define*

$$\text{step} \llbracket C \rrbracket (t)(s, h) = \inf \left\{ \sum \left[p \cdot t(C')(s', h') \mid C, (s, h) \xrightarrow[a]{p} C', (s', h') \right] \right. \\ \left. \mid a \in \text{Act}(C, (s, h)) \right\}.$$

We define the weakest resource-safe liberal preexpectation after n steps for a program C , a postexpectation X and a (qualitative) resource invariant ξ as

$$\text{wrlp}_n \llbracket C \rrbracket (X \mid \xi) = \begin{cases} 1 & \text{if } n = 0 \\ X & \text{if } n \neq 0 \text{ and } C = \downarrow \\ \xi \multimap \text{step} \llbracket C \rrbracket (\lambda C'. \text{wrlp}_{n-1} \llbracket C' \rrbracket (X \mid \xi) \star \xi) & \text{otherwise}^1. \end{cases}$$

¹ We use $\lambda C'. X$ for the function which, when applied to the argument C , reduces to X in which every occurrence of C' in X is replaced by C .

Finally, we define the weakest resource-safe liberal preexpectation for arbitrarily many steps as

$$\text{wrlp}[[C]](X \mid \xi) = \lim_{n \rightarrow \infty} \text{wrlp}_n[[C]](X \mid \xi) .$$

An important observation is that for the special resource invariant **[emp]**, wlp and wrlp coincide (cf. [16]). This enables us to reason about lower bounds for probabilities of qualitative preconditions (and in general lower bounds for the expected value of one-bounded random variables). When reasoning about such probabilities, we first express a property for which we aim to prove a lower bound on wlp , afterwards we can transform it into wrlp with the resource invariant **[emp]** and use special rules to enrich the resource invariant with more information. The resource invariant should always cover all possible states that the shared memory may be in at any time during the program's execution. It is fine if the resource invariant is violated during executions of atomic regions, since we only care about safeness during executions with inferences between threads.

We mention that wrlp is heavily inspired by [53]. We formalise the connection between Vafeiadis' Concurrent Separation Logic and our weakest resource-safe liberal preexpectation below. In [53] a judgement is defined by a safe predicate that is similar to how we defined wrlp .

► **Definition 4.1** (Safe Judgements [53]). *The predicate $\text{safe}_n(C, s, h, \xi, \varphi)$ holds for qualitative φ and ξ and non-probabilistic program C if and only if*

1. *if $n = 0$, then it holds always; and*
2. *if $n > 0$ and $C = \downarrow$, then $\varphi(s, h) = 1$; and*
3. *if $n > 0$ and for all h_ξ and h_F with $\xi(s, h_\xi) = 1$ and $h \perp h_\xi \perp h_F$, then for all enabled actions $a \in \text{Act}(C, (s, h \star h_\xi \star h_F))$ we do not have $C, (s, h \star h_\xi \star h_F) \xrightarrow{a} \text{abort}$; and*
4. *if $n > 0$ and for all h_ξ, h_F, C', s and h , with $\xi(s, h_\xi) = 1$, and $h \perp h_\xi \perp h_F$, and $C, (s, h \star h_\xi \star h_F) \xrightarrow{a} C', (s', h')$, then there exists h'' and h'_ξ such that $h' = h'' \star h'_\xi \star h_F$ and $\xi(s', h'_\xi) = 1$ and $\text{safe}_{n-1}(C', s', h'', \xi, \varphi)$.*

For qualitative φ, ψ and ξ , we say that $\xi \models \{\varphi\} C \{\psi\}$ holds if and only if for all stack/heap pairs s, h the statement $\psi(s, h) = 1 \Rightarrow \forall n \in \mathbb{N}. \text{safe}_n(C, s, h, \xi, \varphi)$ holds.

A program C is framing enabled² if we can always extend the heap without changing the behaviour of C .

► **Definition 4.2** (Framing Enabledness). *A non-probabilistic program C is framing enabled if for all heaps h_F with $h \perp h_F$ and all enabled actions $a \in \text{Act}(C, (s, h \star h_F))$, it holds: if $C, (s, h) \xrightarrow{a} C', (s', h')$, then also $C, (s, h \star h_F) \xrightarrow{a} C', (s', h' \star h_F)$.*

The next theorem states that wrlp is a conservative extension of **safe**.

► **Theorem 4.1** (Conservative Extension of Concurrent Separation Logic). *For a framing-enabled non-probabilistic program C and qualitative expectations φ, ψ and ξ , we have*

$$\varphi \leq \text{wrlp}[[C]](\psi \mid \xi) \quad \text{iff} \quad \xi \models \{\varphi\} C \{\psi\} .$$

Proof. See [16]. ◀

² Indeed every non-probabilistic **chpGCL** program is framing enabled (cf. [16]).

$$\begin{array}{c}
\frac{}{X \leq \text{wrlp}[\downarrow](X \mid \xi)} \text{term} \\
\frac{Y \leq \sup_{v \in \mathbb{Z}} [e \mapsto v] \star ([e \mapsto v] \multimap X[x := v])}{Y \leq \text{wrlp}[x := \langle e \rangle](X \mid \xi)} \text{look} \\
\frac{Y \leq \inf_{v \in \mathbb{Z}} [v \mapsto e_1, \dots, e_n] \multimap X[x := v]}{Y \leq \text{wrlp}[x := \text{new}(e_1, \dots, e_n)](X \mid \xi)} \text{alloc}
\end{array}
\qquad
\begin{array}{c}
\frac{Y \leq X[x := e]}{Y \leq \text{wrlp}[x := e](X \mid \xi)} \text{assign} \\
\frac{Y \leq [e \mapsto -] \star ([e \mapsto e'] \multimap X)}{Y \leq \text{wrlp}[\langle e \rangle := e'](X \mid \xi)} \text{mut} \\
\frac{Y \leq X \star [x \mapsto -]}{Y \leq \text{wrlp}[\text{free}(x)](X \mid \xi)} \text{disp}
\end{array}$$

■ **Figure 5** Proof rules for `wrlp` for basic commands.

$$\begin{array}{c}
\frac{X \leq \text{wrlp}[C_1](Y \mid \xi) \quad Y \leq \text{wrlp}[C_2](Z \mid \xi)}{X \leq \text{wrlp}[C_1; C_2](Z \mid \xi)} \text{seq} \\
\frac{X_1 \leq \text{wrlp}[C_1](Y \mid \xi) \quad X_2 \leq \text{wrlp}[C_2](Y \mid \xi)}{[b] \cdot X_1 + [-b] \cdot X_2 \leq \text{wrlp}[\text{if}(b) \{C_1\} \text{else} \{C_2\}](Y \mid \xi)} \text{if} \\
\frac{I \leq [b] \cdot X + [-b] \cdot Y \quad X \leq \text{wrlp}[C](I \mid \xi)}{I \leq \text{wrlp}[\text{while}(b) \{C\}](Y \mid \xi)} \text{while} \qquad \frac{}{X \leq \text{wrlp}[\text{diverge}](Y \mid \xi)} \text{div} \\
\frac{X_1 \leq \text{wrlp}[C_1](Y \mid \xi) \quad X_2 \leq \text{wrlp}[C_2](Y \mid \xi)}{e_p \cdot X_1 + (1 - e_p) \cdot X_2 \leq \text{wrlp}[\{C_1\} [e_p] \{C_2\}](Y \mid \xi)} \text{p-choice} \qquad \frac{X \leq \text{wrlp}[C](Y \star \xi \mid \text{emp})}{X \leq \text{wrlp}[\text{atomic} \{C\}](Y \mid \xi)} \text{atomic} \\
\frac{X \leq \text{wrlp}[C](Y \mid \xi \star \pi)}{X \star \pi \leq \text{wrlp}[C](Y \star \pi \mid \xi)} \text{share} \\
\frac{X_1 \leq \text{wrlp}[C_1](Y_1 \mid \xi) \quad X_2 \leq \text{wrlp}[C_2](Y_2 \mid \xi) \quad \forall i \in \{1, 2\} \text{Write}(C_i) \cap \text{Vars}(C_{3-i}, Y_{3-i}, \xi) = \emptyset}{X_1 \star X_2 \leq \text{wrlp}[C_1 \parallel C_2](Y_1 \star Y_2 \mid \xi)} \text{concur}
\end{array}$$

■ **Figure 6** Proof rules for `wrlp` for control-flow commands.

We define `wrlp` inductively by means of a number of inference rules. We do not use classic Hoare triples due to difficulties arising when interpreting a `wrlp` statement forward. These difficulties are due to Jones's counterexample [29, p. 135]: Given the constant preexpectation 0.5 and the program $C: \{x := 0\} [0.5] \{x := 1\}$, what is the postexpectation? Two possible answers are $0.5 = \text{wlp}[C]([x = 0])$ and $0.5 = \text{wlp}[C]([x = 1])$, but a combination of both is not possible. For this reason, we highlight the backwards interpretation of our judgements by writing them as $X \leq \text{wrlp}[C](Y \mid \xi)$, where X is a (lower bound for the weakest liberal) preexpectation, C is the program, Y is the postexpectation and ξ is the resource invariant.

For basic commands, as shown in Figure 5, we can just re-use the QSL proof rules for weakest liberal preexpectations (`wlp`) of non-concurrent programs, as given in [5]. However, for `wrlp` these proof rules only allow lower bounding the preexpectation since we do not want to reason about the resource invariant if not necessary.

For commands handling control flow, as shown in Figure 6, we use mostly standard rules. Atomic regions regain access to the resource invariant. The share rule allows us to enrich the resource invariant. The rule for concurrency enforces that only local variables or read-only variables are used in each thread. One could as well allow shared variables that are owned by the resource invariant. However, for the sake of brevity we do not include this here.

We also introduce several proof rules that make reasoning easier, see Figure 7. A program is *almost surely terminating* with respect to a set of schedulers if the program terminates with probability one for every initial state and every scheduler in this set. Even though there is a plethora of work on almost-sure termination for (sequential) probabilistic programs (cf. [25] for an overview), techniques for checking almost-sure termination in a concurrent setting are sparse [22, 23, 36, 52]. Here, interpreting probabilistic choice as non-determinism and proving sure termination instead using techniques such as [15, 49] is an alternative.

25:12 Towards Concurrent Quantitative Separation Logic

$$\begin{array}{c}
\frac{X' \leq \text{wlp}^S[C](X) \quad Y' \leq \text{wlp}^S[C](Y) \quad C \text{ is AST w.r.t. } S \quad a \in \mathbb{R}_{\geq 0}}{a \cdot X' + Y' \leq \text{wlp}^S[C](a \cdot X + Y)} \text{superlin} \\
\frac{X \leq \text{wrlp}[C](Y \mid \mathbf{emp})}{X \leq \text{wlp}^S[C](Y)} \text{wlp-wrlp} \quad \frac{X \leq \text{wrlp}[C](Y \mid \xi) \quad \text{Write}(C) \cap \text{Vars}(Z) = \emptyset}{X \star Z \leq \text{wrlp}[C](Y \star Z \mid \xi)} \text{frame} \\
\frac{X \star \xi \leq \text{wrlp}[C](Y \star \xi \mid \mathbf{emp}) \quad C \text{ is a terminating atom}}{X \leq \text{wrlp}[C](Y \mid \xi)} \text{atom} \\
\frac{X \leq X' \quad X' \leq \text{wrlp}[C](Y' \mid \xi) \quad Y' \leq Y}{X \leq \text{wrlp}[C](Y \mid \xi)} \text{monotonic} \\
\frac{X \leq \text{wrlp}[C](Y \mid \xi) \quad X' \leq \text{wrlp}[C](Y' \mid \xi)}{\max\{X, X'\} \leq \text{wrlp}[C](\max\{Y, Y'\} \mid \xi)} \text{max} \\
\frac{X \leq \text{wrlp}[C](Y \mid \xi) \quad X' \leq \text{wrlp}[C](Y' \mid \xi) \quad \xi \text{ precise}}{\min\{X, X'\} \leq \text{wrlp}[C](\min\{Y, Y'\} \mid \xi)} \text{min} \\
\frac{X \leq \text{wrlp}[C](Y \mid \xi) \quad X' \leq \text{wrlp}[C](Y' \mid \xi) \quad \xi \text{ precise} \quad \text{Write}(C) \cap \text{Vars}(e) = \emptyset}{e \cdot X + (1 - e) \cdot X' \leq \text{wrlp}[C](e \cdot Y + (1 - e) \cdot Y' \mid \xi)} \text{convex}
\end{array}$$

■ **Figure 7** Auxiliary proof rules for wrlp.

The first rule in Figure 7 uses superlinearity to split a given postexpectation into a sum of postexpectations, for which proving a lower bound on the preexpectation might be easier. We only allow the use of superlinearity for wlp (and not for wrlp) because we need a restricted set of schedulers to enforce fairness conditions. Fairness conditions are required to reason about termination for concurrent programs with some sort of blocking behaviour. We are then able to transform wlp into wrlp by using the wlp-wrlp rule. Whether wrlp can also be defined with fairness conditions in mind and thus applying superlinearity directly on wrlp, is an open question. The frame rule is of central importance to the Separation Logic approach, as it supports local reasoning about only the relevant part of the heap [46]. The atom rule can be used similarly to the rule for atomic regions. Monotonicity is the quantitative version of the rule of consequence and is used to reduce and increase the post- and preexpectation respectively. The max, min and convex rules eliminate max, min and convex sum operations, respectively. The min and convex rule require preciseness of the resource invariant – similarly to how [53] required preciseness for the conjunction rule. An expectation is *precise* if for any stack there is at most one heap for which the expectation is not zero. For the min rule, this is not surprising as the minimum behaves like conjunction in case of qualitative expectations. Requiring preciseness also for the convex rule is due to the missing superlinearity of the separating multiplication for non-precise expectations.

► **Theorem 4.2** (Soundness of proof rules). *For every proof rule in Figures 5–7 it holds that if their premises hold, the conclusion holds as well.*

Proof. See [16]. ◀

► **Example 4.2.** We are now able to establish the lower bound computed in Example 3.6 using the proof rules. Instead of constructing a proof tree by composing inference rules, we annotate program locations with their respective pre- and postexpectations. The interpretation is standard; for preexpectation X , postexpectation Y , resource invariant ξ and program C :

$$\begin{array}{l}
\parallel X \mid \xi \\
C \quad \text{iff} \quad X \leq \text{wrlp}[C](Y \mid \xi) \\
\parallel Y \mid \xi
\end{array}$$

Proofs in this style should only be read backwards from bottom to top. They will not include applications of the proof rules for atomic programs and of the share rule as this may lead to incorrect interpretations. For our example, we use the resource invariant $\xi = \max \{ [r \mapsto 0], [r \mapsto -1] \}$, which we guessed by collecting all possible values stored in location r during executions yielding our postexpectation. We assume that the memory of the initial heap h only contains a single location r with value -1 . This assumption is reflected by the resource invariant and will only allow us to reason about executions with such an initial heap. The other possible value for the location r is 0, since the left program may mutate the heap. Indeed, there are also executions where the value of location r is 1. For these, the program only terminates in states violating the postexpectation $[y = 0]$. We can further show:

$$\begin{array}{l}
 \llbracket 0.5 \star 1 \mid \xi \\
 \\
 \llbracket 0.5 \mid \xi \\
 \{ \langle r \rangle := 0 \} [0.5] \{ \langle r \rangle := 1 \} \\
 \llbracket 1 \mid \xi \\
 \\
 \llbracket 1 \star [y = 0] \mid \xi
 \end{array}
 \left\| \begin{array}{l}
 \llbracket 1 \mid \xi \\
 y := \langle r \rangle ; \\
 \llbracket \max \{ [y = 0], [y = -1] \} \mid \xi \\
 \mathbf{while} (y = -1) \{ y := \langle r \rangle \} ; \\
 \llbracket [y = 0] \mid \xi
 \end{array} \right.$$

To handle concurrency, we separate our postexpectation into the expectation 1 for the left program and the expectation $[y = 0]$ for the right program. The left program includes a probabilistic choice, for which we use the atom rule to infer that the preexpectation is 1 in the left branch of the probabilistic choice, as the resource invariant allows mutating the value of location r to 1 and the resource invariant can be re-established since we can lower bound all possible values for the location r that are not -1 or 0 to zero. Moreover, we lower bound the right branch of the probabilistic choice by zero, because zero is a lower bound of any expectation. The right program iterates until the value of location r has been mutated. Our resource invariant contains all possible values that the program can expect here. For the loop invariant, we connect all possible values of y using a disjunction over 0 and -1 , as we disregard executions where $y = 1$. Lastly, we can apply the loop invariant to the lookup of r and since this matches our resource invariant, the resulting preexpectation is one.

Thus, we have established that $0.5 \leq \mathbf{wrlp} \llbracket C \rrbracket ([y = 0] \mid \xi)$. Using the share rule we can further infer that $0.5 \star \xi \leq \mathbf{wrlp} \llbracket C \rrbracket ([y = 0] \star \xi \mid \mathbf{emp})$. Lastly, we can clean up the statement using monotonicity and the $\mathbf{wlp-wrlp}$ rule to obtain $0.5 \star \xi \leq \mathbf{wlp} \llbracket C \rrbracket ([y = 0])$. For details on the probabilistic choice and the loop invariant, we refer to Appendix A.1.

5 Example: A Producer, a Consumer and a Lossy Channel

A producer-consumer system is often used when presenting verification techniques for concurrent programs. We continue this tradition, extending this example by probabilistic elements, see Figure 8. Video and audio streaming is an example for such a system, where data losses are acceptable if they do not exceed a certain limit. Moreover, by enriching the resource invariant with a predicate defining an appropriate data structure, this example can be used as a template to reason about systems communicating using a shared data structure. We consider a producer that randomly generates data (1 or 2) and stores it in an array of

25:14 Towards Concurrent Quantitative Separation Logic

```

l := 0;
y1, y2, y3 := k;

while (y1 ≥ 0) {
  { x1 := 1 } [0.5] { x1 := 2 };
  < z1 + y1 > := x1;
  y1 := y1 - 1
}

||

while (y2 ≥ 0) {
  x2 := < z1 + y2 >;
  if (x2 ≠ 0) {
    { < z2 + y2 > := x2 }
    [p]
    { < z2 + y2 > := -1 };
  }
  y2 := y2 - 1
}

||

while (y3 ≥ 0) {
  x3 := < z2 + y3 >;
  if (x3 ≠ 0) {
    if (x3 ≠ -1) { l := l + 1 };
    y3 := y3 - 1
  }
}

```

■ **Figure 8** A program consisting of the tree threads: a producer (left), a consumer (right) and lossy channel (middle) for communication between the prior threads.

size k indexed by z_1 . The data has to be transferred to a consumer. However, the consumer does not have direct access to the array maintained by the producer. Instead a third party, the lossy channel, transfers data from the array maintained by the producer to a different k -sized array that is indexed by z_2 , and that can be accessed by the consumer. However, the channel is not reliable. With a probability of $1 - p$, it loses a value and instead stores invalid data (encoded as -1) at the respective array position. The consumer discards invalid data and counts in l how many valid elements it received until all array elements have been attempted to be transmitted once. For the sake of brevity, we leave out the allocation of the array index z_1 and z_2 . Instead, we assume already allocated arrays as input.

We are interested in the probability that the data of a certain set of locations has been successfully transmitted. If we additionally prove that the program is almost surely terminating for some reasonable set of fair schedulers, we can use superlinearity to prove lower bounds of probabilities for even more complex postconditions, e.g. the probability that at least half of the data have been transmitted successfully. Indeed, the program is almost surely terminating under a fairness condition. We denote the set of locations that we want to be successfully transmitted as J . For the resource invariant, we use a big separating multiplication. Its semantics is as expected: for a stack s , we connect all choices for the index variable with regular separating multiplications. The resource invariant describes the values we want to tolerate for every entry in both arrays. We join the tolerated values by a disjunction (which is the maximum in our case). We now use the resource invariant, parametrised on the set J as

$$\begin{aligned}
\xi_J &= \left(\star_{i \in \{0, \dots, k\}} \max \{ [z_1 + i \mapsto 0], [z_1 + i \mapsto 1], [z_1 + i \mapsto 2] \} \right) \\
&\star \left(\star_{i \in \{0, \dots, k\} \cap J} \max \{ [z_2 + i \mapsto 0], [z_2 + i \mapsto 1], [z_2 + i \mapsto 2] \} \right) \\
&\star \left(\star_{i \in \{0, \dots, k\} \setminus J} \max \{ [z_2 + i \mapsto 0], [z_2 + i \mapsto -1] \} \right).
\end{aligned}$$

Next, we can use the resource invariant to prove an invariant for each of the three concurrent programs. The corresponding calculations can be found in Appendix A.2. Let C_1 be the producer, C_2 the channel and C_3 the consumer. For the producer program C_1 we can prove the invariant $I_1 = 1$ with respect to the postexpectation 1, for the channel

C_2 we can prove the invariant $I_2 = [0 \leq y_2 \leq k] \cdot p^{|\{0, \dots, y_2\} \cap J|} \cdot (1 - p)^{|\{0, \dots, y_2\} \setminus J|} + [y_2 < 0]$ with respect to the postexpectation 1, and for the consumer program C_3 we can prove the invariant $I_3 = [0 \leq y_3 \leq k] \cdot [l = |J \cap \{0, \dots, y_3\}|] + [y_3 < 0] \cdot [l = |J|]$ with respect to the postexpectation $[l = |J|]$. Using all three invariants, we can now lower bound the probability that $l = |J|$ holds after the execution of the whole program C in Figure 8:

$$\begin{aligned} & \llbracket [0 \leq k] \cdot p^{|\{0, \dots, k\} \cap J|} \cdot (1 - p)^{|\{0, \dots, k\} \setminus J|} \mid \xi_J \\ & l := 0; \\ & y_1, y_2, y_3 := k; \\ & \llbracket I_1 \star I_2 \star I_3 \mid \xi_J \\ & C_1 \parallel C_2 \parallel C_3 \\ & \llbracket 1 \star 1 \star [l = |J|] \mid \xi_J \end{aligned}$$

Here, we first use the concurrency rule to place the postexpectation $[l = |J|]$ into a separating context, thus covering all three programs. The resulting preexpectation is indeed the separating multiplication of the respective invariants. By applying the assignment rules to the first two rows, we finally get the result for a lower bound of the weakest resource-safe preexpectation with respect to resource invariant ξ_J . Thus, the lower bound $([0 \leq k] \cdot p^{|\{0, \dots, k\} \cap J|} \cdot (1 - p)^{|\{0, \dots, k\} \setminus J|}) \star \xi_J \leq \text{wlp}^S[C]([l = |J|] \star \xi_J)$ also holds. We also show in Appendix A.2 how to prove the lower bound of more difficult postexpectations using superlinearity.

6 Conclusion and Future Work

Using resource invariants from Concurrent Separation Logic [53] together with quantitative reasoning from Quantitative Separation Logic [5] allows us to reason about lower-bound probabilities of realizing a postcondition. In our technique, probability mass is local to the thread. This insight gave rise to only allow qualitative expectations in the model of the environment. By this, the resource invariant only describes shared memory and lacks semantics for global probability mass.

However, we may favour a probabilistic model of the environment – for example, if the environment is a black box and only statistic information about its possible behaviours is available. More research is required for logics allowing probabilistic specifications in the environment description, especially logics allowing quantitative resource invariants. Moreover, we are only able to verify lower bounds due to the concurrent rule. We conjecture that a logic for upper bounds requires different, unknown separation connectives.

References

- 1 Christel Baier, Frank Ciesinski, and Markus Grosser. PROBMELA: a modeling language for communicating probabilistic processes. In *MEMOCODE*, pages 57–66. IEEE, 2004.
- 2 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. The MIT Press, 2008.
- 3 Gilles Barthe, Benjamin Grégoire, and Santiago Zanella Béguelin. Probabilistic relational Hoare logics for computer-aided security proofs. In *MPC*, pages 1–6. Springer, 2012.
- 4 Kevin Batz, Ira Fesefeldt, Marvin Jansen, Joost-Pieter Katoen, Florian Kessler, Christoph Matheja, and Thomas Noll. Foundations for entailment checking in quantitative separation logic. In *ESOP*, pages 57–84. Springer, 2022.
- 5 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.

- 6 Josh Berdine, Cristiano Calcagno, Byron Cook, Dino Distefano, Peter W. O’Hearn, Thomas Wies, and Hongseok Yang. Shape analysis for composite data structures. In *CAV*, pages 178–192. Springer, 2007.
- 7 Josh Berdine, Cristiano Calcagno, and Peter W. O’Hearn. Smallfoot: Modular automatic assertion checking with separation logic. In *FMCO*, pages 115–137. Springer, 2005.
- 8 Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
- 9 Stephen Brookes and Peter W. O’Hearn. Concurrent separation logic. *ACM SIGLOG News*, 3(3):47–65, 2016.
- 10 Cristiano Calcagno, Dino Distefano, Peter W. O’Hearn, and Hongseok Yang. Compositional shape analysis by means of bi-abduction. *J. ACM*, 58(6):26:1–26:66, 2011.
- 11 Michael Carbin, Sasa Misailovic, and Martin C. Rinard. Verifying quantitative reliability for programs that execute on unreliable hardware. *Commun. ACM*, 59(8):83–91, 2016.
- 12 Edsger W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.
- 13 Thomas Dinsdale-Young, Lars Birkedal, Philippa Gardner, Matthew Parkinson, and Hongseok Yang. Views: Compositional reasoning for concurrent programs. In *POPL*, pages 287–300. ACM, 2013.
- 14 Thomas Dinsdale-Young, Mike Dodds, Philippa Gardner, Matthew J. Parkinson, and Viktor Vafeiadis. Concurrent abstract predicates. In *ECOOP*, pages 504–528. Springer, 2010.
- 15 Emanuele D’Osualdo, Julian Sutherland, Azadeh Farzan, and Philippa Gardner. TaDA live: Compositional reasoning for termination of fine-grained concurrent programs. *ACM Trans. Program. Lang. Syst.*, 43(4), 2021.
- 16 Ira Fesefeldt, Joost-Pieter Katoen, and Thomas Noll. Towards concurrent quantitative separation logic. *CoRR*, abs/2207.02822, 2022.
- 17 Keir Fraser. Practical lock-freedom. Technical Report UCAM-CL-TR-579, University of Cambridge, Computer Laboratory, 2004.
- 18 Ming Fu, Yong Li, Xinyu Feng, Zhong Shao, and Yu Zhang. Reasoning about optimistic concurrency using a program logic for history. In *CONCUR*, pages 388–402. Springer, 2010.
- 19 Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In *FOSE*, pages 167–181. ACM, 2014.
- 20 Alexey Gotsman, Josh Berdine, Byron Cook, and Mooly Sagiv. Thread-modular shape analysis. In *PLDI*, pages 266–277. ACM, 2007.
- 21 Friedrich Gretz, Joost-Pieter Katoen, and Annabelle McIver. Operational versus weakest pre-expectation semantics for the probabilistic guarded command language. *Performance Evaluation*, 73:110–132, 2014.
- 22 Sergiu Hart and Micha Sharir. Concurrent probabilistic programs, or: How to schedule if you must. *SIAM J. Comput.*, 14(4):991–1012, 1985.
- 23 Sergiu Hart, Micha Sharir, and Amir Pnueli. Termination of probabilistic concurrent programs. *ACM Trans. Program. Lang. Syst.*, 5(3):356–380, 1983.
- 24 Max P. L. Haslbeck. *Verified Quantitative Analysis of Imperative Algorithms*. PhD thesis, Technical University of Munich, Germany, 2021.
- 25 Mingzhang Huang, Hongfei Fu, Krishnendu Chatterjee, and Amir Kafshdar Goharshady. Modular verification for almost-sure termination of probabilistic programs. *Proc. ACM Program. Lang.*, 3(OOPSLA):129:1–129:29, 2019.
- 26 Samin S. Ishtiaq and Peter W. O’Hearn. BI as an assertion language for mutable data structures. In *POPL*, pages 14–26. ACM, 2001.
- 27 Kenneth E. Iverson. *A Programming Language*. John Wiley & Sons, Inc., USA, 1962.
- 28 Bart Jacobs, Jan Smans, Pieter Philippaerts, Frédéric Vogels, Willem Penninckx, and Frank Piessens. Verifast: A powerful, sound, predictable, fast verifier for C and Java. In *NFM*, pages 41–55. Springer, 2011.
- 29 Claire Jones. *Probabilistic Non-Determinism*. PhD thesis, University of Edinburgh, 1992.

- 30 Cliff B. Jones. Tentative steps toward a development method for interfering programs. *ACM Trans. Program. Lang. Syst.*, 5(4):596–619, 1983.
- 31 Ralf Jung, Robbert Krebbers, Jacques-Henri Jourdan, Ales Bizjak, Lars Birkedal, and Derek Dreyer. Iris from the ground up: A modular foundation for higher-order concurrent separation logic. *J. Funct. Program.*, 28:e20, 2018.
- 32 Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. Iris: Monoids and invariants as an orthogonal basis for concurrent reasoning. In *POPL*, pages 637–650. ACM, 2015.
- 33 Benjamin L. Kaminski, Joost-Pieter Katoen, Christoph Matheja, and Federico Olmedo. Weakest precondition reasoning for expected runtimes of randomized algorithms. *J. ACM*, 65(5), 2018.
- 34 Dexter Kozen. Semantics of probabilistic programs. In *FOCS*, pages 101–114. IEEE Computer Society, 1979.
- 35 Dexter Kozen. A probabilistic PDL. In *STOC*, pages 291–297. ACM, 1983.
- 36 Ondrej Lengál, Anthony Widjaja Lin, Rupak Majumdar, and Philipp Rümmer. Fair termination for parameterized probabilistic concurrent systems. In *TACAS*, pages 499–517. Springer, 2017.
- 37 Christoph Matheja. *Automated Reasoning and Randomization in Separation Logic*. PhD thesis, RWTH Aachen University, Germany, 2020.
- 38 Annabelle McIver and Carroll Morgan. Partial correctness for probabilistic demonic programs. *Theoretical Computer Science*, 266(1):513–541, 2001.
- 39 Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer, 2005.
- 40 Annabelle McIver, Tahiry Rabehaja, and Georg Struth. Probabilistic rely-guarantee calculus. *Theoretical Computer Science*, 655:120–134, 2016.
- 41 Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.
- 42 Peter Müller, Malte Schwerhoff, and Alexander J. Summers. Viper: A verification infrastructure for permission-based reasoning. In *Dependable Software Systems Engineering*, volume 50 of *NATO Science for Peace and Security Series - D: Information and Communication Security*, pages 104–125. IOS Press, 2017.
- 43 Aleksandar Nanevski, Ruy Ley-Wild, Ilya Sergey, and Germán Andrés Delbianco. Communicating state transition systems for fine-grained concurrent resources. In *ESOP*, pages 290–310. Springer, 2014.
- 44 Van Chan Ngo, Quentin Carbonneaux, and Jan Hoffmann. Bounded expectations: Resource analysis for probabilistic programs. *SIGPLAN Not.*, 53(4):496–512, 2018.
- 45 Peter W. O’Hearn. Resources, concurrency, and local reasoning. *Theoretical Computer Science*, 375(1):271–307, 2007.
- 46 Peter W. O’Hearn. Separation logic. *Commun. ACM*, 62(2):86–95, 2019.
- 47 Ruzica Piskac, Thomas Wies, and Damien Zufferey. Automating separation logic using SMT. In *CAV*, pages 773–789. Springer, 2013.
- 48 William Pugh. Skip lists: A probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990.
- 49 Tobias Reinhard and Bart Jacobs. Ghost signals: Verifying termination of busy waiting. In *CAV*, pages 27–50. Springer, 2021.
- 50 John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS*, pages 55–74. IEEE Computer Society, 2002.
- 51 Joseph Tassarotti and Robert Harper. A separation logic for concurrent randomized programs. *Proc. ACM Program. Lang.*, 3(POPL):64:1–64:30, 2019.
- 52 Michael L. Tiomkin. Probabilistic termination versus fair termination. *Theor. Comput. Sci.*, 66(3):333–340, 1989.
- 53 Viktor Vafeiadis. Concurrent separation logic and operational semantics. *Electronic Notes in Theoretical Computer Science*, 276:335–351, 2011.

A

 Details on Examples

A.1 Additional Details on the Running Example

To recap, we are given the resource invariant $\xi = \max \{ [r \mapsto 0], [r \mapsto -1] \}$ and have already proven:

$$\begin{array}{l}
 \parallel 0.5 \star 1 \mid \xi \\
 \\
 \left. \begin{array}{l}
 \parallel 0.5 \mid \xi \\
 \{ \langle r \rangle := 0 \} [0.5] \{ \langle r \rangle := 1 \} \\
 \parallel 1 \mid \xi
 \end{array} \right\| \left. \begin{array}{l}
 \parallel 1 \mid \xi \\
 \parallel 1 \star \xi \\
 y := \langle r \rangle ; \\
 \parallel \max \{ [y = 0], [y = -1] \} \star \xi \\
 \parallel \max \{ [y = 0], [y = -1] \} \mid \xi \\
 \mathbf{while} (y = -1) \{ y := \langle r \rangle \} ; \\
 \parallel [y = 0] \mid \xi
 \end{array} \right\| \\
 \\
 \parallel 1 \star [y = 0] \mid \xi
 \end{array}$$

The mutation $\langle r \rangle := -1$ together with the atom rule gives us the inequality

$$0.5 \star [r \mapsto -] \leq [r \mapsto -] \star ([r \mapsto -1] \multimap 0.5 \star \max \{ [r \mapsto 0], [r \mapsto -1] \}) .$$

However, it is easy to verify that $([r \mapsto -1] \multimap 0.5 \star \max \{ [r \mapsto 0], [r \mapsto -1] \})$ simplifies to 0.5, which results in the given lower bound.

For the probabilistic choice we have:

$$\begin{array}{l}
 \parallel 0.5 \mid \xi \\
 \parallel 0.5 \cdot 1 + 0.5 \cdot 0 \mid \xi \\
 \left\{ \begin{array}{l} \parallel 1 \mid \xi \\ \langle r \rangle := 0 \\ \parallel 1 \mid \xi \end{array} \right\} [0.5] \left\{ \begin{array}{l} \parallel 0 \mid \xi \\ \langle r \rangle := 1 \\ \parallel 1 \mid \xi \end{array} \right\} \\
 \parallel 1 \mid \xi
 \end{array}$$

The right part is rather simple, as we can always lower bound anything by zero. The left part holds since with $[r \mapsto 0]$ the mutation is satisfied, however the value before mutating r is unknown. We can lower bound the resulting preexpectation $1 \star [r \mapsto -]$ by $1 \star \max \{ [r \mapsto -1], [r \mapsto 0] \}$ and thus realise the resource invariant again. For the loop invariant $\max \{ [y = 0], [y = -1] \}$ we have:

$$\begin{array}{l}
 \parallel \max \{ [y = 0], [y = -1] \} \mid \xi \\
 y := \langle r \rangle \\
 \parallel \max \{ [y = 0], [y = -1] \} \mid \xi
 \end{array}$$

The lookup operation here results in both $[y = 0]$ and $[y = 1]$ to be evaluated to 1 if $[r \mapsto 0]$ and $[r \mapsto -1]$, respectively. Our resource invariant guarantees this, thus we obtain the expectation 1 and lower bound it by $\max \{ [y = 0], [y = -1] \}$. Lastly we check that it is indeed a loop invariant:

$$\begin{aligned}
 & [y = -1] \cdot \max \{ [y = 0], [y = -1] \} + [y \neq -1] \cdot [y = 0] \\
 &= [y = -1] + [y = 0] \\
 &= \max \{ [y = 0], [y = -1] \}
 \end{aligned}$$

A.2 Example: A Producer, a Consumer and a lossy Channel

Here we have the following program C :

```

l := 0;
y1, y2, y3 := k;

while (y1 ≥ 0) {
  { x1 := 1 } [0.5] { x1 := 2 };
  < z1 + y1 > := x1;
  y1 := y1 - 1
}

||

while (y2 ≥ 0) {
  x2 := < z1 + y2 >;
  if (x2 ≠ 0) {
    { < z2 + y2 > := x2 }
    [p]
    { < z2 + y2 > := -1 };
    y2 := y2 - 1
  }
}

||

while (y3 ≥ 0) {
  x3 := < z2 + y2 >;
  if (x3 ≠ 0) {
    if (x3 ≠ -1) { l := l + 1 };
    y3 := y3 - 1
  }
}

```

We use the resource invariant ξ_J for a set J . The set J encodes which locations in the array starting from z_2 will have an error value of -1 or a valid value of 1 or 2 after the channel inserts data into it. We use a big separating multiplication to connect all the possible instantiations using separating multiplication. That is, $\star\{X\} = X$ and $\star(\{X\} \cup A) = X \star \star A$ for a non-empty and countable set A . ξ_J declares that all locations between z_1 and $z_1 + k$ have either value 0 , 1 or 2 and all locations between z_2 and $z_2 + k$ have values 0 , 1 or 2 if the offset is in J and 0 or -1 if the offset is not in J . The value 0 is always possible for all locations between z_i and $z_i + k$ since we assume 0 to be the initial value. We connect the predicates declaring possible values for the location $z_j + i$ using a maximum, which acts as a qualitative disjunction here.

$$\begin{aligned}
\xi_J = & \left(\star_{i \in \{0, \dots, k\}} \max \{ [z_1 + i \mapsto 0], [z_1 + i \mapsto 1], [z_1 + i \mapsto 2] \} \right) \\
& \star \left(\star_{i \in \{0, \dots, k\} \cap J} \max \{ [z_2 + i \mapsto 0], [z_2 + i \mapsto 1], [z_2 + i \mapsto 2] \} \right) \\
& \star \left(\star_{i \in \{0, \dots, k\} \setminus J} \max \{ [z_2 + i \mapsto 0], [z_2 + i \mapsto -1] \} \right) .
\end{aligned}$$

We will leave out computations of inequalities $X \leq Y$ for the sake of brevity and give an explanation instead. Since our representation of expectations may grow in size, we use a curly bracket after the \llbracket symbol to denote expectations which are too long for one line. Our goal is to prove a lower bound on the probability that $l = |J|$ is realised after termination. If J contains numbers outside the range between 0 and k , we may as well replace $|J|$ with $|J \cap \{0, \dots, k\}|$. We now prove an invariant for each of the three subprograms.

25:20 Towards Concurrent Quantitative Separation Logic

For the producer C_1 , we prove the invariant $I_1 = 1$ with respect to the postexpectation 1 and resource invariant ξ_J . This shows indeed that the probability of safe execution of the loop is one and that our resource invariant ξ_J almost always holds.

$$\begin{array}{l}
\llcorner 1 \mid \xi_J \\
\left\{ \begin{array}{l} \llcorner 1 \mid \xi_J \\ \llcorner [1 \in \{0, \dots, 2\}] \mid \xi_J \\ x_1 := 1 \\ \llcorner [x_1 \in \{0, \dots, 2\}] \mid \xi_J \end{array} \right\} [0.5] \left\{ \begin{array}{l} \llcorner 1 \mid \xi_J \\ \llcorner [2 \in \{0, \dots, 2\}] \mid \xi_J \\ x_1 := 2 \\ \llcorner [x_1 \in \{0, \dots, 2\}] \mid \xi_J \end{array} \right\}; \\
\llcorner [x_1 \in \{0, \dots, 2\}] \mid \xi_J \\
\langle z_1 + y_1 \rangle := x_1; \\
\llcorner 1 \mid \xi_J \\
y_1 := y_1 - 1 \\
\llcorner 1 \mid \xi_J
\end{array}$$

The inequality

$$[x_1 \in \{0, \dots, 2\}] \star \xi_J \leq [z_1 + y_1 \mapsto -] \star ([z_1 + y_1 \mapsto x_1] \multimap (1 \star \xi_J))$$

resulting from the mutation $\langle z_1 + y_2 \rangle := x_1$ together with the atom rule holds because for $[z_1 + y_1 \mapsto x_1] \multimap (1 \star \xi_J)$ to be non-zero, x_1 must coincide with ξ_J – thus x_1 has to be either 0, 1 or 2. Furthermore, we have that $[z_1 + y_1 \mapsto i] \leq [z_1 + y_1 \mapsto -]$ holds for every i , and obtain by this that $i \in \{0, \dots, 2\}$, with which we re-establish ξ_J . We have $[y_1 \geq 0] \cdot 1 + [y_1 < 0] \cdot 1 = 1$ and therefore 1 is a loop invariant.

For the channel C_2 , we use the shorthand notation $P(y)$ to denote cumulated probability mass and define it as

$$P(y) = p^{|\{0, \dots, y\} \cap J|} \cdot (1 - p)^{|\{0, \dots, y\} \setminus J|}.$$

This shorthand notation gives us the probability that all data with offset 0 up to y are transferred according to J . That is, if an element should have been transferred successfully, we multiply with p and if not with $1 - p$ for every location up to y_2 . We prove for the invariant $I_2 = [0 \leq y_2 \leq k] \cdot P(y_2) + [y_2 < 0]$ with respect to the postexpectation 1 and resource invariant ξ_J :

$$\begin{array}{l}
\llcorner [0 \leq y_2 \leq k] \cdot P(y_2) + [y_2 < 0] \mid \xi_J \\
\llcorner \left\{ \begin{array}{l} [x_2 \neq 0] \cdot [0 \leq y_2 \leq k] \cdot P(y_2) \\ + [x_2 = 0] \cdot ([0 \leq y_2 \leq k] \cdot P(y_2) + [y_2 < 0]) \end{array} \right\} \mid \xi_J \\
x_2 := \langle z_1 + y_2 \rangle; \\
\llcorner \left\{ \begin{array}{l} [x_2 \neq 0] \cdot [0 \leq y_2 \leq k] \cdot P(y_2 - 1) \cdot (p \cdot [y_2 \in J] \cdot [x_2 \in \{1, \dots, 2\}] \\ + (1 - p) \cdot [y_2 \notin J]) \\ + [x_2 = 0] \cdot ([0 \leq y_2 \leq k] \cdot P(y_2) + [y_2 < 0]) \end{array} \right\} \mid \xi_J \\
\text{if } (x_2 \neq 0) \{ \\
\llcorner [0 \leq y_2 \leq k] \cdot P(y_2 - 1) \cdot (p \cdot [y_2 \in J] \cdot [x_2 \in \{0, \dots, 2\}] + (1 - p) \cdot [y_2 \notin J]) \mid \xi_J \\
\{ \\
\llcorner ([0 \leq y_2 \leq k] \cdot P(y_2 - 1)) \cdot [y_2 \in J] \cdot [x_2 \in \{0, \dots, 2\}] \mid \xi_J
\end{array}$$

$$\begin{aligned}
& \parallel ([1 \leq y_2 \leq k] \cdot P(y_2 - 1) + [y_2 = 0]) \cdot [y_2 \in J] \cdot [x_2 \in \{0, \dots, 2\}] \mid \xi_J \\
& \langle z_2 + y_2 \rangle := x_2 \\
& \parallel [1 \leq y_2 \leq k + 1] \cdot P(y_2 - 1) + [y_2 < 1] \mid \xi_J \\
& \} \\
& [p] \\
& \{ \\
& \parallel ([0 \leq y_2 \leq k] \cdot P(y_2 - 1)) \cdot [y_2 \notin J] \mid \xi_J \\
& \parallel ([1 \leq y_2 \leq k] \cdot P(y_2 - 1) + [y_2 = 0]) \cdot [y_2 \notin J] \mid \xi_J \\
& \langle z_2 + y_2 \rangle := -1; \\
& \parallel [1 \leq y_2 \leq k + 1] \cdot P(y_2 - 1) + [y_2 < 1] \mid \xi_J \\
& \} \\
& \parallel [1 \leq y_2 \leq k + 1] \cdot P(y_2 - 1) + [y_2 < 1] \mid \xi_J \\
& \parallel [0 \leq y_2 - 1 \leq k] \cdot P(y_2 - 1) + [y_2 - 1 < 0] \mid \xi_J \\
& y_2 := y_2 - 1 \\
& \parallel [0 \leq y_2 \leq k] \cdot P(y_2) + [y_2 < 0] \mid \xi_J \\
& \} \\
& \parallel [0 \leq y_2 \leq k] \cdot P(y_2) + [y_2 < 0] \mid \xi_J
\end{aligned}$$

We explain some of the difficult inequalities in the previous proof. We start with the inequality

$$\begin{aligned}
& (([1 \leq y_2 \leq k] \cdot P(y_2 - 1) + [y_2 = 0]) \cdot [y_2 \notin J]) \star \xi_J \\
& \leq [z_2 + y_2 \mapsto -] \star ([z_2 + y_2 \mapsto -1] \longrightarrow ([1 \leq y_2 \leq k + 1] \cdot P(y_2 - 1) + [y_2 < 1]) \star \xi_J
\end{aligned}$$

resulting from the mutation $\langle z_2 + y_2 \rangle := -1$ together with the atom rule. This inequality holds since for the part $[z_2 + y_2 \mapsto -1] \longrightarrow \dots$ to be non-zero, we require $y_2 \notin J$ due to ξ_J . We lower bound all evaluations where $y_2 < 0$ by 0 as we can not infer any information about these locations from ξ_J . Afterwards, we can lower bound $[z_2 + y_2 \mapsto -]$ by $[z_2 + y_2 \mapsto i]$ for every i and thus re-establish ξ_J .

Next we have the inequality

$$\begin{aligned}
& (([1 \leq y_2 \leq k] \cdot P(y_2 - 1) + [y_2 = 0]) \cdot [y_2 \in J] \cdot [x_2 \in \{0, \dots, 2\}]) \star \xi_J \\
& \leq [z_2 + y_2 \mapsto -] \star ([z_2 + y_2 \mapsto x_2] \longrightarrow ([1 \leq y_2 \leq k + 1] \cdot P(y_2 - 1) + [y_2 < 1]) \star \xi_J
\end{aligned}$$

resulting from the mutation $\langle z_2 + y_2 \rangle := x_2$ together with the atom rule. Here we assume that the location y_2 is in J and obtain that $x_2 \in \{0, \dots, 2\}$. We lower bound any outcome of y_2 not in J by 0 because we already know that we will eventually set the term to 0 due to the previous lookup. Next we establish ξ_J back from lower bounding $[z_2 + y_2 \mapsto -]$.

We have the inequality

$$\begin{aligned}
& [x_2 \neq 0] \cdot [0 \leq y_2 \leq k] \cdot P(y_2) + [x_2 = 0] \cdot ([0 \leq y_2 \leq k] \cdot P(y_2) + [y_2 < 0]) \star \xi \\
& \leq \sup_{v \in \mathbb{Z}} [z_1 + y_2 \mapsto v] \star ([z_1 + y_2 \mapsto v] \longrightarrow \\
& ([v \neq 0] \cdot [0 \leq y_2 \leq k] \cdot P(y_2 - 1) \cdot (p \cdot [y_2 \in J] \cdot [v \in \{1, \dots, 2\}] + (1 - p) \cdot [y_2 \notin J]) \\
& + [v = 0] \cdot ([0 \leq y_2 \leq k] \cdot P(y_2) + [y_2 < 0])) \star \xi
\end{aligned}$$

25:22 Towards Concurrent Quantitative Separation Logic

resulting from the lookup $x_2 := \langle z_1 + y_2 \rangle$ together with the atom rule. We will consider both cases separately. Let us assume that v is not 0. Then either y_2 is in J and v is either 1 or 2 to make $p \cdot [y_2 \in J] \cdot [v \in \{1, \dots, 2\}]$ not zero, or y_2 is not in J . Then, however, v needs to be -1 , because else ξ_J will evaluate to zero. Both cases can then be used to turn $P(y_2 - 1)$ into $P(y_2)$. In both cases, we can also use $[z_1 + y_2 \mapsto v]$ to re-establish the resource invariant ξ_J . If, on the other side, v is 0, we do not get any new information, but also do not need to update $P(y_2)$, and directly re-establish the resource invariant ξ_J .

Due to

$$\begin{aligned} & [y_2 \geq 0] \cdot ([0 \leq y_2 \leq k] \cdot P(y_2) + [y_2 < 0]) + [y_2 < 0] \cdot 1 \\ = & [0 \leq y_2 \leq k] \cdot P(y_2) + [y_2 < 0] \end{aligned}$$

we establish the loop invariant with respect to postexpectation 1.

For the consumer C_3 we require a loop invariant that checks if l indeed matches the size of the set J . We prove the loop invariant $I_3 = [0 \leq y_3 \leq k] \cdot [y_3 + l = |J \cap \{0, \dots, y_3\}|]$ with respect to the postexpectation $[l = |J|]$ and the resource invariant ξ_J :

$$\begin{aligned} & \llbracket [0 \leq y_3 \leq k] \cdot [l = |J \cap \{y_3 + 1, \dots, k\}|] \mid \xi_J \\ x_3 := \langle z_2 + y_3 \rangle; \\ & \llbracket \left\{ \begin{array}{l} [x_3 = -1] \cdot [1 \leq y_3 \leq k + 1] \cdot [l = |J \cap \{y_3, \dots, k\}|] \\ + [x_3 = -1] \cdot [y_3 < 1] \cdot [l = |J|] \\ + [x_3 \neq 0] \cdot [x_3 \neq -1] \cdot [1 \leq y_3 \leq k + 1] \cdot [l + 1 = |J \cap \{y_3, \dots, k\}|] \\ + [x_3 \neq 0] \cdot [x_3 \neq -1] \cdot [y_3 < 1] \cdot [l + 1 = |J|] \\ + [x_3 = 0] \cdot [0 \leq y_3 \leq k] \cdot [l = |J \cap \{y_3 + 1, \dots, k\}|] \\ + [x_3 = 0] \cdot [y_3 < 0] \cdot [l = |J|] \end{array} \right\} \mid \xi_J \\ \text{if } (x_3 \neq 0) \{ \\ & \llbracket \left\{ \begin{array}{l} [x_3 = -1] \cdot [1 \leq y_3 \leq k + 1] \cdot [l = |J \cap \{y_3, \dots, k\}|] \\ + [x_3 = -1] \cdot [y_3 < 1] \cdot [l = |J|] \\ + [x_3 \neq -1] \cdot [1 \leq y_3 \leq k + 1] \cdot [l + 1 = |J \cap \{y_3, \dots, k\}|] \\ + [x_3 \neq -1] \cdot [y_3 < 1] \cdot [l + 1 = |J|] \end{array} \right\} \mid \xi_J \\ \text{if } (x_3 \neq -1) \{ \\ & \llbracket [1 \leq y_3 \leq k + 1] \cdot [l + 1 = |J \cap \{y_3, \dots, k\}|] + [y_3 < 1] \cdot [l + 1 = |J|] \mid \xi_J \\ & l := l + 1 \\ & \llbracket [1 \leq y_3 \leq k + 1] \cdot [l = |J \cap \{y_3, \dots, k\}|] + [y_3 < 1] \cdot [l = |J|] \mid \xi_J \\ \}; \\ & \llbracket [1 \leq y_3 \leq k + 1] \cdot [l = |J \cap \{y_3, \dots, k\}|] + [y_3 < 1] \cdot [l = |J|] \mid \xi_J \\ & \llbracket [0 \leq y_3 - 1 \leq k] \cdot [l = |J \cap \{y_3, \dots, k\}|] + [y_3 - 1 < 0] \cdot [l = |J|] \mid \xi_J \\ y_3 := y_3 - 1 \\ & \llbracket [0 \leq y_3 \leq k] \cdot [l = |J \cap \{y_3 + 1, \dots, k\}|] + [y_3 < 0] \cdot [l = |J|] \mid \xi_J \\ \} \\ & \llbracket [0 \leq y_3 \leq k] \cdot [l = |J \cap \{y_3 + 1, \dots, k\}|] + [y_3 < 0] \cdot [l = |J|] \mid \xi_J \end{aligned}$$

Here we will take a closer look at the inequality

$$\begin{aligned}
& ([1 \leq y_3 \leq k] \cdot [l = |J \cap \{y_3 + 1, \dots, k\}|] + [y_3 = 0] \cdot [l = |J \cap \{1, \dots, k\}|]) \star \xi_J \\
& \leq \sup_{v \in \mathbb{Z}} [z_2 + y_2 \mapsto v] \star ([z_2 + y_2 \mapsto v] \multimap \star \\
& \quad ([v = -1] \cdot [1 \leq y_3 \leq k + 1] \cdot [l = |J \cap \{y_3, \dots, k\}|] \\
& \quad + [v = -1] \cdot [y_3 < 1] \cdot [l = |J|] \\
& \quad + [v \neq 0] \cdot [v \neq -1] \cdot [1 \leq y_3 \leq k + 1] \cdot [l + 1 = |J \cap \{y_3, \dots, k\}|] \\
& \quad + [v \neq 0] \cdot [v \neq -1] \cdot [y_3 < 1] \cdot [l + 1 = |J|] \\
& \quad + [v = 0] \cdot [0 \leq y_3 \leq k] \cdot [l = |J \cap \{y_3 + 1, \dots, k\}|] \\
& \quad + [v = 0] \cdot [y_3 < 0] \cdot [l = |J|]) \star \xi_J)
\end{aligned}$$

due to the lookup $x_3 := \langle z_2 + y_2 \rangle$ together with the atom rule. We consider all cases separately.

- First, let v be -1
 - If moreover y_3 is between 1 and $k + 1$, then we can directly lower bound the case that y_3 is $k + 1$ by zero as ξ_J does not have carry information for this location. Because v is -1 , we know that y_3 is not in J due to ξ_J . Thus, we also have that $|J \cap \{y_3 + 1, \dots, k\}| = |J \cap \{y_3, \dots, k\}|$.
 - If y_3 is below 1, the same reasoning holds, with the difference that we lower bound the expectation for every value of y_3 below 0 as zero and consider only the case where y_3 is 0.
- In the case that v is neither 0 nor -1 , we first observe that only 1 and 2 are valid values, because ξ_J does not allow any other value for y_3 between 0 and k .
- In the cases where v is either $k + 1$ or below 0, we just lower bound the formula by zero. However, for the latter cases we have $|J \cap \{y_3 + 1, \dots, k\}| + 1 = |J \cap \{y_3, \dots, k\}|$.
- Lastly, in the case that v is 0, the expression already matches the target lower bound, but again, we lower bound the formula by zero if y_3 has a value below 0.

Moreover, we have

$$\begin{aligned}
& [y_3 \geq 0] \cdot ([0 \leq y_3 \leq k] \cdot [l = |J \cap \{y_3 + 1, \dots, k\}|] + [y_3 < 0] \cdot [l = |J|]) \\
& = [0 \leq y_3 \leq k] \cdot [l = |J \cap \{y_3 + 1, \dots, k\}|] + [y_3 < 0] \cdot [l = |J|]
\end{aligned}$$

and thus established a loop invariant with respect to postexpectation $[l = |J|]$.

Now we can combine all three results

$$\begin{aligned}
& \text{// } P(k) \cdot [0 \leq k] \quad | \quad \xi_J \\
& \text{// } [0 \leq k] \cdot P(k) + [k < 0] \cdot [0 = |J|] \quad | \quad \xi_J \\
& l := 0; \\
& \text{// } [0 \leq k] \cdot [l = 0] \cdot P(k) + [k < 0] \cdot [l = |J|] \quad | \quad \xi_J \\
& \left. \begin{aligned}
& \text{// } \left\{ \begin{array}{l} 1 \\ \star ([0 \leq k] \cdot P(k) + [k < 0]) \\ \star ([0 \leq k] \cdot [l = |J \cap \{k + 1, \dots, k\}|] + [k < 0] \cdot [l = |J|]) \end{array} \right. \quad \Bigg| \quad \xi_J
\end{aligned} \right. \\
& y_1, y_2, y_3 := k; \\
& \left. \begin{aligned}
& \text{// } \left\{ \begin{array}{l} 1 \\ \star ([0 \leq y_2 \leq k] \cdot P(y_2) + [y_2 < 0]) \\ \star ([0 \leq y_3 \leq k] \cdot [l = |J \cap \{y_3 + 1, \dots, k\}|] + [y_3 < 0] \cdot [l = |J|]) \end{array} \right. \quad \Bigg| \quad \xi_J
\end{aligned} \right.
\end{aligned}$$

25:24 Towards Concurrent Quantitative Separation Logic

$$C_1 \parallel C_2 \parallel C_3 \\ \parallel 1 \star 1 \star [l = |J|] \mid \xi_J$$

and we have for the whole program C and a set of schedulers $S \subseteq \mathbb{S}$:

$$\begin{aligned} & (P(k) \cdot [0 \leq k]) \leq \text{wrlp}[[C]] ([l = |J|] \mid \xi_J) \\ \text{implies } & (P(k) \cdot [0 \leq k]) \star \xi_J \leq \text{wrlp}[[C]] ([l = |J|] \star \xi_J \mid [\mathbf{emp}]) \quad (\text{share}) \\ \text{implies } & (P(k) \cdot [0 \leq k]) \star \xi_J \leq \text{wlp}^S[[C]] ([l = |J|] \star \xi_J) \quad (\text{wlp-wrlp}) \end{aligned}$$

We can use this to prove the lower bound of probabilities for even more elaborated post-conditions if we have a set of schedulers $S \subseteq \mathbb{S}$ such that C is almost surely terminating with respect to S . One of these is the probability that at least half of the messages are sent successfully, i.e., the probability of the postexpectation $[k + 1 \geq l \geq \frac{k+1}{2}]$ – or equivalently $\sum_{\frac{k+1}{2} \leq j \leq k+1} [l = j]$. For this, we use the resource invariant $\xi_j = \max_{J \subseteq \{0, \dots, k\}, |J|=j} \xi_J$ where ξ_J is defined as previous. Although we call ξ_j a resource invariant, we never prove that it is a resource invariant. We only prove that ξ_J is a resource invariant. We can now compute:

$$\begin{aligned} & \text{wlp}^S[[C]] ([l = |J|] \star \xi_J) \geq (P(k) \cdot [0 \leq k]) \star \xi_J \\ \text{implies } & \text{wlp}^S[[C]] \left(\max_{J \subseteq \{0, \dots, k\}, |J|=j} [l = |J|] \star \xi_J \right) \geq \max_{J \subseteq \{0, \dots, k\}, |J|=j} (P(k) \cdot [0 \leq k]) \star \xi_J \quad (\text{max}) \\ \text{implies } & \text{wlp}^S[[C]] ([l = j] \star \xi_j) \geq (p^j \cdot (1-p)^{k-j+1} \cdot [0 \leq k]) \star \xi_j \quad (\text{Definition of } \xi_j) \\ \text{implies } & \text{wlp}^S[[C]] \left(\sum_{\frac{k+1}{2} \leq j \leq k+1} [l = j] \star \xi_j \right) \geq \sum_{\frac{k+1}{2} \leq j \leq k+1} (p^j \cdot (1-p)^{k-j+1} \cdot [0 \leq k]) \star \xi_j \quad (\text{Superlinearity}) \\ \text{implies } & \text{wlp}^S[[C]] \left(\left[k + 1 \geq l \geq \frac{k+1}{2} \right] \star \xi_j \right) \\ & \geq \left(\sum_{\frac{k+1}{2} \leq j \leq k+1} p^j \cdot (1-p)^{k-j+1} \cdot [0 \leq k] \right) \star \xi_j \\ & \quad (\xi_j \text{ is precise and } [k + 1 \geq l \geq \frac{k+1}{2}] \text{ as above}) \end{aligned}$$

We could drop the resource invariant ξ_j inside wlp^S due to monotonicity of wlp^S , for which we do not provide a proof. However, this shows that we can use superlinearity to partition a big problem in smaller problems and afterwards reason about these smaller problems with the help of easier resource invariants, as it is standard in probability theory.

Completeness Theorems for Kleene Algebra with Top

Damien Pous

Plume, LIP, CNRS, ENS de Lyon, France

Jana Wagemaker

University of Nijmegen, The Netherlands

Abstract

We prove two completeness results for Kleene algebra with a top element, with respect to languages and binary relations. While the equational theories of those two classes of models coincide over the signature of Kleene algebra, this is no longer the case when we consider an additional constant “top” for the full element. Indeed, the full relation satisfies more laws than the full language, and we show that those additional laws can all be derived from a single additional axiom. We recover that the two equational theories coincide if we slightly generalise the notion of relational model, allowing sub-algebras of relations where top is a greatest element but not necessarily the full relation.

We use models of closed languages and reductions in order to prove our completeness results, which are relative to any axiomatisation of the algebra of regular events.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Theory of computation → Logic and verification; Theory of computation → Regular languages

Keywords and phrases Kleene algebra, Hypotheses, Completeness, Closed languages

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.26

Funding This work was supported 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 would like to thank Paul Brunet, Amina Doumane, and Jurriaan Rot for the discussions that eventually led to this work, and the CONCUR reviewers for all their comments.

1 Introduction

The axiomatic treatment of regular expressions and languages was developed extensively by Conway [9], after earlier work of Kleene [16]. He raised there a difficult question: how to axiomatise the equations between regular expressions that hold under their standard interpretation as formal languages? Redko had proved that every purely equational axiomatisation must be infinite [32]. Conway proposed such an infinite axiomatisation, which Krob proved to be complete twenty years later [24]. Conway had also proposed finite quasi-equational axiomatisations, one of which Kozen proved to be complete the same year [21] – this axiomatisation is now commonly called *Kleene algebra*. By an additional remark of Boffa [5], this latter completeness result can also be obtained as a consequence of Krob’s completeness result. In the end, all finite quasi-equational axiomatisations proposed by Conway, as well as a few other ones, are actually complete [24, 6].

In symbols, writing $[e]$ for the language of a regular expression e and $\text{KA} \vdash e = f$ when the equation $e = f$ is derivable in any of the aforementioned axiomatisations, we have that for all regular expressions e, f ,

$$\text{KA} \vdash e = f \iff [e] = [f]$$



© Damien Pous and Jana Wagemaker;

licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Sławomir Lasota, and Anca Muscholl; Article No. 26; pp. 26:1–26:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

26:2 Completeness Theorems for Kleene Algebra with Top

The above equivalence extends with two more clauses. When an equation is derivable, it must hold in all models of the chosen axiomatisation. These include in particular language models (LANG) and relational models (REL), for which we actually have an equivalence: writing $\mathcal{X} \models e = f$ when the equation $e = f$ holds in all members of a class of models \mathcal{X} , we actually have:

$$\text{KA} \vdash e = f \iff \text{REL} \models e = f \iff \text{LANG} \models e = f \iff [e] = [f]$$

Completeness w.r.t. LANG is immediate given the previous equivalence: the language interpretation of a regular expression lies in LANG. This is less obvious for REL: completeness comes from a nice trick due to Pratt showing that every member of LANG embeds into a member of REL [31, third page].

As an immediate consequence of the above equivalence, the equational theory of REL (or LANG) is decidable – more precisely, PSPACE-complete. This has important applications in program verification: Kleene algebras and their extension to Kleene algebras with tests [17] make it possible to represent and reason about the big-step semantics of while programs, algebraically. This was used for instance to analyse compiler optimisations [19]. The decidability result was also implemented in proof assistants such as Coq and Isabelle/HOL, in order to automate some reasoning steps about binary relations and Hoare logic on while programs [28, 23].

The above-mentioned results apply to the *regular* operations and constants: composition, union, Kleene star, identity, emptiness. A natural question is whether they extend to other operations or constants, such as intersection, converse, fullness. The case of converse was dealt with by Ésik et al.: the equational theories of REL and LANG differ in the presence of converse but both can be axiomatised [4, 13], and they remain PSPACE-complete [8]. The case of intersection (with or without converse or the various constants) is significantly more difficult, and remains partly open, see [2, 7, 26, 12]. In this paper we focus on the addition of a constant \top , interpreted as the full language in LANG and as the full relation in REL.

The usefulness of adding such a constant was demonstrated recently in the context of Kleene algebras with tests (KAT), to model *incorrectness logic* [33]. Indeed, while KAT alone makes it possible to model Hoare triples for partial correctness [18], the addition of a full element makes it possible to compare the (co)domains of relations, and thus to encode *incorrectness triples* [27, Section 5.3]. KAT with a top element was also used earlier, as an intermediate structure to characterise a semantics for abnormal termination [25, Definition 12].

As expected, one should consider an axiom expressing that \top is a greatest element:

$$x \leq \top \tag{T}$$

(Where $x \leq y$ is a shorthand for $x + y = y$.) Together with the Kleene algebra axioms, axiom (T) yields a complete axiomatisation w.r.t. language models: we sketched a proof in [30, Example 3.4], which we make fully explicit here in Section 3 (Theorem 3.5). This proof gives us as a byproduct that the equational theory of Kleene algebras with a greatest element remains PSPACE-complete.

Unfortunately, the previous axiom is not enough to deal with relational models. In fact, in the presence of \top , the equational theories of LANG and REL differ. Indeed, there are laws such as $\top x \top y \top = \top y \top x \top$ [29, page 13], or $\top x \top x = \top x$ [33, page 14], which are valid in REL, but not in LANG.

In the present paper, we show that it suffices to further add the following axiom in order to obtain a complete axiomatisation for REL (Theorem 4.16):

$$x \leq x \cdot \top \cdot x \tag{F}$$

This inequation is mentioned in [33, page 14]; it holds in relational models, but not in language ones. Thanks to (T), axiom (F) may be seen as a consequence of Ésik et al.'s axiom $x \leq x \cdot x^\circ \cdot x$ for dealing with converse (\cdot°) in relational models [13, 4]. How to use axiom (F) in an equational proof is not so intuitive: it does not give rise to a natural notion of normal form, and it must often be used in conjunction with (T) in order to compensate the fact that it duplicates subterms. For instance here is how we can prove the first of the aforementioned laws:

$$\begin{aligned} \top x \top y \top &\leq \top x \top y \top \top \top x \top y \top && \text{(by (F))} \\ &\leq \top x \top y \top \top \top x \top && \text{(by (T))} \\ &\leq \top x \top y \top x \top && \text{(by (T))} \\ &\leq \top y \top x \top && \text{(by (T))} \end{aligned}$$

(We wrote compositions by juxtaposition, skipped the associativity steps, and underlined the subterms to be simplified by axiom (T) – the converse inequation is derived symmetrically.) Our completeness proof actually goes via a factorisation property (Proposition 4.7) intuitively asserting that one can always proceed in this way to reason about star-free expressions: expand the expressions using (F) a number of times, then remove spurious subterms using (T). Combining such a technique together with Kleene algebra reasoning for star is the second challenge we address in the present work. To get a grasp on the difficulties, the reader may try to find a proof of the following valid law of REL, using KA and axiom (F):

$$a^+ \leq o(a + \top o) \top a^+ \tag{*}$$

There, o and a^+ are shorthands for $(aa)^*a$ and a^*a ; we give a solution in Examples 4.13-4.15.

Finally, we show that the difference between the equational theories of language and relational models can be blurred if we slightly generalise the notion of relational model, allowing \top to be any greatest relation rather than the full one¹ (Corollary 5.2).

We prove our two main theorems using the concept of *closed language model* for Kleene algebra with hypotheses [11], and the reduction technique made explicit in [30, 15]². Intuitively, we establish reductions from KA with (T) and KA with (T, F) to plain KA, so that we can deduce completeness and decidability of the former theories from completeness and decidability of the latter one.

While the first reduction is relatively straightforward – this is a syntactical linear reduction, the second one is not. We exploit the aforementioned factorisation result (Proposition 4.7) and Kleene's theorem in order to show that regular languages are preserved by a certain closure operation, and that this preservation property can be justified algebraically (Proposition 4.14). Moreover, in order to establish the correspondence between the closed languages used there and relational models, we resort to a graph theoretical characterisation of the equational theory of REL [7, Theorem 6] (whose main ingredient dates back to the works of Freyd and Scedrov [14, page 208] and Andréka and Bredikhin [3, Theorem 1]).

¹ considering subalgebras where only certain relations are kept, since otherwise the only greatest relation is the full one.

² Such a technique is somehow implicit in Kozen and Smith's completeness proof for KAT [20] and Ésik et al.'s completeness proof for Kleene algebra with converse [4, 13].

Related work

Zhang et al. give a completeness result for KAT together with axiom (T), in terms of guarded string languages [33, Theorem 9]. They observe that this axiomatisation is incomplete for REL, that it does not suffice to properly express *incorrectness triples*, and they leave the existence of a complete axiomatisation for relational models open. Our Theorem 4.16 gives a positive answer to this question, in the more primitive setting of plain Kleene algebra, without tests. We believe that a similar answer holds also for KAT; if this is the case, then we would obtain a system where we can reason purely equationally about incorrectness triples, as envisioned by Zhang et al.

For the weaker theory of KAT with (T), the main completeness results of Zhang et al. [33, Theorems 7 and 9] are wrong: the model of guarded strings they designed equates too many expressions (namely, Σ^* and \top – see Remark 3.6). Our Theorem 3.5 uses a different language model, and we believe our simple (and linear) reduction from KA with (T) to KA is also a reduction from KAT with (T) to KAT, so that, e.g., [33, Theorem 10] about the complexity of KAT with (T) remains true.

Zhang et al. also give a completeness result w.r.t. generalised relational models [33, Theorem 8]. Their proof is problematic because it relies on their Theorem 7, but the key idea remains valid: adapting Pratt’s trick to embed language models into relational ones. We use the very same technique to obtain Corollary 5.2.

Outline

We setup and recall basic notation for regular expressions, formal languages and universal algebra in Section 2. Then we deal with language models in Section 3, and relational models in Section 4. While the language case was already sketched in [30, Example 3.4], we find it useful to treat it explicitly here, before dealing with the more involved case of relations: it illustrates the reduction method in a simpler setting, and we build on the reduction for languages to establish the reduction for relations. We finally prove completeness with respect to generalised relational models in Section 5.

2 Preliminaries

Given a set X , we write X^* for the set of *words* over X : finite sequences of elements of X . We let u, v range over words, we write ϵ for the empty word, and uv for the concatenation of two words u, v . A *language* is a set of words. We let e, f range over *regular expressions over X* , generated by the following grammar:

$$e, f ::= e + f \mid e \cdot f \mid e^* \mid 0 \mid 1 \mid x \in X$$

We sometimes omit the dots in regular expressions, writing, e.g., ab^* for $a \cdot b^*$. As usual, we associate a language $[e]$ to every regular expression e , the *language of e* . A language is *regular* if it is the language of a regular expression.

We fix a finite set Σ of *letters*, ranged over using a, b . We write Σ_\top for the set Σ extended with a new element \top . We call the regular expressions over Σ_\top *regular expressions with top* (or often just *expressions*, since we are mostly concerned with these). We shall sometimes see words over Σ_\top as regular expressions with top. E.g., the word $a\top$ can be seen as the expression $a \cdot \top$.

We consider signatures $S \triangleq \{+_2, \cdot_2, \cdot^*_1, 0_0, 1_0\}$ and $S_\top \triangleq S \cup \{\top_0\}$. Given an S -algebra A and a valuation $\sigma : \Sigma \rightarrow A$, we write $\hat{\sigma}$ for the unique homomorphism extending σ to regular expressions over Σ . Similarly, given an S_\top -algebra A and a valuation $\sigma : \Sigma \rightarrow A$, we

write $\hat{\sigma}$ for the unique homomorphism extending σ to regular expressions with top. (Note in that case that the domain of the valuation is only Σ , and that $\hat{\sigma}(\top) = \top_A$ by definition: \top is a constant, not a variable.)

Given a class \mathcal{X} of S -algebras and two regular expressions e, f over Σ , we write $\mathcal{X} \models e = f$ if for all members A of \mathcal{X} and all valuations $\sigma : \Sigma \rightarrow A$, we have $\hat{\sigma}(e) = \hat{\sigma}(f)$. We use similar notations for classes of S_{\top} -algebras and regular expressions with top.

An *equation* is a pair of regular expressions e, f , written $e = f$. We write $e \leq f$, an *inequation*, as a shorthand for the equation $e + f = f$. An *axiomatisation* is a set of equations (or implications between equations). Given such a set \mathcal{E} , we write $\mathcal{E} \vdash e = f$ when the equation $e = f$ is derivable from \mathcal{E} using the rules of equational reasoning (where letters from Σ appearing in the equations of \mathcal{E} can be substituted by arbitrary terms).

We let KA stand for any axiomatisation over plain regular expressions which is sound and complete w.r.t. the regular language interpretation, i.e., such that for all regular expressions e, f (without top), we have³

$$\text{KA} \vdash e = f \iff [e] = [f] \quad (\dagger)$$

As explained in the introduction, valid candidates for KA include Conway's infinite but purely equational axiomatisation [9, page 116] (proved complete by Krob [24]), Kozen's Kleene algebras [21], left-handed Kleene algebras [22, 10], and Boffa's algebras [6].

Also note that the above requirement is equivalent to the following one, since $L \subseteq K$ iff $L \cup K = K$ for all languages L, K :

$$\text{KA} \vdash e \leq f \iff [e] \subseteq [f] \quad (\ddagger)$$

3 Languages

We let L, K range over languages on some alphabet X , and $\mathcal{P}(X^*)$ denotes the set of all such languages. Languages on X form a S_{\top} -algebra with the operations defined as follows:

$$\begin{aligned} L + K &\triangleq L \cup K \\ L \cdot K &\triangleq \{uv \mid u \in L \wedge v \in K\} \\ L^* &\triangleq \{u_0 \dots u_{n-1} \mid \exists n \in \mathbb{N}, \forall i < n, u_i \in L\} \\ 0 &\triangleq \emptyset \\ 1 &\triangleq \{\epsilon\} \\ \top &\triangleq X^* \end{aligned}$$

(That is, $+$ is set-theoretic union, \cdot is language concatenation, $*$ is Kleene star, 0 and \top are the empty and full languages, respectively, and 1 is the singleton language that contains the empty word.) We write LANG for the class of all S_{\top} -algebras of the above shape.

Let KA_{\top} , *Kleene Algebra with a Top element*, denote the union of the axioms from KA and axiom (T). We prove in this section that KA_{\top} is sound and complete for LANG.

Following the strategy from [11, 30], the first step consists of defining the closure operation below, according to the axiom (T) we add to Kleene algebra:

³ Actually, we require slightly more if the axiomatisation contains implications: those implications should be valid in the models of languages and binary relations.

26:6 Completeness Theorems for Kleene Algebra with Top

► **Definition 3.1** (Language closure C_T). *Given two words u, v over Σ_\top , we write $u \leftarrow_T v$ if u is obtained from v by replacing an occurrence of \top with an arbitrary word $w \in \Sigma_\top^*$. Given a language L over Σ_\top , we call T -closure of L the following language*

$$C_T(L) \triangleq \{u \mid u \leftarrow_T^* v \text{ for some } v \in L\}$$

C_T is indeed a closure operator, and $C_T(L)$ may alternatively be described as the set of words obtained by replacing occurrences of \top in a word of L with arbitrary words over Σ_\top .

► **Lemma 3.2.** *C_T is an S_\top -algebra homomorphism.*

Proof. By a routine verification; the case for composition follows from the fact that we replace single letters. ◀

► **Definition 3.3** (Expression closure r). *Let r be the unique S -algebra homomorphism on expressions with top such that $r(a) = a$ for all letters $a \in \Sigma$, and $r(\top) = \Sigma_\top^*$ (where Σ_\top^* is a regular expression with top for the full language – e.g., $(a + b + \dots + \top)^*$).*

► **Proposition 3.4.** *For all expressions e , we have*

- (i) $[r(e)] = C_T[e]$, and
- (ii) $\text{KA}_T \vdash e = r(e)$.

Proof.

- (i) $[r(\cdot)]$ and $C_T[\cdot]$ are S -algebra homomorphisms, and they agree on Σ_\top .
- (ii) We proceed by induction on e ; the only interesting case is when $e = \top$, for which we have $\text{KA}_T \vdash r(\top) \leq \top$ by axiom (T), and $\text{KA}_T \vdash \top \leq r(\top)$ by completeness of KA (‡), since $[\top] = \{\top\} \subseteq \Sigma_\top^* = [r(\top)]$. ◀

► **Theorem 3.5.** *For all regular expressions with top e, f , we have*

$$\text{LANG} \models e = f \iff C_T[e] = C_T[f] \iff \text{KA}_T \vdash e = f$$

Proof. We have

$$\begin{aligned} & \text{LANG} \models e = f \\ \Rightarrow & C_T[e] = C_T[f] && (C_T[\cdot] \text{ is an interpretation into a member of LANG, by Lemma 3.2}) \\ \Leftrightarrow & [r(e)] = [r(f)] && (\text{Proposition 3.4(i)}) \\ \Leftrightarrow & \text{KA} \vdash r(e) = r(f) && (\text{completeness of KA (‡)}) \\ \Rightarrow & \text{KA}_T \vdash e = f && (\text{transitivity and Proposition 3.4(ii)}) \\ \Rightarrow & \text{LANG} \models e = f && (\text{soundness of KA}_T \text{ axioms w.r.t. LANG}) \end{aligned}$$

(In the last step, soundness w.r.t. LANG comes from our assumption about KA, and a trivial verification for axiom (T).) ◀

Note that the first equivalence in the above theorem can be obtained in a more direct way, without resorting to completeness of some axiomatisation; moreover the right-to-left implication of the second equivalence is an instance of a general property of closed language models [11, Theorem 2]. The reduction r is used only for the left-to-right implication of this second equivalence.

According to the above proof, we could complete the statement with “... $\iff [r(e)] = [r(f)]$ ”. Doing so gives us a PSPACE algorithm: compute the regular expressions $r(e)$ and $r(f)$, and compare them for language equivalence.

► **Remark 3.6.** Note that it is crucial that $r(\top)$ be defined as a regular expression Σ_{\top}^* for the full language on Σ_{\top} rather than an expression Σ^* for the full language on just Σ : otherwise we would equate Σ^* and \top , while those are different in LANG (e.g., for a counterexample when $\Sigma = \{a, b\}$, interpret both a and b as the empty language on some non-empty alphabet).

4 Relations

Given a set X , a *relation on X* is a set of pairs of elements from X . We let R, S range over such relations, whose set is written $\mathcal{P}(X \times X)$, and we write $x R y$ for $\langle x, y \rangle \in R$. Relations on X form an S_{\top} -algebra with the operations defined as follows:

$$R + S \triangleq R \cup S$$

$$R \cdot S \triangleq \{\langle x, z \rangle \mid \exists y \in X, x R y \wedge y S z\}$$

$$R^* \triangleq \{\langle x_0, x_n \rangle \mid \exists n \in \mathbb{N}, x_1, \dots, x_{n-1}, \forall i < n, x_i R x_{i+1}\}$$

$$0 \triangleq \emptyset$$

$$1 \triangleq \{\langle x, x \rangle \mid x \in X\}$$

$$\top \triangleq X \times X$$

($+$ is set-theoretic union, \cdot is relational composition, \cdot^* is reflexive transitive closure, $0, 1$ and \top are the empty, identity and full relations, respectively.) We write REL for the class of all S_{\top} -algebras of the above shape.

Let KA_F , *Kleene Algebra with a Full element*, denote the union of the axioms from KA_T and axiom (F). Let us emphasise that despite the abbreviation, KA_F extends KA_T and thus contains axiom (T). We prove in this section that KA_F is sound and complete for REL. The proof consists of two parts. First we characterise the equational theory of REL in terms of closed languages (Section 4.1, Proposition 4.8), then we use reductions to show completeness of KA_F w.r.t. this closed language interpretation and obtain our main result (Section 4.2, Theorem 4.16).

4.1 Characterisation via closed languages

We start by extending the previous closure function (Definition 3.1), in order to take into account the new axiom (F):

► **Definition 4.1** (Language closure C_F). *Given two words u, v over Σ_{\top} , we write $u \leftarrow_F v$ if either $u \leftarrow_T v$, or u is obtained by replacing a subword of the shape $w\top w$ in v , with w (for some word $w \in \Sigma_{\top}^*$). Given a language L over Σ_{\top} , we call F -closure of L the language*

$$C_F(L) \triangleq \{u \mid u \leftarrow_F^* v \text{ for some } v \in L\}$$

C_F is a closure operator, but unlike C_T in the previous section, C_F is not a homomorphism – e.g., $C_F(\{a\}) \cdot C_F(\{\top a\})$ contains the word a while $C_F(\{a\}) \cdot C_F(\{\top a\})$ does not. Moreover, an elementary description of C_F requires more work than for C_T in the previous section.

Let E be the following function on languages over Σ_{\top} , where for a word u and a natural number n , we write u^n for the word obtained by concatenating n copies of u :

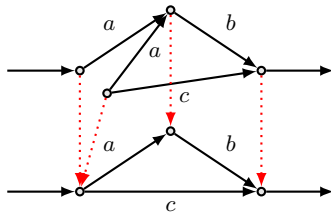
$$E(L) \triangleq \{w \mid \exists n, w(\top w)^n \in L\}$$

We shall prove that $C_F = E \circ C_T$, and that C_F can be characterised in terms of certain graph homomorphisms (Proposition 4.7 below). Before doing so, we need to define the type of graphs we will use in what follows.

► **Definition 4.2** (Graph, graph homomorphism). A graph is a tuple $\langle V, E, \iota, o \rangle$, where V is a set of vertices, $E \subseteq V \times \Sigma \times V$ is a set of labelled edges, and $\iota, o \in V$ are two distinguished vertices, respectively called input and output.

A graph homomorphism from the graph G to the graph H is a function from vertices of G to vertices of H that preserves labelled edges, input, and output. We write $H \triangleleft G$ when there exists a homomorphism from G to H .

The relation \triangleleft on graphs is a preorder. We depict graphs as usual, using an unlabelled ingoing (resp. outgoing) arrow to indicate the input (resp. output); we use dotted red arrows to depict graph homomorphisms. For instance, we depict two finite connected graphs below, and a homomorphism between them:



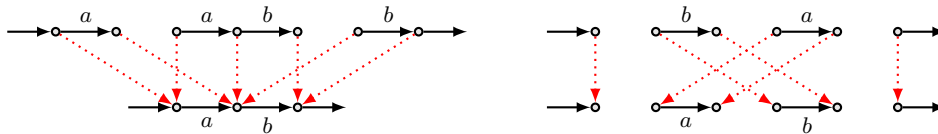
► **Definition 4.3** (Graph of a word). We associate to each word $u \in \Sigma_{\top}^*$ the graph $g(u)$ defined as follows:

- the vertices are the natural numbers smaller or equal to the length n of u ;
- for $a \in \Sigma$ there is an a -labelled edge from i to $i + 1$ if the i -th letter of u is a ;
- the input is 0 and the output is n .

Graphs of words are rather simple: graphs as depicted above do not arise as graphs of words. For words not containing \top , they are just directed paths from the input to the output. For words containing \top , they are collections of (possibly empty) directed paths where the input is the starting-point of some path and the output is the end-point of some path. For example, the graphs of abc and $d\top de\top$ are depicted below:



Nevertheless, homomorphisms between graphs of words may be non-trivial. For instance, we have $g(ab) \triangleleft g(a\top ab\top b)$ and $g(\top a\top b\top) \triangleleft g(\top b\top a\top)$, as witnessed below:



In the sequel, we shall represent homomorphisms between graphs of words in a slightly more compact way, starting directly from the natural writing of the words, and using horizontal lines and shaded parallelograms to emphasise distinguished subwords and mappings between them. For instance, the above homomorphisms can be generalised to $g(uv) \triangleleft g(u\top uv\top v)$ and $g(\top u\top v\top) \triangleleft g(\top v\top u\top)$ for arbitrary words u, v , which we can represent as follows:



Our main interest in graphs and homomorphisms comes from the following characterisation of the equational theory of REL. This characterisation appeared first in [7, Theorem 6], for the syntax of Kleene allegories. Its (trivial) extension to Kleene allegories with top then appeared in [29, Theorem 16].

► **Theorem 4.4** ([7, Theorem 6]). *For all regular expressions with top e, f , we have:*

$$\text{REL} \models e \leq f \iff \forall u \in [e], \exists v \in [f], g(u) \triangleleft g(v)$$

Proof. Cf. the above references. That we need the theorem only in a small fragment here (without intersection and converse) does not seem to enable substantial simplifications. In particular, we still need to consider arbitrary graphs, and a variant of [3, Lemma 3] with top. We give a proof in Appendix A for the sake of completeness. ◀

► **Remark 4.5.** For words u, v without top, we have $g(u) \triangleleft g(v)$ iff $u = v$. Therefore, for regular expressions e, f without top (whose languages only contain words without top), the above theorem reduces to $\text{REL} \models e \leq f \iff [e] \subseteq [f]$, a standard variant of one of the equivalences recalled in the introduction.

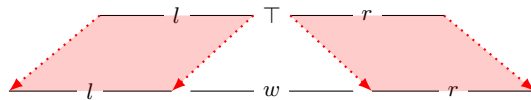
Thanks to Theorem 4.4, it suffices to relate homomorphisms between graphs of words to the notion of C_F -closure. We do so in the following lemma.

► **Lemma 4.6.** *For all words $u, v \in \Sigma_{\top}^*$, the following are equivalent:*

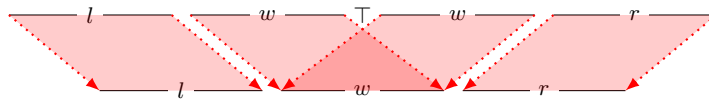
- (i) $u \rightsquigarrow_F^* v$,
- (ii) $g(u) \triangleleft g(v)$,
- (iii) $u \in E(C_T \{v\})$.

Proof. We show (i) \Rightarrow (ii) \Rightarrow (iii) \Rightarrow (i). For the first implication, since \triangleleft is a preorder, it suffices to show that $u \rightsquigarrow_F v$ entails $g(u) \triangleleft g(v)$. There are two cases to consider.

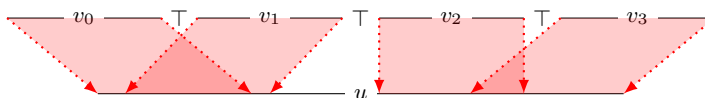
- either the rewriting rule associated to axiom (T) was used, i.e., $u = lwr$ and $v = l\top r$ for some words $l, w, r \in \Sigma_{\top}^*$. In that case we have the following homomorphism from the graph of v to the graph of u :



- or the rewriting rule associated to axiom (F) was used, i.e., $u = lwr$ and $v = lw\top wr$ for some words $l, w, r \in \Sigma_{\top}^*$. In that case we have the following homomorphism from the graph of v to the graph of u :

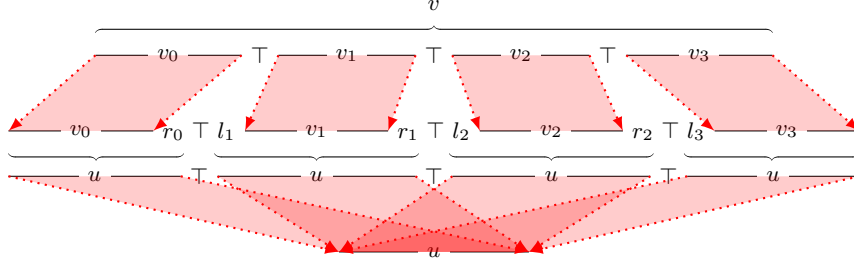


For the second implication, assume $g(u) \triangleleft g(v)$. Let n be the number of occurrences of \top in v , and let v_0, \dots, v_n be top-free words such that $v = v_0\top v_1\top \dots \top v_n$. Since they are top-free, those subwords must be mapped linearly to u , and thus be subwords of u . For instance, when $n = 3$, the homomorphism may look as follows:



26:10 Completeness Theorems for Kleene Algebra with Top

For all $0 \leq i \leq n$, let l_i, r_i be the words such that $u = l_i v_i r_i$. We have that l_0 and r_n must be the empty word since inputs and outputs must be preserved by homomorphisms. We have $u(\top u)^n \leftarrow_T^n v$: we can obtain $u(\top u)^n$ from v by replacing the i th occurrence of \top in v with the word $r_{i-1} \top l_i$, for $0 < i \leq n$. This suffices to conclude that $u \in E(C_T \{v\})$: we have proven (ii) \Rightarrow (iii). As an example, when $n = 3$, the situation may be depicted as follows:



For the last implication, assume that $u \in E(C_T \{v\})$. There exists n such that $u(\top u)^n \leftarrow_T^* v$, and thus in particular $u(\top u)^n \leftarrow_F^* v$. Finally observe that $u \leftarrow_F^* u(\top u)^n$ using n rewriting steps using (F), so that we can conclude by transitivity: $u \leftarrow_F^* u(\top u)^n \leftarrow_F^* v$. \blacktriangleleft

The above lemma has two important immediate consequences. First we have the announced factorisation of the closure C_F , and second, combined with Theorem 4.4, we obtain a characterisation of the equational theory of REL in terms of closed languages:

► **Proposition 4.7.** *We have $C_F = E \circ C_T$.*

► **Proposition 4.8.** *For all regular expressions with top e, f , we have:*

$$\text{REL} \models e = f \iff C_F[e] = C_F[f]$$

Proof. For all e, f , we have:

$$\text{REL} \models e \leq f \iff \forall u \in [e], \exists v \in [f], g(u) \triangleleft g(v) \quad (\text{by Theorem 4.4})$$

$$\iff [e] \subseteq C_F[f] \quad (\text{by Lemma 4.6})$$

The initial statement follows by antisymmetry and the fact that C_F is a closure (so that for all languages L, K , $L \subseteq C_F(K)$ iff $C_F(L) \subseteq C_F(K)$). \blacktriangleleft

4.2 Completeness w.r.t. closed languages

It remains to show that KA_F is complete w.r.t. the previous closed language interpretation. We use reductions in order to do so: we find a counterpart to the function r from Section 3 (Definition 3.3), for the F -closure rather than the T -closure. By Proposition 4.7, and since we already have the function r for T -closure, it actually suffices to find a function s that corresponds to the function E , i.e., such that for all expressions e , $s(e)$ is an expression whose language is $E[e]$.

To this end, we use Kleene's theorem stating that a language is regular if and only if it is recognisable by a finite automaton, and the fact that regular languages are closed under union and intersections. Using those tools, we show that the language $E(L)$ is regular whenever L is a regular language, by forming unions and intersections of regular languages extracted from some finite automaton for L .

We first recall standard notions from finite automata theory.

► **Definition 4.9** (Non-deterministic finite automaton). *Let X be a finite set. A non-deterministic finite automaton (NFA) over the alphabet X is a tuple $\mathcal{A} = \langle Q, i, \Delta, F \rangle$ where:*

- Q is a finite set of states;
- $i \in Q$ is an initial state;
- $\Delta : X \rightarrow \mathcal{P}(Q \times Q)$ is the transition relation, associating to each letter of X a relation on states;
- $F \subseteq Q$ is a subset of accepting states.

We extend the transition relation Δ into a function Δ' on words as follows (where as before, 1 is the identity relation on Q and \cdot is relation composition):

$$\begin{cases} \Delta'(\epsilon) \triangleq 1 \\ \Delta'(xu) \triangleq \Delta(x) \cdot \Delta'(u) \quad \text{for } x \in X \text{ and } u \in X^* \end{cases}$$

The language of \mathcal{A} from states p to q , written $\mathcal{L}_{\mathcal{A}}(p, q)$ or just $\mathcal{L}(p, q)$ when \mathcal{A} is clear from the context, is defined as follows:

$$\mathcal{L}_{\mathcal{A}}(p, q) \triangleq \{u \in X^* \mid \langle p, q \rangle \in \Delta'(u)\}$$

The language of \mathcal{A} , written $\mathcal{L}_{\mathcal{A}}$ is finally obtained as $\bigcup_{f \in F} \mathcal{L}_{\mathcal{A}}(i, f)$.

Intuitively, the language from p to q consists of those words that label a path from p to q in the automaton, and the language of the automaton consists of those words labelling a path from the initial state to some accepting state.

We will also need a function which is intuitive in the end, but cumbersome to define. Let us use the standard notations for lists: $[]$ for the empty list, $x :: q$ for the insertion of an element x in front of a list q , and $[x; y; \dots; z]$ for concrete lists. Given a set Q , two elements $p, q \in Q$, and a list $l \in (Q \times Q)^*$ of pairs elements of Q , we write $\text{pr}(p, l, q)$ for the list of pairs of elements of Q defined as follows, by recursion on l :

$$\begin{cases} \text{pr}(p, [], q) \triangleq \{\langle p, q \rangle\} \\ \text{pr}(p, \langle r, s \rangle :: k, q) \triangleq \langle p, r \rangle :: \text{pr}(s, k, q) \quad \text{for } r, s \in Q, \text{ and } k \in (Q \times Q)^* \end{cases}$$

Intuitively, $\text{pr}(p, l, q)$ shifts the pairs found in l , integrating p at the beginning and q at the end. For instance, we have $\text{pr}(p, [\langle q, r \rangle; \langle s, t \rangle], u) = [\langle p, q \rangle; \langle r, s \rangle; \langle t, u \rangle]$.

► **Example 4.10.** The function pr is useful for the following reason. Consider an automaton with initial state i , a single final state f , $\Delta(a) = \{\langle r, s \rangle\}$, and $\Delta(b) = \{\langle t, u \rangle\}$. Suppose we want to characterise the set of words w such that $wawbw$ is accepted. Those are precisely those words in the intersection $\mathcal{L}(i, r) \cap \mathcal{L}(s, t) \cap \mathcal{L}(u, f)$. The terms from this intersection are easily described using pr : we have $\text{pr}(i, [\langle r, s \rangle; \langle t, u \rangle], f) = [\langle i, r \rangle; \langle s, t \rangle; \langle u, f \rangle]$. ◀

We finally write X^{\otimes} for the set of duplicate-free finite sequences over X (i.e., such that every element of X appears at most once). When X is finite, so is X^{\otimes} .

We now have all that we need to characterise the image of E on regular languages:

► **Proposition 4.11.** *Let $\mathcal{A} = \langle Q, i, \Delta, F \rangle$ be a NFA over Σ_{\top} with language \mathcal{L} . We have*

$$E(\mathcal{L}) = \bigcup \left\{ \bigcap \{ \mathcal{L}(p, q) \mid \langle p, q \rangle \in \text{pr}(i, l, f) \} \mid l \in \Delta(\top)^{\otimes}, f \in F \right\}$$

26:12 Completeness Theorems for Kleene Algebra with Top

Proof. We prove the two inclusions separately.

To prove the inclusion from left to right, assume $w \in E(\mathcal{L})$. Let m be the length of w , and let n be the least natural number such that $w(\top w)^n \in \mathcal{L}$. By definition, there is some $f \in F$ and a path from i to f labelled with $w(\top w)^n$ in \mathcal{A} . Call a \top -transition a pair $\langle p, q \rangle$ belonging to $\Delta(\top)$. Let l be the sequence of \top -transitions used in this path at positions $m+1, 2m+1, \dots, nm+1$. This sequence is duplicate-free by minimality of n : if the same \top -transition was appearing twice, we would find a smaller witness for the membership of w in $E(\mathcal{L})$. We check easily that for all pairs $\langle p, q \rangle \in \text{pr}(i, l, f)$, we have $w \in \mathcal{L}(p, q)$. Therefore, w belongs to the right-hand side expression.

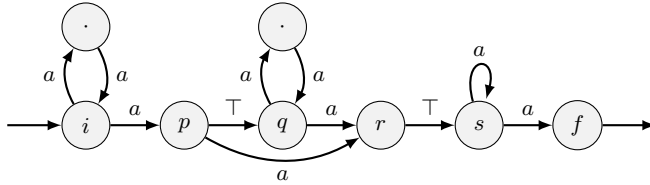
To prove the inclusion from right to left, let w, l, f such that for all $\langle p, q \rangle \in \text{pr}(i, l, f)$, we have $w \in \mathcal{L}(p, q)$. Let n be the length of l . We can construct a path from i to f labelled by $w(\top w)^n$ in \mathcal{A} . Therefore we have $w(\top w)^n \in \mathcal{L}$, whence $w \in E(\mathcal{L})$. \blacktriangleleft

The above formula expresses $E(\mathcal{L})$ as a finite union of finite intersections of languages of the form $\mathcal{L}(p, q)$, which are all regular by Kleene's theorem. Since regular languages are closed under unions and intersections, we deduce that $E(\mathcal{L})$ is regular. In other words, the function E preserves regularity of languages over Σ_{\top} .

► **Definition 4.12** (Expression closure s). Given a regular expression with top e , we define the regular expression with top $s(e)$ as follows:

1. construct a NFA $\langle Q, i, \Delta, F \rangle$ whose language is $[e]$;
2. for all $l \in \Delta(\top)^{\otimes}$ and all $f \in F$, compute a regular expression with top $g_{l,f}$ for the regular language $\bigcap \{ \mathcal{L}(p, q) \mid \langle p, q \rangle \in \text{pr}(i, l, f) \}$;
3. set $s(e) \triangleq \sum_{l,f} g_{l,f}$.

► **Example 4.13.** Call e the expression $o(a + \top o)\top a^+$ from the introduction (\star), where $o \triangleq (aa)^*a$ is an expression for the set of words of as of odd length, and $a^+ \triangleq a^*a$ is an expression for the set of non-empty words of as . Let us compute $s(e)$ using the following automaton for $[e]$, where i is the initial state, and f is the only final state.



We can easily describe various languages of interest in this automaton:

x	y	$\mathcal{L}(x, y)$
i	p	$[o]$
q	r	$[o]$
s	f	$[a^+]$

x	y	$\mathcal{L}(x, y)$
i	r	$[o(a + \top o)]$
q	f	$[o\top a^+]$
s	p	\emptyset

There are exactly two \top -transitions, so that we have five lists in $\Delta(\top)^{\otimes}$ to consider for $s(e)$:

1. the empty list, contributing $\mathcal{L}(i, f) = [e]$ to $E[e]$;
2. $[\langle p, q \rangle]$, not contributing since $\mathcal{L}(i, p) \cap \mathcal{L}(q, f) = [o] \cap [o\top a^+] = \emptyset$;
3. $[\langle r, s \rangle]$, contributing $\mathcal{L}(i, r) \cap \mathcal{L}(s, f) = [o(a + \top o)] \cap [a^+] = [oa]$;
4. $[\langle p, q \rangle; \langle r, s \rangle]$, contributing $\mathcal{L}(i, p) \cap \mathcal{L}(q, r) \cap \mathcal{L}(s, f) = [o] \cap [o] \cap [a^+] = [o]$;
5. $[\langle r, s \rangle; \langle p, q \rangle]$, not contributing since $\mathcal{L}(i, r) \cap \mathcal{L}(s, p) \cap \mathcal{L}(q, f) = \emptyset$;

So in the end, $s(e)$ can simply be taken to be $e + oa + o$. \blacktriangleleft

► **Proposition 4.14.** *For all expressions e , we have*

- (i) $[s(e)] = E[e]$, and
- (ii) $\text{KA}_F \vdash e = s(e)$.

Proof. The first item holds by definition of s and Proposition 4.11. We prove two inequations for the second item. Taking $n = 0$ in the definition of E , we have $[e] \subseteq E[e] = [s(e)]$, so that $\text{KA} \vdash e \leq s(e)$ by completeness of KA (\dagger). For the converse implication, it suffices to prove that for all $l \in \Delta(\top)^\otimes$ and all $f \in F$, the expression $g_{l,f}$ from Definition 4.12 is provably smaller than e in KA_F . Let us abbreviate $g_{l,f}$ as g , let n be the length of l , and let $\langle p_j, q_j \rangle_{j \leq n}$ be the successive elements of $\text{pr}(i, l, f)$ (so that $p_0 = i$ and $q_n = f$). We have

$$\begin{aligned}
& [g(\top g)^n] \\
&= [g] \cdot \{\top\} \cdot [g] \cdot \dots \cdot \{\top\} \cdot [g] && ([\cdot] \text{ is a homomorphism}) \\
&\subseteq \mathcal{L}(p_0, q_0) \cdot \{\top\} \cdot \mathcal{L}(p_1, q_1) \cdot \dots \cdot \{\top\} \cdot \mathcal{L}(p_n, q_n) \\
&\quad \text{(by definition of } g, [g] \text{ is contained in each } \mathcal{L}(p_j, q_j)) \\
&\subseteq \mathcal{L}(p_0, q_0) \cdot \mathcal{L}(q_0, p_1) \cdot \mathcal{L}(p_1, q_1) \cdot \dots \cdot \mathcal{L}(q_{n-1}, p_n) \cdot \mathcal{L}(p_n, q_n) \\
&\quad \text{(since } l \in \Delta(\top)^\otimes, \text{ we have } \langle q_j, p_{j+1} \rangle \in \Delta(\top), \text{ and thus } \top \in \mathcal{L}(q_j, p_{j+1})) \\
&\subseteq \mathcal{L}(i, f) \subseteq [e] && (p_0 = i, q_n = f, \text{ and definition of } \mathcal{L})
\end{aligned}$$

We deduce $\text{KA} \vdash g(\top g)^n \leq e$ by completeness of KA (\dagger), and we conclude by prepending n applications of axiom (F): $\text{KA}_F \vdash g \leq g(\top g)^n \leq e$. ◀

► **Example 4.15.** Continuing Example 4.13, we check that both $oa \leq oa\top oa \leq e$ and $o \leq o\top o\top o \leq e$ are derivable in KA_F , in both cases using axiom (F) for the first inequation (once or twice), and KA completeness for the second one. Also observe that $[a^+] = [oa + o]$, so that $\text{KA} \vdash a^+ = oa + o$ once again by KA completeness. Putting everything together, we obtain a derivation of the following shape for the law (\star) from the introduction.

$$\text{KA}_F \vdash a^+ = oa + o \leq oa\top oa + o\top o\top o \leq e \quad \blacktriangleleft$$

More generally, we can combine all the above results to obtain our main theorem:

► **Theorem 4.16.** *For all regular expressions with top e, f , we have*

$$\text{REL} \models e = f \iff C_F[e] = C_F[f] \iff \text{KA}_F \vdash e = f$$

Proof. We have

$$\begin{aligned}
& \text{REL} \models e = f \\
&\iff C_F[e] = C_F[f] && \text{(Proposition 4.8)} \\
&\iff E(C_T[e]) = E(C_T[f]) && \text{(by Proposition 4.7)} \\
&\iff [s(r(e))] = [s(r(f))] && \text{(by Propositions 3.4(i) and 4.14(i))} \\
&\iff \text{KA} \vdash s(r(e)) = s(r(f)) && \text{(by completeness of KA } (\dagger)) \\
&\implies \text{KA}_F \vdash e = f && \text{(by transitivity and Propositions 3.4(ii) and 4.14(ii))} \\
&\implies \text{REL} \models e = f && \text{(soundness of KA}_F \text{ axioms w.r.t. REL)}
\end{aligned}$$

The above proof follows the same strategy as the one for Theorem 3.5. Like there, the right-to-left implication of the second equivalence is an instance of [11, Theorem 2], and we use reductions only for the left-to-right part of this equivalence. ◀

Like for Theorem 3.5, we could complete the statement of Theorem 4.16 with “... $\iff [s(r(e))] = [s(r(f))]$ ”. This gives decidability since the function s is computable, but this does not give a reasonable algorithm: given an expression e , the size of $s(e)$ (or of an automaton for it) might be very big. We leave open the question of whether there is a better algorithm, hopefully in PSPACE.

5 Relations with a greatest element

A *generalised S_{\top} -algebra of relations* is an S -subalgebra A of an algebra of relations such that A has a greatest element, seen as an S_{\top} -algebra by using this greatest element for the constant \top . We write REL' for the class of all generalised S_{\top} -algebras of relations.

Intuitively, REL' consists of models of binary relations where \top is not necessarily the full relation, only a greatest element. As an example, consider relations R over the natural numbers such that $i \leq j$ whenever $i R j$. Those form an S -algebra with greatest element the order relation \leq itself, which is not the full relation.

In the literature, REL' is sometimes preferred over REL because it is closed under taking subalgebras and products, and actually forms a quasivariety [1]. (In contrast, it is not clear whether REL is closed under products: the two obvious ways of embedding a pair of relations into a new relation fail to preserve either union or top – REL as defined here is not closed under taking subalgebras either, but defining it in such a way would not change the results from the present paper.)

The equational theory of REL' differs from that of REL . For instance, the previous example of ordered relations shows that $\text{REL}' \not\models x \leq x \cdot \top \cdot x$. Indeed, for $x = \{\langle 0, 1 \rangle\}$, $x \cdot \top \cdot x$ is empty since \top does not relate 1 to 0.

We show below that the equational theory of REL' actually coincides with that of LANG , and can thus be axiomatised by KA_{\top} .

► **Proposition 5.1.** *Every member of LANG embeds into a member of REL' .*

Proof. We adapt the technique used by Pratt for Kleene algebras (without top) [31, third page] and later reused by Kozen and Smith for Kleene algebras with tests [20, Lemma 5]. For a set X , let $M(X)$ be the set of relations R on X^* such that for all words u, v , u is a prefix of v whenever $u R v$. The S -operations on relations restrict to $M(X)$, so that $M(X)$ is an S -algebra, and setting $\top \triangleq \{\langle u, uv \rangle \mid u, v \in X^*\}$ turns it into a member of REL' . We embed a member $\mathcal{P}(X^*)$ of LANG into $M(X)$ as follows:

$$\begin{aligned} \iota : \mathcal{P}(X^*) &\rightarrow M(X) \\ L &\mapsto \{\langle u, uv \rangle \mid u \in X^*, v \in L\} \end{aligned}$$

The function ι is easily shown to be an S_{\top} -algebra homomorphism, and it is injective (since, e.g., $L = \{u \mid \langle \epsilon, u \rangle \in \iota(L)\}$). ◀

Note that it is crucial that we consider REL' rather than REL here: the above construction would not give an S_{\top} -algebra homomorphism if we were not restricting to relations of a certain shape: \top would not be preserved.

► **Corollary 5.2.** *For all regular expressions with top, we have*

$$\text{LANG} \models e = f \iff \text{REL}' \models e = f \iff \text{KA}_{\top} \vdash e = f$$

Proof. That $\text{REL}' \models e = f$ entails $\text{LANG} \models e = f$ is a direct consequence of Proposition 5.1. That $\text{KA}_T \vdash e = f$ entails $\text{REL}' \models e = f$ follows from the soundness of KA_T axioms w.r.t. REL' . We conclude by Theorem 3.5. \blacktriangleleft

Similarly to REL' , we can define a class LANG' of S_\top -algebras which is closed under taking subalgebras and where \top is not necessarily the full language. However, unlike with REL' and REL , the equational theory of LANG' coincides with that of LANG (and REL'). Indeed the axioms of KA_T remain sound for LANG' .

References

- 1 H. Andr eka and S. Mikul as. Axiomatizability of positive algebras of binary relations. *Algebra Universalis*, 66(1):7–34, 2011. doi:10.1007/s00012-011-0142-3.
- 2 H. Andr eka, S. Mikul as, and I. N emeti. The equational theory of Kleene lattices. *Theoretical Computer Science*, 412(52):7099–7108, 2011. doi:10.1016/j.tcs.2011.09.024.
- 3 H. Andr eka and D.A. Bredikhin. The equational theory of union-free algebras of relations. *Algebra Universalis*, 33(4):516–532, 1995. doi:10.1007/BF01225472.
- 4 Stephen L. Bloom, Zolt an  sik, and Gheorghe Stefanescu. Notes on equational theories of relations. *Algebra Universalis*, 33(1):98–126, 1995. doi:10.1007/BF01190768.
- 5 Maurice Boffa. Une remarque sur les syst emes complets d’identit es rationnelles. *Informatique Th eorique et Applications*, 24:419–428, 1990. URL: http://archive.numdam.org/article/ITA_1990__24_4_419_0.pdf.
- 6 Maurice Boffa. Une condition impliquant toutes les identit es rationnelles. *Informatique Th eorique et Applications*, 29(6):515–518, 1995. URL: http://www.numdam.org/article/ITA_1995__29_6_515_0.pdf.
- 7 Paul Brunet and Damien Pous. Petri automata for Kleene allegories. In *LICS*, pages 68–79. ACM, 2015. doi:10.1109/LICS.2015.17.
- 8 Paul Brunet and Damien Pous. Algorithms for Kleene algebra with converse. *Journal of Logical and Algebraic Methods in Programming*, 85(4):574–594, 2016. doi:10.1016/j.jlamp.2015.07.005.
- 9 J.H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall mathematics series. Chapman and Hall, 1971. URL: <https://books.google.nl/books?id=1KAXc5TpEV8C>.
- 10 Anupam Das, Amina Doumane, and Damien Pous. Left-handed completeness for Kleene algebra, via cyclic proofs. In *LPAR*, volume 57 of *EPiC Series in Computing*, pages 271–289. EasyChair, 2018. doi:10.29007/hzq3.
- 11 Amina Doumane, Denis Kuperberg, Pierre Pradic, and Damien Pous. Kleene algebra with hypotheses. In *FoSSaCS*, volume 11425 of *LNCS*, pages 207–223. Springer, 2019. doi:10.1007/978-3-030-17127-8_12.
- 12 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.
- 13 Zolt an  sik and Laszlo Bern atsky. 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.
- 14 P. Freyd and A. Scedrov. *Categories, Allegories*. North Holland, 1990.
- 15 Tobias Kapp e, Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. Concurrent Kleene algebra with observations: from hypotheses to completeness. In *FoSSaCS*, volume 12077 of *LNCS*, pages 381–400. Springer, 2020. doi:10.1007/978-3-030-45231-5_20.
- 16 S. C. Kleene. Representation of events in nerve nets and finite automata. In *Automata Studies*, pages 3–41. Princeton University Press, 1956. URL: http://www.rand.org/pubs/research_memoranda/2008/RM704.pdf.

- 17 D. Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 19(3):427–443, May 1997. doi:10.1145/256167.256195.
- 18 D. Kozen. On Hoare logic and Kleene algebra with tests. *ACM Transactions on Computational Logic*, 1(1):60–76, 2000. doi:10.1145/343369.343378.
- 19 D. Kozen and M.-C. Patron. Certification of compiler optimizations using Kleene algebra with tests. In *CL2000*, volume 1861 of *LNAI*, pages 568–582. Springer, 2000. doi:10.1007/3-540-44957-4_38.
- 20 D. Kozen and F. Smith. Kleene algebra with tests: Completeness and decidability. In *CSL*, volume 1258 of *LNCS*, pages 244–259. Springer, September 1996. doi:10.1007/3-540-63172-0_43.
- 21 Dexter Kozen. A completeness theorem for Kleene Algebras and the algebra of regular events. In *LICS*, pages 214–225. IEEE Computer Society, 1991. doi:10.1109/LICS.1991.151646.
- 22 Dexter Kozen and Alexandra Silva. Left-handed completeness. In *RAMiCS*, volume 7560 of *LNCS*, pages 162–178. Springer, 2012. doi:10.1007/978-3-642-33314-9_11.
- 23 A. Krauss and T. Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *Journal of Automated Reasoning*, 49(1):95–106, 2012. doi:10.1007/s10817-011-9223-4.
- 24 D. Krob. Complete systems of B-rational identities. *Theoretical Computer Science*, 89(2):207–343, 1991. doi:10.1016/0304-3975(91)90395-I.
- 25 Konstantinos Mamouras. Equational theories of abnormal termination based on Kleene algebra. In *FoSSaCS*, volume 10203 of *LNCS*, pages 88–105. Springer, 2017. doi:10.1007/978-3-662-54458-7_6.
- 26 Yoshiki Nakamura. Partial derivatives on graphs for Kleene allegories. In *LICS*, pages 1–12. IEEE, 2017. doi:10.1109/LICS.2017.8005132.
- 27 Peter W. O’Hearn. Incorrectness logic. *Proc. ACM Program. Lang.*, 4(POPL):10:1–10:32, 2020. doi:10.1145/3371078.
- 28 Damien Pous. Kleene Algebra with Tests and Coq tools for while programs. In *ITP*, volume 7998 of *LNCS*, pages 180–196. Springer, 2013. doi:10.1007/978-3-642-39634-2_15.
- 29 Damien Pous. On the positive calculus of relations with transitive closure. In *STACS*, volume 96 of *LIPICs*, pages 3:1–3:16. Schloss Dagstuhl, 2018. doi:10.4230/LIPICs.STACS.2018.3.
- 30 Damien Pous, Jurriaan Rot, and Jana Wagemaker. On tools for completeness of Kleene algebra with hypotheses. In *RAMiCS*, volume 13027 of *LNCS*, pages 378–395. Springer, 2021. doi:10.1007/978-3-030-88701-8_23.
- 31 V. R. Pratt. Dynamic algebras and the nature of induction. In *ACM Symposium on Theory of Computing*, STOC ’80, pages 22–28, New York, NY, USA, 1980. Association for Computing Machinery. doi:10.1145/800141.804649.
- 32 V.N. Redko. On the algebra of commutative events. *Ukrainian Math. Zh.*, 16:185–195, 1964.
- 33 Cheng Zhang, Arthur Azevedo de Amorim, and Marco Gaboardi. On incorrectness logic and Kleene algebra with top and tests. *Proc. ACM Program. Lang.*, 6(POPL):1–30, 2022. doi:10.1145/3498690.

A Proof of Theorem 4.4

We give here a proof of Theorem 4.4. Variants of this theorem appeared for Kleene allegories without top in [7, Theorem 6], and for Kleene allegories with top in [29, Theorem 16]. The latter variant subsumes Theorem 4.4: it deals with a strictly larger fragment of relation algebra. We nevertheless give a proof here for the sake of completeness.

First we observe that valuations into relational models are very close to (potentially infinite) graphs in the sense of Definition 4.2: it suffices to adjoin to them an input and an output.

► **Definition A.1** (Graph of a valuation). *Let $\sigma : \Sigma \rightarrow \mathcal{P}(X \times X)$ be a valuation of Σ into relations on some set X . For all elements $i, j \in X$, we define the graph $\langle \sigma, i, j \rangle \triangleq \langle X, F, i, j \rangle$ where $F \triangleq \{\langle x, a, y \rangle \mid a \in \Sigma, \langle x, y \rangle \in \sigma(a)\}$.*

The first key lemma characterises evaluation of expressions not using $0, +, \cdot^*$ in a relational model, in terms of graph homomorphisms. In our case, expressions not using $0, +, \cdot^*$ can be represented by words with top. Such a lemma appeared first in [3, Lemma 3] for a signature including intersection and converse, but not top. Under its original formulation, its extension to cover top is trivial once we realise that the graph of \top should simply be a graph without edges and exactly two vertices (the input and the output).

► **Lemma A.2.** *Let $\sigma : \Sigma \rightarrow \mathcal{P}(X \times X)$ be a valuation of Σ into a member of REL. For all words $u \in \Sigma_{\top}^*$, we have*

$$\langle i, j \rangle \in \hat{\sigma}(u) \iff \langle \sigma, i, j \rangle \triangleleft g(u)$$

Proof. By induction on u .

- if u is empty, then both sides reduce to the condition $i = j$;
- if u is a letter a , then both sides reduce to the condition $\langle i, j \rangle \in \sigma(a)$;
- if u is \top , then both sides hold independently of i, j ;
- if $u = vw$ for two smaller words v, w then we have

$$\begin{aligned} & \langle i, j \rangle \in \hat{\sigma}(vw) \\ \Leftrightarrow & \exists k, \langle i, k \rangle \in \hat{\sigma}(v) \wedge \langle k, j \rangle \in \hat{\sigma}(w) && \text{(by definition)} \\ \Leftrightarrow & \exists k, \langle \sigma, i, k \rangle \triangleleft g(v) \wedge \langle \sigma, k, j \rangle \triangleleft g(w) && \text{(by induction hypothesis on } v \text{ and } w) \\ \Leftrightarrow & \langle \sigma, i, j \rangle \triangleleft g(vw) \end{aligned}$$

(The last equivalence comes from a simple analysis of the homomorphisms whose source is a sequential composition of two graphs – see, e.g., [3, Lemma 2(ii)].) ◀

The second key lemma characterises the evaluation of an arbitrary expression in terms of (the evaluations of) the words in the language of that expression. Variants of such a lemma often appear in the literature for *star-continuous* models, rather than just relational ones (e.g., [20, Lemma 4]).

► **Lemma A.3.** *Let $\sigma : \Sigma \rightarrow \mathcal{P}(X \times X)$ be a valuation of Σ into a member of REL. For all regular expressions with top e , we have*

$$\hat{\sigma}(e) = \bigcup_{u \in [e]} \hat{\sigma}(u)$$

Proof. By an easy induction on e , using distributivity of \cdot over arbitrary unions in REL. ◀

Equipped with those two lemmas, we obtain the announced theorem.

► **Theorem A.4.** *For all regular expressions with top e, f , we have:*

$$\text{REL} \models e \leq f \iff \forall u \in [e], \exists v \in [f], g(u) \triangleleft g(v)$$

Proof. For the forward implication, assume $\text{REL} \models e \leq f$ and let $u \in [e]$. Let n be the length of u and consider relations on $[0; n]$, a member of REL. Define $\sigma : \Sigma \rightarrow \mathcal{P}([0; n] \times [0; n])$ by $\langle i, j \rangle \in \sigma(a)$ if the i -th letter of u is a and $j = i + 1$. The graph $g(u)$ is nothing but $\langle \sigma, 0, n \rangle$, so that we have $\langle 0, n \rangle \in \hat{\sigma}(u)$ by Lemma A.2, using the identity graph homomorphism. Thus we

26:18 Completeness Theorems for Kleene Algebra with Top

consecutively get $\langle 0, n \rangle \in \hat{\sigma}(e)$ by Lemma A.3, $\langle 0, n \rangle \in \hat{\sigma}(f)$ by assumption, and $\langle 0, n \rangle \in \hat{\sigma}(v)$ for some $v \in [f]$ by Lemma A.3 again. Lemma A.2 finally gives $g(u) = \langle \sigma, 0, n \rangle \triangleleft g(v)$, as required.

For the backward implication, assume the right-hand side and let $\sigma : \Sigma \rightarrow \mathcal{P}(X \times X)$ be a valuation into a member of REL. For all $i, j \in X$, we have

$$\begin{aligned}
 & \langle i, j \rangle \in \hat{\sigma}(e) \\
 \Leftrightarrow & \langle i, j \rangle \in \hat{\sigma}(u) \text{ for some } u \in [e] && \text{(by Lemma A.3)} \\
 \Leftrightarrow & \langle \sigma, i, j \rangle \triangleleft g(u) \text{ for some } u \in [e] && \text{(by Lemma A.2)} \\
 \Rightarrow & \langle \sigma, i, j \rangle \triangleleft g(u) \text{ for some } u, v \text{ s.t. } v \in [f] \text{ and } g(u) \triangleleft g(v) && \text{(by assumption)} \\
 \Rightarrow & \langle \sigma, i, j \rangle \triangleleft g(v) \text{ for some } v \in [f] && \text{(by transitivity of } \triangleleft \text{)} \\
 \Leftrightarrow & \langle i, j \rangle \in \hat{\sigma}(v) \text{ for some } v \in [f] && \text{(by Lemma A.2)} \\
 \Leftrightarrow & \langle i, j \rangle \in \hat{\sigma}(f) && \text{(by Lemma A.3)}
 \end{aligned}$$

Whence $\hat{\sigma}(e) \subseteq \hat{\sigma}(f)$, and thus $\text{REL} \models e \leq f$ as required. \blacktriangleleft

Expressiveness and Decidability of Temporal Logics for Asynchronous Hyperproperties

Laura Bozzelli ✉

University of Napoli “Federico II”, Italy

Adriano Peron ✉

University of Napoli “Federico II”, Italy

César Sánchez ✉ 

IMDEA Software Institute, Madrid, Spain

Abstract

Hyperproperties are properties of systems that relate different execution traces, with many applications from security to symmetry, consistency models of concurrency, etc. In recent years, different linear-time logics for specifying *asynchronous* hyperproperties have been investigated. Though model checking of these logics is undecidable, useful decidable fragments have been identified with applications e.g. for asynchronous security analysis. In this paper, we address expressiveness and decidability issues of temporal logics for asynchronous hyperproperties. We compare the expressiveness of these logics together with the extension $S1S[E]$ of $S1S$ with the equal-level predicate by obtaining an almost complete expressiveness picture. We also study the expressive power of these logics when interpreted on singleton sets of traces. We show that for two asynchronous extensions of HyperLTL, checking the existence of a singleton model is already undecidable, and for one of them, namely Context HyperLTL (HyperLTL_C), we establish a characterization of the singleton models in terms of the extension of standard $\text{FO}[\langle \rangle]$ over traces with addition. This last result generalizes the well-known equivalence between $\text{FO}[\langle \rangle]$ and LTL. Finally, we identify new boundaries on the decidability of model checking HyperLTL_C .

2012 ACM Subject Classification Theory of computation \rightarrow Logic and verification

Keywords and phrases Hyperproperties, Asynchronous hyperproperties, Temporal logics for hyperproperties, Expressiveness, Decidability, Model checking

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.27

Related Version *Full Version*: <https://arxiv.org/abs/2207.02956> [7]

Funding César Sánchez: funded in part by the Madrid Regional Government under project “S2018/TCS-4339 (BLOQUES-CM)” and by a research grant from Nomadic Labs and the Tezos Foundation.

1 Introduction

Hyperproperties. In the last decade, a novel specification paradigm has been introduced that generalizes traditional trace properties by properties of sets of traces, the so called *hyperproperties* [9]. Hyperproperties relate execution traces of a reactive system and are useful in many settings. In the area of information flow control, hyperproperties can formalize security policies (like noninterference [18, 26] and observational determinism [32]) which compare observations made by an external low-security agent along traces resulting from different values of not directly observable inputs. These security requirements go, in general, beyond regular properties and cannot be expressed in classical regular temporal logics such as LTL [27], CTL, and CTL^* [12]. Hyperproperties also have applications in other settings, such as the symmetric access to critical resources in distributed protocols [15], consistency models in concurrent computing [4], and distributed synthesis [14].



© Laura Bozzelli, Adriano Peron, and César Sánchez;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Sławomir Lasota, and Anca Muscholl; Article No. 27; pp. 27:1–27:16



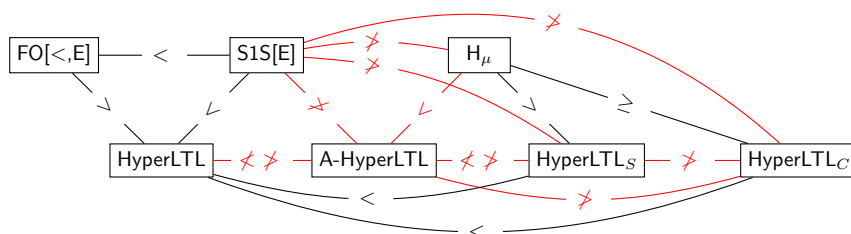
Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In the context of model checking of finite-state reactive systems, many temporal logics for hyperproperties have been proposed [11, 8, 5, 28, 13, 10, 21] for which model checking is decidable, including HyperLTL [8], HyperCTL* [8], HyperQPTL [28, 10], and HyperPDL- Δ [21] which extend LTL, CTL*, QPTL [29], and PDL [17], respectively, by explicit first-order quantification over traces and trace variables to refer to multiple traces at the same time. The semantics of all these logics is *synchronous* and the temporal modalities are evaluated by a lockstepwise traversal of all the traces assigned to the quantified trace variables.

A different approach for the formalization of synchronous hyper logics is based on hyper variants of monadic second-order logic over traces or trees [10]. For the linear-time setting, we recall the logic S1S[E] [10] (and its first-order fragment FO[\langle, E] [16]) which syntactically extends monadic second-order logic of one successor S1S with the *equal-level predicate* E , which relate the same time points on different traces. Another class of hyperlogics is obtained by adopting a *team semantics* for standard temporal logics, in particular, LTL [24, 25, 31, 19].

Asynchronous extensions of Hyper logics. Many application domains require asynchronous properties that relate traces at distinct time points which can be arbitrarily far from each other. For example, asynchronous specifications are needed to reason about a multithreaded environment in which threads are not scheduled in lockstep, and traces associated with distinct threads progress at different speed. Asynchronous hyperproperties are also useful in information-flow security where an observer is not time-sensitive, so the observer cannot distinguish consecutive time points along an execution having the same observation. This again requires asynchronously matching sequences of observations along distinct execution traces. A first systematic study of asynchronous hyperproperties is done in [22], where two powerful and expressively equivalent linear-time asynchronous formalisms are introduced: the temporal fixpoint calculus H_μ and an automata-theoretic formalism where the quantifier-free part of a specification is expressed by the class of parity multi-tape Alternating Asynchronous Word Automata (AAWA) [22]. While the expressive power of the quantifier-part of HyperLTL is just that of LTL over tuples of traces of fixed arity (*multi-traces*), AAWA allow to specify very expressive non-regular multi-trace properties. Model checking against H_μ or its AAWA-based counterpart is undecidable even for the quantifier alternation-free fragment. In [22], two decidable subclasses of parity AAWA are identified which express only multi-trace ω -regular properties and lead to two H_μ fragments with decidable model checking. More recently, other temporal logics [2, 6] which syntactically extend HyperLTL have been introduced for expressing asynchronous hyperproperties. *Asynchronous HyperLTL* (A-HyperLTL) [2], useful for asynchronous security analysis, models asynchronicity by means of an additional quantification layer over the so called *trajectories*. Intuitively, a trajectory controls the relative speed at which traces progress by choosing at each instant which traces move and which traces stutter. The general logic also has an undecidable model-checking problem, but [2] identifies practical decidable fragments, and reports an empirical evaluation. *Stuttering HyperLTL* (HyperLTL_S) and *Context HyperLTL* (HyperLTL_C) are introduced in [6] as more expressive asynchronous extensions of HyperLTL. HyperLTL_S uses relativized versions of the temporal modalities with respect to finite sets Γ of LTL formulas. Intuitively, these modalities are evaluated by a lockstepwise traversal of the sub-traces of the given traces which are obtained by removing “redundant” positions with respect to the pointwise evaluation of the LTL formulas in Γ . HyperLTL_C extends HyperLTL by unary modalities $\langle C \rangle$ parameterized by a non-empty subset C of trace variables – called the *context* – which restrict the evaluation of the temporal modalities to the traces associated with the variables in C . Both HyperLTL_S and HyperLTL_C are subsumed by H_μ and still have an undecidable model-checking problem, and fragments of the two logics with a decidable model-checking have been investigated [6].



■ **Figure 1** Expressiveness comparisons between linear-time hyper logics.

Our contribution. In this paper, we study expressiveness and decidability of asynchronous extensions of HyperLTL [22, 2, 6]. Our main goal is to compare the expressive power of these logics together with the known logics for linear-time hyperproperties based on the equal-level predicate whose most powerful representative is S1S[E]. The first-order fragment $\text{FO}[\langle, E]$ of S1S[E] is already strictly more expressive than HyperLTL [16] and, unlike S1S[E], its model-checking problem is decidable [10]. We obtain an almost complete expressiveness picture, summarized in Figure 1, where novel results are annotated in red. In particular, for A-HyperLTL, we show that although HyperLTL and A-HyperLTL are expressively incomparable, HyperLTL can be embedded into A-HyperLTL using a natural encoding. We also establish that A-HyperLTL is strictly less expressive than H_μ and its AAWA counterpart. For the relative expressiveness of A-HyperLTL, HyperLTL_S, and HyperLTL_C, we prove that A-HyperLTL and HyperLTL_S are expressively incomparable, and that HyperLTL_C is not subsumed by A-HyperLTL or by HyperLTL_S. The question of whether A-HyperLTL and HyperLTL_S are subsumed or not by HyperLTL_C remains open. Additionally, we show that each of these logics is not subsumed by S1S[E]. This last result solves a recent open question [22, 2].

Since hyperproperties are a generalization of trace properties, we also investigate the expressive power of the considered asynchronous extensions of HyperLTL when interpreted on singleton sets of traces. For HyperLTL and its more expressive extension HyperLTL_S, singleton models are just the ones whose traces are LTL-definable and checking the existence of such a model (*single-trace satisfiability*) is decidable and PSPACE-complete. On the other hand, we show that for both A-HyperLTL and HyperLTL_C, single-trace satisfiability is highly undecidable being Σ_1^1 -hard. Moreover, for HyperLTL_C extended with past temporal modalities, we provide a nice characterization of the singleton models which generalizes the well-known equivalence of LTL and first-order logic $\text{FO}[\langle]$ over traces established by Kamp’s theorem. We show that over singleton models, HyperLTL_C with past corresponds to the extension $\text{FO}[\langle, +]$ of $\text{FO}[\langle]$ with addition over variables.

Finally, we investigate the decidability frontier for model-checking HyperLTL_C by enforcing the undecidability result of [6] and by identifying a maximal fragment of HyperLTL_C for which model checking is decidable. This fragment subsumes HyperLTL and can be translated into $\text{FO}[\langle, E]$. Due to lack of space, many proofs are omitted and included in the longer version of this paper [7].

2 Preliminaries

Let \mathbb{N} be the set of natural numbers. For all $i, j \in \mathbb{N}$, $[i, j]$ denotes the set of natural numbers h such that $i \leq h \leq j$. Given a word w over some alphabet Σ , $|w|$ is the length of w ($|w| = \infty$ if w is infinite). For each $0 \leq i < |w|$, $w(i)$ is the $(i + 1)^{\text{th}}$ symbol of w , and w^i is the suffix of w from position i , i.e., the word $w(i)w(i + 1) \dots$.

We fix a finite set AP of atomic propositions. A *trace* is an infinite word over 2^{AP} . A *pointed trace* is a pair (π, i) consisting of a trace π and a position $i \in \mathbb{N}$. Two traces π and π' are *stuttering equivalent* if there are two infinite sequences of positions $0 = i_0 < i_1 \dots$ and $0 = i'_0 < i'_1 \dots$ s.t. for all $k \geq 0$ and for all $\ell \in [i_k, i_{k+1} - 1]$ and $\ell' \in [i'_k, i'_{k+1} - 1]$, $\pi(\ell) = \pi'(\ell')$. The trace π' is a *stuttering expansion* of the trace π if there is an infinite sequence of positions $0 = i_0 < i_1 \dots$ such that for all $k \geq 0$ and for all $\ell \in [i_k, i_{k+1} - 1]$, $\pi'(\ell) = \pi(k)$.

Kripke structures. A *Kripke structure* (over AP) is a tuple $\mathcal{K} = \langle S, S_0, E, V \rangle$, where S is a finite set of states, $S_0 \subseteq S$ is the set of initial states, $E \subseteq S \times S$ is a transition relation and $V : S \rightarrow 2^{AP}$ is an *AP-valuation* of the set of states. A *path* of \mathcal{K} is an infinite sequence of states t_0, t_1, \dots such that $t_0 \in S_0$ and $(t_i, t_{i+1}) \in E$ for all $i \geq 0$. The Kripke structure \mathcal{K} induces the set $\mathcal{L}(\mathcal{K})$ of traces of the form $V(t_0), V(t_1), \dots$ such that t_0, t_1, \dots is a path of \mathcal{K} .

Relative Expressiveness. In Sections 3-4, we compare the expressiveness of various logics for linear-time hyperproperties. Let \mathcal{M} be a set of models (in our case, a model is a set of traces), and L and L' be two logical languages interpreted over models in \mathcal{M} . Given two formulas $\varphi \in L$ and $\varphi' \in L'$, we say that φ and φ' are *equivalent* if for each model $M \in \mathcal{M}$, M satisfies φ iff M satisfies φ' . The language L is *subsumed by* L' , denoted $L \leq L'$, if each formula in L has an equivalent formula in L' . The language L is *strictly less expressive than* L' (written $L < L'$) if $L \leq L'$ and there is a L' -formula which has no equivalent in L . Finally, two logics L and L' are *expressively incomparable* if both $L \not\leq L'$ and $L' \not\leq L$.

Linear-time hyper specifications. We consider an abstract notion of linear-time hyper specifications which are interpreted over sets of traces. We fix a finite set VAR of trace variables. A *pointed-trace assignment* Π is a partial mapping over VAR assigning to each trace variable x in its domain $\text{Dom}(\Pi)$ a pointed trace. The assignment Π is *initial* if for each $x \in \text{Dom}(\Pi)$, $\Pi(x)$ is of the form $(\pi, 0)$ for some trace π . For a variable $x \in \text{VAR}$ and a pointed trace (π, i) , we denote by $\Pi[x \mapsto (\pi, i)]$ the pointed-trace assignment having domain $\text{Dom}(\Pi) \cup \{x\}$ defined as: $\Pi[x \mapsto (\pi, i)](x) = (\pi, i)$ and $\Pi[x \mapsto (\pi, i)](y) = \Pi(y)$ if $y \neq x$.

A *multi-trace specification* $S(x_1, \dots, x_n)$ is a specification (in some formalism) parameterized by a subset $\{x_1, \dots, x_n\}$ of VAR whose semantics is given by a set Υ of *initial* pointed-trace assignments with domain $\{x_1, \dots, x_n\}$: we write $\Pi \models S(x_1, \dots, x_n)$ for the trace assignments Π in Υ . Given a class \mathcal{C} of multi-trace specifications, *linear-time hyper expressions* ξ over \mathcal{C} are defined as: $\xi \stackrel{\text{def}}{=} \exists x. \xi \mid \forall x. \xi \mid S(x_1, \dots, x_n)$, where $x, x_1, \dots, x_n \in \text{VAR}$, $S(x_1, \dots, x_n) \in \mathcal{C}$, and $\exists x$ (resp., $\forall x$) is the *hyper* existential (resp., universal) trace quantifier for variable x . An expression ξ is a *sentence* if every variable x_i in the multi-trace specification $S(x_1, \dots, x_n)$ of ξ is *not free* (i.e., x_i is in the scope of a quantifier for variable x_i). The *quantifier alternation depth* of ξ is the number of switches between \exists and \forall quantifiers in the quantifier prefix of ξ . For a set \mathcal{L} of traces and an initial pointed-trace assignment Π such that $\text{Dom}(\Pi)$ contains the free variables of ξ and the traces referenced by Π are in \mathcal{L} , the satisfaction relation $(\mathcal{L}, \Pi) \models \xi$ is inductively defined as follows:

$$\begin{aligned} (\mathcal{L}, \Pi) \models \exists x. \xi & \Leftrightarrow \text{for some trace } \pi \in \mathcal{L} : (\mathcal{L}, \Pi[x \mapsto (\pi, 0)]) \models \xi \\ (\mathcal{L}, \Pi) \models \forall x. \xi & \Leftrightarrow \text{for each trace } \pi \in \mathcal{L} : (\mathcal{L}, \Pi[x \mapsto (\pi, 0)]) \models \xi \\ (\mathcal{L}, \Pi) \models S(x_1, \dots, x_n) & \Leftrightarrow \Pi \models S(x_1, \dots, x_n) \end{aligned}$$

For a sentence ξ , we write $\mathcal{L} \models \xi$ to mean that $(\mathcal{L}, \Pi_\emptyset) \models \xi$, where Π_\emptyset is the empty assignment. If $\mathcal{L} \models \xi$ we say that \mathcal{L} is a *model* of ξ . If, additionally, \mathcal{L} is a singleton we call it a *single-trace model*. By restricting our attention to the single-trace models, a linear-time hyper sentence ξ denotes a trace property consisting of the traces π such that $\{\pi\} \models \xi$. For a class \mathcal{C} of multi-trace specifications, we consider the following decision problems:

- the *satisfiability* (resp., *single-trace satisfiability*) problem is checking for a linear-time hyper sentence ξ over \mathcal{C} , whether ξ has a model (resp., a single-trace model), and
- the model checking problem is checking for a Kripke structure \mathcal{K} and a linear-time hyper sentence ξ over \mathcal{C} , whether $\mathcal{L}(\mathcal{K}) \models \xi$.

For instance, HyperLTL formulas are linear-time hyper sentences over the class of multi-trace specifications, called *HyperLTL quantifier-free formulas*, obtained by standard LTL formulas [27] by replacing atomic propositions p with relativized versions $p[x]$, where $x \in \text{VAR}$. Intuitively, $p[x]$ asserts that p holds at the current position of the trace assigned to x . Given an HyperLTL quantifier-free formula $\psi(x_1, \dots, x_n)$, an initial pointed trace assignment Π such that $\text{Dom}(\Pi) \supseteq \{x_1, \dots, x_n\}$, and a position $i \geq 0$, the satisfaction relation $(\Pi, i) \models \psi$ is defined as a natural extension of the satisfaction relation $(\pi, i) \models \theta$ for LTL formulas θ and traces π . In particular,

- $(\Pi, i) \models p[x_k]$ if $\Pi(x_k) = (\pi, 0)$ and $p \in \pi(i)$,
- $(\Pi, i) \models X\psi$ if $(\Pi, i+1) \models \psi$, and
- $(\Pi, i) \models \psi_1 \cup \psi_2$ if there is $j \geq i$ such that $(\Pi, j) \models \psi_2$ and $(\Pi, k) \models \psi_1$ for all $k \in [i, j-1]$.

Asynchronous Word Automata and the Fixpoint Calculus H_μ . We shortly recall the framework of parity alternating asynchronous word automata (parity AAWA) [22], a class of finite-state automata for the asynchronous traversal of multiple infinite words. Intuitively, given $n \geq 1$, an AAWA with n tapes (n AAWA) has access to n infinite words over the input alphabet Σ and at each step, activates multiple copies where for each of them, there is exactly one input word whose current input symbol is consumed (i.e., the reading head of such word moves one position to the right). In particular, the target of a move of \mathcal{A} is encoded by a pair (q, i) , where q indicates the target state while the direction $i \in [1, n]$ indicates on which input word to progress. Details on the syntax and semantics of AAWA are given in [22, 7]. We denote by *Hyper AAWA* the class of linear-time hyper sentences over the multi-trace specifications given by parity AAWA. We also consider the fixpoint calculus H_μ introduced in [22] that provides a logical characterization of Hyper AAWA.

3 Advances in Asynchronous Extensions of HyperLTL

In this section, we investigate expressiveness and decidability issues on known asynchronous extensions of HyperLTL, namely, *Asynchronous HyperLTL* [2], *Stuttering HyperLTL* [6], and *Context HyperLTL* [6].

3.1 Results for Asynchronous HyperLTL (A-HyperLTL)

We first recall A-HyperLTL [2], a syntactical extension of HyperLTL which allows to express pure asynchronous hyperproperties. Then, we show that although A-HyperLTL does not subsume HyperLTL, HyperLTL can be embedded into A-HyperLTL by means of an additional proposition. Second, we establish that A-HyperLTL is subsumed by Hyper AAWA and the fixpoint calculus H_μ . Finally, we show that unlike HyperLTL, single-trace satisfiability of A-HyperLTL is undecidable.

The logic A-HyperLTL models the asynchronous passage of time between computation traces using the notion of a trajectory. Given a non-empty subset $V \subseteq \text{VAR}$, a *trajectory over V* is an infinite sequence t of non-empty subsets of V . Intuitively, the positions $i \geq 0$ along t model the global time flow and for each position $i \geq 0$, $t(i)$ determines the trace variables in V whose associated traces make progress at time i . The trajectory t is *fair* if for each $x \in V$, there are infinitely many positions i such that $x \in t(i)$.

A-HyperLTL formulas are linear-time hyper sentences over multi-trace specifications ψ , called *A-HyperLTL quantifier-free formulas*, where ψ is of the form $E\theta$ or $A\theta$ and θ is a HyperLTL quantifier-free formula: E is the existential trajectory modality and A is the universal trajectory modality. Given a pointed trace assignment Π and a trajectory t over $Dom(\Pi)$, the *successor of* (Π, t) , denoted by $(\Pi, t) + 1$, is defined as (Π', t') , where: (1) t' is the trajectory t^1 (the suffix of t from position 1), and (2) $Dom(\Pi') = Dom(\Pi)$ and for each $x \in Dom(\Pi)$ with $\Pi(x) = (\pi, i)$, $\Pi'(x) = (\pi, i + 1)$ if $x \in t(0)$, and $\Pi'(x) = \Pi(x)$ otherwise.

For each $k \geq 1$, we write $(\Pi, t) + k$ for denoting the pair (Π'', t'') obtained by k -applications of the successor function starting from (Π, t) . Given a HyperLTL quantifier-free formula θ such that $Dom(\Pi)$ contains the variables occurring in θ , the satisfaction relations $\Pi \models E\theta$, $\Pi \models A\theta$, and $(\Pi, t) \models \theta$ are defined as follows (we omit the semantics of the Boolean connectives):

$$\begin{aligned}
 \Pi \models E\theta &\Leftrightarrow \text{for some fair trajectory } t \text{ over } Dom(\Pi), (\Pi, t) \models \theta \\
 \Pi \models A\theta &\Leftrightarrow \text{for all fair trajectories } t \text{ over } Dom(\Pi), (\Pi, t) \models \theta \\
 (\Pi, t) \models p[x] &\Leftrightarrow \Pi(x) = (\pi, i) \text{ and } p \in \pi(i) \\
 (\Pi, t) \models X\theta &\Leftrightarrow (\Pi, t) + 1 \models \theta \\
 (\Pi, t) \models \theta_1 U \theta_2 &\Leftrightarrow \text{for some } i \geq 0 : (\Pi, t) + i \models \theta_2 \text{ and } (\Pi, t) + k \models \theta_1 \text{ for all } 0 \leq k < i
 \end{aligned}$$

We also exploit an alternative characterization of the semantics of quantifier-free A-HyperLTL formulas which easily follows from the definition of trajectories.

► **Proposition 1.** *Let θ be a quantifier-free HyperLTL formula over trace variables x_1, \dots, x_k , and let π_1, \dots, π_k be k traces. Then:*

- $\{x_1 \leftarrow (\pi_1, 0), \dots, x_k \leftarrow (\pi_k, 0)\} \models E\theta$ iff for all $i \in [1, k]$, there is a stuttering expansion π'_i of π_i such that $\{x_1 \leftarrow (\pi'_1, 0), \dots, x_k \leftarrow (\pi'_k, 0)\} \models \theta$.
- $\{x_1 \leftarrow (\pi_1, 0), \dots, x_k \leftarrow (\pi_k, 0)\} \models A\theta$ iff for all $i \in [1, k]$ and for all stuttering expansions π'_i of π_i , it holds that $\{x_1 \leftarrow (\pi'_1, 0), \dots, x_k \leftarrow (\pi'_k, 0)\} \models \theta$.

A-HyperLTL versus HyperLTL. We now show that, unlike other temporal logics for asynchronous hyperproperties (see Sections 3.2 and 3.3), A-HyperLTL does not subsume HyperLTL. Given an atomic proposition p , we consider the following linear-time hyperproperty.

p -synchronicity: a set \mathcal{L} of traces satisfies the p -synchronicity hyperproperty if for all traces $\pi, \pi' \in \mathcal{L}$ and positions $i \geq 0$, $p \in \pi(i)$ iff $p \in \pi'(i)$.

This hyperproperty can be expressed in HyperLTL as follows: $\forall x_1. \forall x_2. G(p[x_1] \leftrightarrow p[x_2])$. However, it cannot be expressed in A-HyperLTL (for details, see [7]).

► **Theorem 2.** *A-HyperLTL cannot express p -synchronicity. Hence, A-HyperLTL does not subsume HyperLTL.*

Though A-HyperLTL does not subsume HyperLTL, we can embed HyperLTL into A-HyperLTL by using an additional proposition $\# \notin AP$ as follows. We can ensure that along a trajectory, traces progress at each global instant by requiring that proposition $\#$ holds exactly at the even positions. Formally, given a trace π over AP , we denote by $enc_{\#}(\pi)$ the trace over $AP \cup \{\#\}$ defined as: $enc_{\#}(\pi)(2i) = \pi(2i) \cup \{\#\}$ and $enc_{\#}(\pi)(2i + 1) = \pi(2i + 1)$ for all $i \geq 0$. We extend the encoding $enc_{\#}$ to sets of traces \mathcal{L} and assignments Π over AP in the obvious way. For each $x \in VAR$, let $\theta_{\#}(x)$ be the following one-variable quantifier-free HyperLTL formula: $\#[x] \wedge G(\#[x] \leftrightarrow \neg X\#[x])$. It is easy to see that for a trace ρ over $AP \cup \{\#\}$, a stuttering expansion ρ' of ρ satisfies $\theta_{\#}(x)$ with x bound to ρ' iff $\rho' = \rho$ and ρ is the $\#$ -encoding of some trace over AP . It follows that satisfiability of an HyperLTL

formula φ can be reduced in linear-time to the satisfiability of the A-HyperLTL formula $\varphi_{\#}$ obtained from φ by replacing the quantifier-free part $\psi(x_1, \dots, x_k)$ of φ with the quantifier-free A-HyperLTL formula $E.(\psi(x_1, \dots, x_k) \wedge \bigwedge_{i \in [1, k]} \theta_{\#}(x_i))$. For model checking, given a Kripke structure $\mathcal{K} = \langle S, S_0, E, V \rangle$, we construct in linear-time a Kripke structure $\mathcal{K}_{\#}$ over $AP \cup \{\#\}$ such that $\mathcal{L}(\mathcal{K}_{\#}) = enc_{\#}(\mathcal{L}(\mathcal{K}))$. Formally, $\mathcal{K}_{\#} = \langle S \times \{0, 1\}, S_0 \times \{1\}, E', V' \rangle$ where $E' = \{((s, b), (s', 1 - b)) \mid (s, s') \in E \text{ and } b = 0, 1\}$, $V'((s, 1)) = V(s) \cup \{\#\}$ and $V'((s, 0)) = V(s)$ for all $s \in S$. Thus, we obtain the following result.

► **Theorem 3.** *Satisfiability (resp., model checking) of HyperLTL can be reduced in linear-time to satisfiability (resp., model checking) of A-HyperLTL.*

A-HyperLTL versus Hyper AAWA and H_{μ} . We show that A-HyperLTL is subsumed by Hyper AAWA and H_{μ} . To this purpose, we exhibit an exponential-time translation of quantifier-free A-HyperLTL formulas into equivalent parity AAWA.

► **Theorem 4.** *Given an A-HyperLTL quantifier-free formula ψ with trace variables x_1, \dots, x_n , one can build in singly exponential time a parity nAAWA \mathcal{A}_{ψ} over 2^{AP} accepting the set of n -tuples (π_1, \dots, π_n) of traces such that $(\{x_1 \leftarrow (\pi_1, 0), \dots, x_n \leftarrow (\pi_n, 0)\}) \models \psi$.*

Proof sketch. We first assume that ψ is of the form $E\theta$ for some HyperLTL quantifier-free formula θ . By an adaptation of the standard automata theoretic approach for LTL [30], we construct a *nondeterministic nAAWA (nNAWA)* $\mathcal{A}_{E\theta}$ equipped with standard generalized Büchi acceptance conditions which accepts a n -tuple (π_1, \dots, π_n) of traces iff there is a fair trajectory t such that $(\{x_1 \leftarrow (\pi_1, 0), \dots, x_n \leftarrow (\pi_n, 0)\}), t \models \theta$. By standard arguments, a generalized Büchi nNAWA can be converted in quadratic time into an equivalent parity nNAWA. The behaviour of the automaton is subdivided into phases where each phase intuitively corresponds to a global timestamp. During a phase, $\mathcal{A}_{E\theta}$ keeps tracks in its state of the guessed set of subformulas of θ that hold at the current global instant and guesses which traces progress at the next global instant by moving along a non-empty guessed set of directions in $\{1, \dots, n\}$ in turns. In particular, after a movement along direction i , the automaton keeps track in its state of the previous chosen direction i and *either* moves to the next phase, *or* remains in the current phase by choosing a direction $j > i$. The transition function in moving from the end of a phase to the beginning of the next phase captures the semantics of the next modalities and the “local” fixpoint characterization of the until modalities. Moreover, the generalized Büchi acceptance condition is used for ensuring the fulfillment of the liveness requirements θ_2 in the until sub-formulas $\theta_1 U \theta_2$, and for guaranteeing that the guessed trajectory is fair (i.e., for each direction $i \in [1, n]$, the automaton moves along i infinitely often). Details of the construction are given in [7].

Now, let us consider a quantifier-free A-HyperLTL formula of the form $A\theta$ with trace variables x_1, \dots, x_n , and let $\mathcal{A}_{E-\theta}$ be the parity nAAWA associated with the formula $E-\theta$. By [22], one can construct in linear-time (in the size of $\mathcal{A}_{E-\theta}$), a parity nAAWA $\mathcal{A}_{A\theta}$ accepting the complement of the language of n -tuples of traces accepted by $\mathcal{A}_{E-\theta}$. ◀

Thus, being H_{μ} and Hyper AAWA expressively equivalent, we obtain the following result.

► **Corollary 5.** *Hyper AAWA subsumes A-HyperLTL. H_{μ} also subsumes A-HyperLTL.*

Undecidability of single-trace satisfiability for A-HyperLTL. It is easy to see that for HyperLTL, single-trace satisfiability corresponds to LTL satisfiability (hence, it is PSPACE-complete). We show now that for A-HyperLTL, the problem is highly undecidable being at

least Σ_1^1 -hard. The crucial observation is that we can enforce alignment requirements on distinct stuttering expansions of the same trace which allow to encode recurrent computations of *non-deterministic 2-counter machines* [23]. Recall that such a machine is a tuple $M = \langle Q, \Delta, \delta_{init}, \delta_{rec} \rangle$, where Q is a finite set of (control) locations, $\Delta \subseteq Q \times L \times Q$ is a transition relation over the instruction set $L = \{\text{inc}, \text{dec}, \text{if_zero}\} \times \{1, 2\}$, and $\delta_{init} \in \Delta$ and $\delta_{rec} \in \Delta$ are two designated transitions, the initial and the recurrent one.

An M -configuration is a pair (δ, ν) consisting of a transition $\delta \in \Delta$ and a counter valuation $\nu : \{1, 2\} \rightarrow \mathbb{N}$. A computation of M is an *infinite* sequence of configurations of the form $((q_0, (op_0, c_0), q_1), \nu_0), ((q_1, (op_1, c_1), q_2), \nu_1), \dots$ such that for each $i \geq 0$:

- $\nu_{i+1}(3 - c_i) = \nu_i(3 - c_i)$;
- $\nu_{i+1}(c_i) = \nu_i(c_i) + 1$ if $op_i = \text{inc}$, and $\nu_{i+1}(c_i) = \nu_i(c_i) - 1$ if $op_i = \text{dec}$;
- $\nu_{i+1}(c_i) = \nu_i(c_i) = 0$ if $op_i = \text{if_zero}$.

The recurrence problem is to decide whether for a given machine M , there is a computation starting at the initial configuration (δ_{init}, ν_0) , where $\nu_0(c) = 0$ for each $c \in \{1, 2\}$, which visits δ_{rec} infinitely often. This problem is known to be Σ_1^1 -complete [23].

► **Theorem 6.** *The single-trace satisfiability problem of A-HyperLTL is Σ_1^1 -hard.*

Proof sketch. Let $M = \langle Q, \Delta, \delta_{init}, \delta_{rec} \rangle$ be a counter machine. We construct a two-variable A-HyperLTL formula φ_M such that M is a positive instance of the recurrence problem if and only if φ_M has a single-trace model. The set of atomic propositions is $\text{AP} \stackrel{\text{def}}{=} \Delta \cup \{c_1, c_2, \#, \text{beg}, \text{pad}\}$. Intuitively, propositions c_1 and c_2 are used to encode the values of the two counters in M and $\#$ is used to ensure that the values of the counters are not modified in the stuttering expansions of a trace encoding a computation of M . Proposition *beg* marks the beginning of a configuration code, and proposition *pad* is exploited for encoding a padding word at the end of a configuration code: formula φ_M will ensure that only these words can be “expanded” in the stuttering expansions of a trace. Formally, an M -configuration (δ, ν) is encoded by the finite words over 2^{AP} (called *segments*) of the form $\{\text{beg}, \delta\}P_1 \dots P_m \{\text{pad}\}^k$, where $k \geq 1$, $m = \max(\nu(1), \nu(2))$, and for all $i \in [1, m]$,

- $\emptyset \neq P_i \subseteq \{\#, c_1, c_2\}$,
- $\# \in P_i$ iff i is odd, and
- for all $\ell \in \{1, 2\}$, $c_\ell \in P_i$ iff $i \leq \nu(\ell)$.

A computation ρ of M is then encoded by the traces obtained by concatenating the codes of the configurations along ρ starting from the first one. The A-HyperLTL formula φ_M is given by $\exists x_1 \exists x_2. \mathbf{E} \psi$, where the quantifier-free HyperLTL formula ψ guarantees that for the two stuttering expansions π_1 and π_2 of the given trace π , the following holds:

- both π_1 and π_2 are infinite concatenations of segments;
- the first segment of π_1 encodes the initial configuration (δ_{init}, ν_0) of M and the second segment of π_1 encodes a configuration which is a successor of (δ_{init}, ν_0) in M ;
- δ_{rec} occurs infinitely often along π_1 ;
- for each $i \geq 2$, the $(i+1)^{\text{th}}$ segment s_2 of π_2 starts at the same position as the i^{th} segment s_1 of π_1 . Moreover, s_1 and s_2 have the same length and the configuration encoded by s_2 is a successor in M of the configuration encoded by s_1 .

Now, since π_1 and π_2 are stuttering expansions of the same trace π , the alternation requirement for proposition $\#$ in the encoding of an M -configurations ensures that π_1 and π_2 encode the same infinite sequence of M -configurations. Hence, φ_M has a single-trace model if and only if M is a positive instance of the recurrence problem. Details appear in [7]. ◀

3.2 Results for Stuttering HyperLTL (HyperLTL_S)

Stuttering HyperLTL (HyperLTL_S) [6] is an asynchronous extension of HyperLTL obtained by using *stutter-relativized* versions of the temporal modalities w.r.t. finite sets Γ of LTL formulas. In this section, we show that A-HyperLTL and HyperLTL_S are expressively incomparable.

In HyperLTL_S, the notion of successor of a position i along a trace π is relativized using a finite set Γ of LTL formulas. If in the interval $[i, \infty[$, the truth value of each formula in Γ does not change along π (i.e., for each $j \geq i$ and for each $\theta \in \Gamma$, $(\pi, i) \models \theta$ iff $(\pi, j) \models \theta$), then the Γ -successor of i in π coincides with the local successor $i + 1$. Otherwise, the Γ -successor of i in π is the smallest position $j > i$ such that the truth value of some formula θ in Γ changes in moving from i to j (i.e., for some $\theta \in \Gamma$, $(\pi, i) \models \theta$ iff $(\pi, j) \not\models \theta$). The Γ -successor induces a trace, called Γ -stutter trace of π and denoted by $stfr_{\Gamma}(\pi)$, obtained from π by repeatedly applying the Γ -successor starting from position 0, i.e. $stfr_{\Gamma}(\pi) \stackrel{\text{def}}{=} \pi(i_0)\pi(i_1)\dots$, where $i_0 = 0$ and i_{k+1} is the Γ -successor of i_k in π for all $k \geq 0$. Note that $stfr_{\Gamma}(\pi) = \pi$ if $\Gamma = \emptyset$. Given a pointed-trace assignment Π , the Γ -successor $succ_{\Gamma}(\Pi)$ of Π is the pointed trace-assignment with domain $Dom(\Pi)$ defined as follows for each $x \in Dom(\Pi)$: if $\Pi(x) = (\pi, i)$, then $succ_{\Gamma}(\Pi)(x) = (\pi, \ell)$ where ℓ is the Γ -successor of i in π . For each $j \in \mathbb{N}$, we use $succ_{\Gamma}^j$ for the function obtained by j applications of the function $succ_{\Gamma}$.

HyperLTL_S formulas are linear-time hyper sentences over multi-trace specifications ψ , called *HyperLTL_S quantifier-free formulas*, where the syntax of ψ is as follows:

$$\psi ::= \top \mid p[x] \mid \neg\psi \mid \psi \wedge \psi \mid X_{\Gamma}\psi \mid \psi U_{\Gamma}\psi$$

where $p \in AP$, $x \in VAR$, Γ is a finite set of LTL formulas over AP , and X_{Γ} and U_{Γ} are the stutter-relativized versions of the LTL temporal modalities. Informally, the relativized temporal modalities X_{Γ} and U_{Γ} are evaluated by a lockstepwise traversal of the Γ -stutter traces associated with the currently quantified traces. Standard HyperLTL corresponds to the fragment of HyperLTL_S where the subscript of each temporal modality is the empty set \emptyset .

Given a HyperLTL_S quantifier-free formula ψ and a pointed trace assignment Π such that $Dom(\Pi)$ contains the trace variables occurring in ψ , the satisfaction relation $\Pi \models \psi$ is inductively defined as follows (we omit the semantics of the Boolean connectives):

$$\begin{aligned} \Pi \models p[x] &\Leftrightarrow \Pi(x) = (\pi, i) \text{ and } p \in \pi(i) \\ \Pi \models X_{\Gamma}\psi &\Leftrightarrow succ_{\Gamma}(\Pi) \models \psi \\ \Pi \models \psi_1 U_{\Gamma}\psi_2 &\Leftrightarrow \text{for some } i \geq 0 : succ_{\Gamma}^i(\Pi) \models \psi_2 \text{ and } succ_{\Gamma}^k(\Pi) \models \psi_1 \text{ for all } 0 \leq k < i \end{aligned}$$

Stuttering LTL formulas, corresponding to one-variable HyperLTL_S quantifier-free formulas, can be translated in polynomial time into equivalent LTL formulas (see [6]). Thus, since LTL satisfiability is PSPACE-complete, the following result holds.

► **Proposition 7.** *The trace properties definable by HyperLTL_S formulas are LTL definable, and single-trace satisfiability of HyperLTL_S is PSPACE-complete.*

HyperLTL_S versus A-HyperLTL. We show that HyperLTL_S and A-HyperLTL are expressively incomparable even over singleton sets of atomic propositions. By Theorem 2, unlike HyperLTL_S, A-HyperLTL does not subsume HyperLTL even when $|AP| = 1$. Hence, HyperLTL_S is not subsumed by A-HyperLTL even when $|AP| = 1$. We show now that the converse holds as well. Intuitively, A-HyperLTL can encode counting mechanisms which cannot be expressed in HyperLTL_S. Let $AP = \{p\}$. We exhibit two families $\{\mathcal{L}_n\}_{n \geq 1}$ and $\{\mathcal{L}'_n\}_{n \geq 1}$ of trace sets and an A-HyperLTL formula φ_A such that

- φ_A can distinguish the traces set \mathcal{L}_n and \mathcal{L}'_n for each $n \geq 1$, but
- for each HyperLTL_S formula ψ , there is n such that ψ does not distinguish \mathcal{L}_n and \mathcal{L}'_n .

For each $n \geq 1$, let π_n , ρ_n , and ρ'_n be the traces defined as:

$$\pi_n \stackrel{\text{def}}{=} (\emptyset \cdot p)^n \cdot \emptyset^\omega \quad \rho_n \stackrel{\text{def}}{=} (\emptyset \cdot p)^{2n} \cdot \emptyset^\omega \quad \rho'_n \stackrel{\text{def}}{=} (\emptyset \cdot p)^{2n+1} \cdot \emptyset^\omega$$

For each $n \geq 1$, define $\mathcal{L}_n \stackrel{\text{def}}{=} \{\pi_n, \rho_n\}$ and $\mathcal{L}'_n \stackrel{\text{def}}{=} \{\pi_n, \rho'_n\}$. Let $\psi_1(x)$ and $\psi_2(x)$ be two one-variable quantifier-free HyperLTL formulas capturing the following requirements:

- $\psi_1(x)$ captures traces of the form $(\emptyset \cdot p)^k \cdot \emptyset^\omega$ for some $k \geq 1$,
- $\psi_2(x)$ captures traces of the form $(\emptyset^2 \cdot p^2)^k \cdot \emptyset^\omega$ for some $k \geq 1$.

Let $\psi(x, y)$ be the two-variable quantifier-free HyperLTL formula defined as follows:

$$\psi(x, y) \stackrel{\text{def}}{=} \mathbf{F}(p[x] \wedge p[y] \wedge \mathbf{XG}(\neg p[x] \wedge \neg p[y]))$$

Intuitively, if x is bound to a trace ν_1 satisfying ψ_1 and y is bound to a trace ν_2 satisfying ψ_2 , then $\psi(x, y)$ holds iff ν_1 is of the form $(\emptyset \cdot p)^{2k}$ and ν_2 is of the form $(\emptyset^2 \cdot p^2)^k$ for some $k \geq 1$. The A-HyperLTL formula φ_A is then defined as follows:

$$\varphi_A \stackrel{\text{def}}{=} \forall x_1. \forall x_2. \mathbf{E}([\psi(x_1, x_2) \wedge \psi_1(x_1) \wedge \psi_1(x_2)] \vee [\psi(x_1, x_2) \wedge \bigvee_{i \in \{1, 2\}} (\psi_1(x_i) \wedge \psi_2(x_{3-i}))])$$

Let π a trace of the form $\pi = (\emptyset \cdot p)^k \cdot \emptyset^\omega$ for some $k \geq 1$. We observe that the unique stuttering expansion ν_1 of π such that ν_1 satisfies $\psi_1(x)$ is π itself. Similarly, there is a unique stuttering expansion ν_2 of π such that ν_2 satisfies $\psi_2(x)$, and such a trace ν_2 is given by $(\emptyset^2 \cdot p^2)^k \cdot \emptyset^\omega$. Fix $n \geq 1$. Let us consider the trace set $\mathcal{L}_n = \{\pi_n, \rho_n\}$. By construction, if both variables x_1 and x_2 in the definition of φ_A are bound to the same trace π in \mathcal{L}_n , then the first disjunct in the definition of φ_A is fulfilled by taking π itself as an expansion of π . On the other hand, assume that variable x_1 (resp., x_2) is bound to trace π_n and variable x_2 (resp., x_1) is bound to trace ρ_n . In this case, by taking as stuttering expansion of π_n the trace $(\emptyset^2 \cdot p^2)^n \cdot \emptyset^\omega$ and as stuttering expansion of $\rho_n = (\emptyset \cdot p)^{2n} \cdot \emptyset^\omega$ the trace ρ_n itself, the second disjunct in the definition of φ_A is fulfilled. Hence, \mathcal{L}_n is a model of φ_A .

Now, we show that $\mathcal{L}'_n = \{\pi_n, \rho'_n\}$ does not satisfy φ_A . Let us consider the mapping assigning to variable x_1 the trace π_n and to variable x_2 the trace ρ'_n . With this mapping, the quantifier-free part of φ_A cannot be fulfilled. This because the unique stuttering expansion of $\pi_n = (\emptyset \cdot p)^n \cdot \emptyset^\omega$ (resp., $\rho'_n = (\emptyset \cdot p)^{2n+1} \cdot \emptyset^\omega$) satisfying ψ_1 is π_n (resp., ρ'_n) itself. Moreover, the unique stuttering expansion of π_n (resp., ρ'_n) satisfying ψ_2 is $(\emptyset^2 \cdot p^2)^n \cdot \emptyset^\omega$ (resp., $(\emptyset^2 \cdot p^2)^{2n+1} \cdot \emptyset^\omega$). Hence, for all $n \geq 1$, $\mathcal{L}_n \models \varphi_A$ and $\mathcal{L}'_n \not\models \varphi_A$. On the other hand, one can show that the following holds (for details, see [7]).

► **Proposition 8.** *For each HyperLTL_S formula ψ , there is $n \geq 1$ s.t. $\mathcal{L}_n \models \psi$ iff $\mathcal{L}'_n \models \psi$.*

Thus, since A-HyperLTL does not subsume HyperLTL_S, we obtain the following result.

► **Corollary 9.** *A-HyperLTL and HyperLTL_S are expressively incomparable.*

3.3 Results for Context HyperLTL (HyperLTL_C)

Context HyperLTL (HyperLTL_C) [6] extends HyperLTL by unary modalities $\langle C \rangle$ parameterized by a non-empty subset C of trace variables – called the *context* – which restrict the evaluation of the temporal modalities to the traces associated with the variables in C . We show that HyperLTL_C is not subsumed by A-HyperLTL or HyperLTL_S, and single-trace satisfiability of HyperLTL_C is undecidable. Moreover, we provide a characterization of the *finite trace properties* denoted by HyperLTL_C formulas in terms of the extension $\text{FO}_f[<, +]$ of standard first-order logic $\text{FO}_f[<]$ over finite words with *addition*. We also establish that the variant of

$\text{FO}_f[<, +]$ over infinite words characterizes the trace properties expressible in the extension of HyperLTL_C with past temporal modalities. Finally, we identify a fragment of HyperLTL_C which subsumes HyperLTL and for which model checking is decidable.

HyperLTL_C formulas are linear-time hyper sentences over multi-trace specifications ψ , called *HyperLTL_C quantifier-free formulas*, where the syntax of ψ is as follows:

$$\psi ::= \top \mid p[x] \mid \neg\psi \mid \psi \wedge \psi \mid \mathbf{X}\psi \mid \psi \mathbf{U}\psi \mid \langle C \rangle \psi$$

where $p \in \text{AP}$, $x \in \text{VAR}$, and $\langle C \rangle$ is the context modality with $\emptyset \neq C \subseteq \text{VAR}$. A context C is *global for a formula* φ if C contains all the trace variables occurring in φ . Let Π be a pointed-trace assignment. Given a context C and an offset $i \geq 0$, we denote by $\Pi +_C i$ the pointed-trace assignment with domain $\text{Dom}(\Pi)$ defined as follows. For each $x \in \text{Dom}(\Pi)$, where $\Pi(x) = (\pi, h)$: $[\Pi +_C i](x) = (\pi, h + i)$ if $x \in C$, and $[\Pi +_C i](x) = \Pi(x)$ otherwise. Intuitively, the positions of the pointed traces associated with the variables in C advance of the offset i , while the positions of the other pointed traces remain unchanged. Let ψ be a HyperLTL_C quantifier-free formula such that $\text{Dom}(\Pi)$ contains the variables occurring in ψ . The satisfaction relation $(\Pi, C) \models \psi$ is defined as follows (we omit the semantics of the Boolean connectives):

$$\begin{aligned} (\Pi, C) \models p[x] &\iff \Pi(x) = (\pi, i) \text{ and } p \in \pi(i) \\ (\Pi, C) \models \mathbf{X}\psi &\iff (\Pi +_C 1, C) \models \psi \\ (\Pi, C) \models \psi_1 \mathbf{U}\psi_2 &\iff \text{for some } i \geq 0 : (\Pi +_C i, C) \models \psi_2 \text{ and } (\Pi +_C k, C) \models \psi_1 \text{ for all } k < i \\ (\Pi, C) \models \langle C' \rangle \psi &\iff (\Pi, C') \models \psi \end{aligned}$$

We write $\Pi \models \psi$ to mean that $(\Pi, \text{VAR}) \models \psi$.

HyperLTL_C versus A-HyperLTL and HyperLTL_S. We show that HyperLTL_C is able to capture powerful non-regular trace properties which cannot be expressed in A-HyperLTL or in HyperLTL_S . In particular, we consider the following trace property over $\text{AP} = \{p\}$:

Suffix Property: a trace π satisfies the suffix property if π has a proper suffix π^i for some $i > 0$ such that $\pi^i = \pi$.

This property can be expressed in HyperLTL_C by the following formula

$$\varphi_{\text{suffix}} \stackrel{\text{def}}{=} \forall x_1. \forall x_2. \bigwedge_{p \in \text{AP}} \mathbf{G}(p[x_1] \leftrightarrow p[x_2]) \wedge \{x_2\} \mathbf{FX}\{x_1, x_2\} \bigwedge_{p \in \text{AP}} \mathbf{G}(p[x_1] \leftrightarrow p[x_2])$$

We show that no A-HyperLTL and no HyperLTL_S formula is equivalent to φ_{suffix} .

► **Theorem 10.** *A-HyperLTL and HyperLTL_S cannot express the suffix property. Hence, HyperLTL_C is not subsumed by A-HyperLTL or by HyperLTL_S.*

Proof sketch. By Proposition 7, the set of single-trace models of a HyperLTL_S formula is regular. Thus, since the suffix trace property is not regular, the result for HyperLTL_S follows.

Consider now A-HyperLTL . For each $n \geq 1$, let $\pi_n \stackrel{\text{def}}{=} (p^n \cdot \emptyset)^\omega$ and $\pi'_n \stackrel{\text{def}}{=} p^{n+1} \cdot \emptyset \cdot (p^n \cdot \emptyset)^\omega$. By construction π_n satisfies the suffix property but π'_n not. Hence, for each $n \geq 1$, the HyperLTL_C formula φ_{suffix} distinguishes the singleton sets $\{\pi_n\}$ and $\{\pi'_n\}$. On the other hand, we can show the following result, hence, Theorem 10 directly follows (a proof of the following claim is given in [7]).

▷ **Claim.** For each A-HyperLTL formula ψ , there is $n \geq 1$ such that $\{\pi_n\} \models \psi$ iff $\{\pi'_n\} \models \psi$. ◀

Single-trace satisfiability and characterization of HyperLTL_C finite-trace properties. Like A-HyperLTL, and unlike HyperLTL and HyperLTL_S, single-trace satisfiability of HyperLTL_C turns out to be undecidable. In particular, by a straightforward adaptation of the undecidability proof in [6] for model checking HyperLTL_C, one can reduce the recurrence problem in Minsky counter machines [23] to single-trace satisfiability of HyperLTL_C.

► **Theorem 11.** *The single-trace satisfiability problem for A-HyperLTL is Σ_1^1 -hard.*

A finite trace (over AP) is a finite non-empty word over 2^{AP} . By adding a fresh proposition $\# \notin \text{AP}$, a finite trace can be encoded by the trace $\text{enc}(w)$ over $\text{AP} \cup \{\#\}$ given by $w \cdot \{\#\}^\omega$. Given a HyperLTL_C formula φ over $\text{AP} \cup \{\#\}$, the *finite-trace property denoted by φ* is the language $\mathcal{L}_f(\varphi)$ of finite traces w over AP such that the single-trace model $\{\text{enc}(w)\}$ satisfies φ . We provide now a characterization of the finite-trace properties denoted by HyperLTL_C formulas over $\text{AP} \cup \{\#\}$ in terms of the extension $\text{FO}_f[<, +]$ of standard first-order logic $\text{FO}_f[<]$ over finite words on 2^{AP} with addition. Formally, $\text{FO}_f[<, +]$ is a first-order logic with equality over the signature $\{<, +\} \cup \{P_a \mid a \in \text{AP}\}$, where the atomic formulas ψ have the following syntax with x, y and z being first-order variables: $\psi \stackrel{\text{def}}{=} x = y \mid x < y \mid z = y + x \mid P_a(x)$. A $\text{FO}_f[<, +]$ sentence (i.e., a $\text{FO}_f[<, +]$ formula with no free variables) is interpreted over finite traces w , where: (i) variables ranges over the set $\{0, \dots, |w| - 1\}$ of positions of w , (ii) the binary predicate $<$ is the natural ordering on $\{0, \dots, |w| - 1\}$, and (iii) the predicates $z = y + x$ and $P_a(x)$ are interpreted in the obvious way. We establish the following result.

► **Theorem 12.** *Given a $\text{FO}_f[<, +]$ sentence φ over AP, one can construct in polynomial time a HyperLTL_C formula ψ over $\text{AP} \cup \{\#\}$ such that $\mathcal{L}_f(\psi)$ is the set of models of φ . Vice versa, given a HyperLTL_C formula ψ over $\text{AP} \cup \{\#\}$, one can construct in single exponential time a $\text{FO}_f[<, +]$ sentence φ whose set of models is $\mathcal{L}_f(\psi)$.*

Intuitively, when a HyperLTL_C formula ψ over $\text{AP} \cup \{\#\}$ is interpreted over singleton models $\{\text{enc}(w)\}$ for a given finite trace w , the trace variables in the quantifier-free part of ψ and the temporal modalities evaluated in different contexts can simulate quantification over positions in w and the atomic formulas of $\text{FO}_f[<, +]$. In particular, the addition predicate $z = x + y$ can be simulated by requiring that two segments of w whose endpoints are referenced by trace variables have the same length: this is done by shifting with the eventually modality in a suitable context the left segment of a non-negative offset, and by checking that the endpoints of the resulting segments coincide. Note that for trace variables x and y which refer to positions i and j of w , one can require that i and j coincide by the HyperLTL_C formula $\{x, y\}F(\neg\#[x] \wedge \neg\#[y] \wedge X(\#[x] \wedge \#[y]))$. Similarly, if we consider the extension of HyperLTL_C with the past counterparts of the temporal modalities, then the trace properties denoted by past HyperLTL_C formulas correspond to the ones denoted by sentences in the variant $\text{FO}[<, +]$ of $\text{FO}_f[<, +]$ over infinite words on 2^{AP} (traces). For arbitrary traces, past temporal modalities are crucial for enforcing that two variables refer to the same position (for details see [7]).

► **Theorem 13.** *Past HyperLTL_C and $\text{FO}[<, +]$ capture the same class of trace properties.*

Results about model checking HyperLTL_C. Model checking HyperLTL_C is known to be undecidable even for formulas where the unique temporal modality occurring in the scope of a non-global context is F. For the fragment where the unique temporal modality occurring in a non-global context is X, then the problem is decidable. This fragment has the same expressiveness as HyperLTL but it is exponentially more succinct than HyperLTL [6]. We

provide now new insights on model checking HyperLTL_C . On the negative side, we show that model checking is undecidable even for the fragment \mathcal{U} consisting of two-variable quantifier alternation-free formulas of the form $\exists x_1. \exists x_2. \psi_0 \wedge \{x_2\}F\{x_1, x_2\}\psi$, where ψ_0 and ψ are quantifier-free HyperLTL formulas. A proof of Theorem 14 appears in [7].

► **Theorem 14.** *Model-checking against the fragment \mathcal{U} of HyperLTL_C is Σ_0^1 -hard.*

By Theorem 14, HyperLTL_C model checking becomes undecidable whenever in a formula a non-singleton context C occurs within a distinct context $C' \neq C$. Thus, we consider the fragment of HyperLTL_C , called *simple HyperLTL_C* , where each context C which occurs in the scope of a distinct context $C' \neq C$ is a singleton. Note that simple HyperLTL_C subsumes HyperLTL .

► **Theorem 15.** *The model checking problem of simple HyperLTL_C is decidable.*

Theorem 15 is proven by a polynomial time translation of simple HyperLTL_C formulas into equivalent sentences of *first-order logic $\text{FO}[\prec, E]$ with the equal-level predicate E* (see [16]) whose model checking problem is known to be decidable [10]. This logic is interpreted over sets \mathcal{L} of traces, and first-order variables refer to pointed traces over \mathcal{L} . In simple HyperLTL_C , the evaluation of temporal modalities is subdivided in two phases. In the first phase, modalities are evaluated by a synchronous traversal of the traces bound to the variables in a non-singleton context. In the second phase, the temporal modalities are evaluated along a single trace and singleton contexts allows to switch from a trace to another one by enforcing a weak form of mutual temporal relation. This behaviour can be encoded in $\text{FO}[\prec, E]$ (for details, see [7]).

3.4 H_μ versus A-HyperLTL, HyperLTL_S , and HyperLTL_C

Both HyperLTL_S and HyperLTL_C are subsumed by H_μ and Hyper AAWA [6]. In particular, quantifier-free formulas of HyperLTL_S and HyperLTL_C can be translated in polynomial time into equivalent Büchi AAWA. Corollary 5 shows that A-HyperLTL is subsumed by H_μ as well. Thus, since A-HyperLTL and HyperLTL_S are expressively incomparable (by Corollary 9), there is an H_μ formula which cannot be expressed in A-HyperLTL (resp., HyperLTL_S). Therefore, we obtain the following corollary.

► **Corollary 16.** *H_μ is strictly more expressive than A-HyperLTL and HyperLTL_S , and subsumes HyperLTL_C .*

4 Asynchronous vs Synchronous Extensions of HyperLTL

We compare now the expressiveness of the asynchronous extensions of HyperLTL against $\text{S1S}[E]$ [10]. $\text{S1S}[E]$ is a monadic second-order logic with equality over the signature $\{\prec, E\} \cup \{P_a \mid a \in \text{AP}\}$ which syntactically extends the monadic second-order logic of one successor S1S with the equal-level binary predicate E . While S1S is interpreted over traces, $\text{S1S}[E]$ is interpreted over sets of traces. A set \mathcal{L} of traces induces the relational structure with domain $\mathcal{L} \times \mathbb{N}$ (i.e., the set of pointed traces associated with \mathcal{L}), where

- the binary predicate \prec is interpreted as the set of pairs of pointed traces in $\mathcal{L} \times \mathbb{N}$ of the form $((\pi, i_1), (\pi, i_2))$ such that $i_1 < i_2$, and
- the equal-level predicate E is interpreted as the set of pairs of pointed traces in $\mathcal{L} \times \mathbb{N}$ of the form $((\pi_1, i), (\pi_2, i))$.

Hence, $<$ allows to compare distinct timestamps along the same trace, while the equal-level predicate allows to compare distinct traces at the same timestamp. For a formal definition of the syntax and semantics of S1S[E] , see [7].

We show that the considered asynchronous extensions of HyperLTL are not subsumed by S1S[E] . Intuitively, for some $k \geq 1$, the logic A-HyperLTL (resp., HyperLTL_S , resp., HyperLTL_C) can express hyperproperties whose set of models having cardinality k (k -models) can be encoded by a non-regular set of traces. On the other hand, we show that the encoding of the k -models of a S1S[E] formula always leads to a regular language.

Let $k \geq 1$. We consider the set of atomic propositions given by $\text{AP} \times [1, k]$ for encoding sets \mathcal{L} of traces (over AP) having cardinality k by traces over $\text{AP} \times [1, k]$.

A trace ν over $\text{AP} \times [1, k]$ is *well-formed* if for all $\ell, \ell' \in [1, k]$ with $\ell \neq \ell'$, there is $i \in \mathbb{N}$ and $p \in \text{AP}$ so that $(p, \ell) \in \nu(i)$ iff $(p, \ell') \notin \nu(i)$. A well-formed trace ν over $\text{AP} \times [1, k]$ encodes the set $\mathcal{L}(\nu)$ of the traces π (over AP) such that there is $\ell \in [1, k]$ where π corresponds to the projection of ν over $\text{AP} \times \{\ell\}$, i.e. for each $i \geq 0$, $\pi(i) = \{p \in \text{AP} \mid (p, \ell) \in \nu(i)\}$. Since ν is well-formed, $|\mathcal{L}(\nu)| = k$ and we say that ν is a k -code of $\mathcal{L}(\nu)$. Note that for a set \mathcal{L} of traces (over AP) of cardinality k , each ordering of the traces in \mathcal{L} induces a distinct k -code.

Given an hyperproperty specification ξ over AP , a k -model of ξ is a set of traces satisfying ξ having cardinality k . The k -language of ξ is the set of k -codes associated with the k -models of ξ . The specification ξ is k -regular if its k -language is a regular language over $\text{AP} \times [1, k]$.

We first show that for each $k \geq 1$, S1S[E] sentences are k -regular.

► **Lemma 17.** *Let $k \geq 1$ and φ be a S1S[E] sentence over AP . Then, one can construct a S1S sentence φ' over $\text{AP} \times [1, k]$ whose set of models is the k -language of φ .*

A proof of Lemma 17 appears in [7]. Since S1S sentences capture only regular languages of traces, by Lemma 17, we obtain the following result.

► **Proposition 18.** *Let $k \geq 1$ and φ be a S1S[E] sentence over AP . Then, φ is k -regular.*

We now show that given one of the considered asynchronous extensions L of HyperLTL, there is $k \geq 1$ and a L formula φ such that φ is *not* k -regular.

► **Proposition 19.** *There is a HyperLTL_C formula over $\{p\}$ which is not 1-regular, and there are A-HyperLTL and HyperLTL_S formulas over $\{p\}$ which are not 2-regular.*

Proof. Let $\text{AP} = \{p\}$. The HyperLTL_C formula φ_{suff} defined in Section 3.3 whose models consist of the singletons $\{\pi\}$ such that π satisfies the suffix property is not 1-regular.

Consider now A-HyperLTL and HyperLTL_S . For all $k, n \geq 1$, let $\pi_{k,n} \stackrel{\text{def}}{=} \emptyset^k \cdot (\{p\} \cdot \emptyset)^n \cdot \emptyset^\omega$ and $\mathcal{L}_{k,n} \stackrel{\text{def}}{=} \{\pi_{1,n}, \pi_{k,n}\}$. We denote by \mathcal{L}_2 the set of traces over $\text{AP} \times [1, 2]$ which are 2-codes of the sets $\mathcal{L}_{k,n}$ for $k > 1$ and $n \geq 1$. Clearly, \mathcal{L}_2 is not regular. Let $\theta(x)$ be a one-variable quantifier-free HyperLTL formula capturing the traces $\pi_{k,n}$ for $k, n \geq 1$. We define a HyperLTL_S formula ψ_S and an A-HyperLTL formula ψ_A whose 2-language is \mathcal{L}_2 :

$$\psi_S \stackrel{\text{def}}{=} \exists x_1. \forall x_2. \text{X}p[x_1] \wedge \theta(x_1) \wedge \theta(x_2) \wedge \text{G}_{\{p\}}(p[x_1] \leftrightarrow p[x_2]);$$

$$\psi_A \stackrel{\text{def}}{=} \exists x_1. \forall x_2. \forall x_3. \text{E}. \text{X}p[x_1] \wedge \theta(x_2) \wedge \theta(x_3) \wedge \text{G}(p[x_2] \leftrightarrow p[x_3]). \quad \blacktriangleleft$$

By Propositions 18 and 19, and since A-HyperLTL, HyperLTL_S , and HyperLTL_C are subsumed by H_μ (Corollary 16), we obtain the following result.

► **Corollary 20.** *A-HyperLTL, HyperLTL_S , HyperLTL_C , and H_μ are not subsumed by S1S[E] .*

5 Conclusions

Two interesting questions are left open. The first concerns the expressiveness of HyperLTL_C versus A-HyperLTL and HyperLTL_S . We have shown that HyperLTL_C is not subsumed by A-HyperLTL or HyperLTL_S . We conjecture that the converse holds too. The intuition is that (unlike HyperLTL_C) A-HyperLTL and HyperLTL_S implicitly allow a restricted form of monadic second-order quantification. In particular, we conjecture that the hyperproperty characterizing the sets consisting of stuttering-equivalent traces, which can be easily expressed both in A-HyperLTL and HyperLTL_S , cannot be captured by HyperLTL_C .

The second question is whether $\text{S1S}[E]$ is subsumed or not by H_μ . It is known that contrary to $\text{S1S}[E]$ and $\text{FO}[\langle, E]$, HyperLTL cannot express requirements which relate at some point an unbounded number of traces [5]. The main reason is that – differently from $\text{S1S}[E]$ and $\text{FO}[\langle, E]$ – quantifiers in HyperLTL only refer to the initial positions of the traces. Since in H_μ the semantics of quantifiers is the same as HyperLTL , we conjecture that the inexpressiveness result for HyperLTL in [5] can be extended to H_μ as well. This would imply together with the results of Corollary 20 that $\text{S1S}[E]$ and H_μ are expressively incomparable and that so are $\text{FO}[\langle, E]$ and H_μ .

Future work also includes an expressive comparison with Hypertrace Logic [1], a logical framework recently introduced which extends $\text{FO}[\langle]$ with quantification over traces to express sequential information flow-properties. Also, we plan to study other extensions of temporal logic for asynchronous hyperproperties, in particular for recursive programs [20] and for multi-agent systems [3].

References

- 1 E. Bartocci, T. Ferrère, T.A. Henzinger, D. Nickovic, and A.O. da Costa. Flavors of Sequential Information Flow. In *Proc. 23rd VMCAI*, volume 13182 of *LNCS*, pages 1–19. Springer, 2022. doi:10.1007/978-3-030-94583-1_1.
- 2 J. Baumeister, N. Coenen, B. Bonakdarpour, B. Finkbeiner, and C. Sánchez. A Temporal Logic for Asynchronous Hyperproperties. In *Proc. 33rd CAV*, LNCS 12759, pages 694–717. Springer, 2021. doi:10.1007/978-3-030-81685-8_33.
- 3 R. Beutner and B. Finkbeiner. A Logic for Hyperproperties in Multi-Agent Systems. *CoRR*, abs/2203.07283, 2022. doi:10.48550/arXiv.2203.07283.
- 4 B. Bonakdarpour, C. Sánchez, and G. Schneider. Monitoring Hyperproperties by Combining Static Analysis and Runtime Verification. In *Proc. 8th ISoLA*, LNCS 11245, pages 8–27. Springer, 2018. doi:10.1007/978-3-030-03421-4_2.
- 5 L. Bozzelli, B. Maubert, and S. Pinchinat. Unifying Hyper and Epistemic Temporal Logics. In *Proc. 18th FoSSaCS*, LNCS 9034, pages 167–182. Springer, 2015. doi:10.1007/978-3-662-46678-0_11.
- 6 L. Bozzelli, A. Peron, and C. Sánchez. Asynchronous Extensions of HyperLTL. In *Proc. 36th LICS*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470583.
- 7 L. Bozzelli, A. Peron, and C. Sánchez. Expressiveness and Decidability of Temporal Logics for Asynchronous Hyperproperties. *CoRR*, abs/2207.02956, 2022. doi:10.48550/arXiv.2207.02956.
- 8 M.R. Clarkson, B. Finkbeiner, M. Koleini, K.K. Micinski, M.N. Rabe, and C. Sánchez. Temporal Logics for Hyperproperties. In *Proc. 3rd POST*, LNCS 8414, pages 265–284. Springer, 2014. doi:10.1007/978-3-642-54792-8_15.
- 9 M.R. Clarkson and F.B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.
- 10 N. Coenen, B. Finkbeiner, C. Hahn, and J. Hofmann. The hierarchy of hyperlogics. In *Proc. 34th LICS*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785713.
- 11 R. Dimitrova, B. Finkbeiner, M. Kovács, M.N. Rabe, and H. Seidl. Model Checking Information Flow in Reactive Systems. In *Proc. 13th VMCAI*, LNCS 7148, pages 169–185. Springer, 2012. doi:10.1007/978-3-642-27940-9_12.

- 12 E.A. Emerson and J.Y. Halpern. “Sometimes” and “Not Never” revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986. doi:10.1145/4904.4999.
- 13 B. Finkbeiner and C. Hahn. Deciding Hyperproperties. In *Proc. 27th CONCUR*, LIPIcs 59, pages 13:1–13:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.13.
- 14 B. Finkbeiner, C. Hahn, P. Lukert, M. Stenger, and L. Tentrup. Synthesis from hyperproperties. *Acta Informatica*, 57(1-2):137–163, 2020. doi:10.1007/s00236-019-00358-2.
- 15 B. Finkbeiner, M.N. Rabe, and C. Sánchez. Algorithms for Model Checking HyperLTL and HyperCTL*. In *Proc. 27th CAV Part I*, LNCS 9206, pages 30–48. Springer, 2015. doi:10.1007/978-3-319-21690-4_3.
- 16 B. Finkbeiner and M. Zimmermann. The first-order logic of hyperproperties. In *Proc. 34th STACS*, LIPIcs 66, pages 30:1–30:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.30.
- 17 M.J. Fischer and R.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.
- 18 J.A. Goguen and J. Meseguer. Security Policies and Security Models. In *IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society, 1982. doi:10.1109/SP.1982.10014.
- 19 J.O. Gutsfeld, A. Meier, C. Ohrem, and J. Virtema. Temporal Team Semantics Revisited. *CoRR*, abs/2110.12699, 2021. doi:10.48550/arXiv.2110.12699.
- 20 J.O. Gutsfeld, M. Müller-Olm, and C. Ohrem. Deciding Asynchronous Hyperproperties for Recursive Programs. *CoRR*, abs/2201.12859, 2022. doi:10.48550/arXiv.2201.12859.
- 21 J. Oliver. Gutsfeld, M. Müller-Olm, and C. Ohrem. Propositional dynamic logic for hyperproperties. In *Proc. 31st CONCUR*, LIPIcs 171, pages 50:1–50:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.50.
- 22 J. Oliver. Gutsfeld, M. Müller-Olm, and C. Ohrem. Automata and fixpoints for asynchronous hyperproperties. *Proc. ACM Program. Lang.*, 4(POPL), 2021. doi:10.1145/3434319.
- 23 D. Harel. Effective transformations on infinite trees, with applications to high undecidability, dominoes, and fairness. *J. ACM*, 33(1):224–248, 1986. doi:10.1145/4904.4993.
- 24 A. Krebs, A. Meier, J. Virtema, and M. Zimmermann. Team Semantics for the Specification and Verification of Hyperproperties. In *Proc. 43rd MFCS*, LIPIcs 117, pages 10:1–10:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.MFCS.2018.10.
- 25 M. Lück. On the complexity of linear temporal logic with team semantics. *Theor. Comput. Sci.*, 837:1–25, 2020. doi:10.1016/j.tcs.2020.04.019.
- 26 J. McLean. A General Theory of Composition for a Class of “Possibilistic” Properties. *IEEE Trans. Software Eng.*, 22(1):53–67, 1996. doi:10.1109/32.481534.
- 27 A. Pnueli. The Temporal Logic of Programs. In *Proc. 18th FOCS*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.
- 28 M.N. Rabe. *A temporal logic approach to information-flow control*. PhD thesis, Saarland University, 2016.
- 29 A.P. Sistla, M.Y. Vardi, and P. Wolper. The Complementation Problem for Büchi Automata with Applications to Temporal Logic. *Theoretical Computer Science*, 49:217–237, 1987. doi:10.1016/0304-3975(87)90008-9.
- 30 M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, 1994. doi:10.1006/inco.1994.1092.
- 31 J. Virtema, J. Hofmann, B. Finkbeiner, J. Kontinen, and F. Yang. Linear-Time Temporal Logic with Team Semantics: Expressivity and Complexity. In *Proc. 41st IARCS FSTTCS*, LIPIcs 213, pages 52:1–52:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.FSTTCS.2021.52.
- 32 S. Zdancewic and A.C. Myers. Observational Determinism for Concurrent Program Security. In *Proc. 16th IEEE CSFW-16*, pages 29–43. IEEE Computer Society, 2003. doi:10.1109/CSFW.2003.1212703.

Propositional Dynamic Logic and Asynchronous Cascade Decompositions for Regular Trace Languages

Bharat Adsul  

IIT Bombay, Mumbai, India

Paul Gastin  

Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, 91190, Gif-sur-Yvette, France
CNRS, ReLaX, IRL 2000, Siruseri, India

Saptarshi Sarkar  

IIT Bombay, Mumbai, India

Pascal Weil  

CNRS, ReLaX, IRL 2000, Siruseri, India
Univ. Bordeaux, LaBRI, CNRS UMR 5800, F-33400 Talence, France

Abstract

One of the main motivations for this work is to obtain a distributed Krohn-Rhodes theorem for Mazurkiewicz traces. Concretely, we focus on the recently introduced operation of local cascade product of asynchronous automata and ask if every regular trace language can be accepted by a local cascade product of “simple” asynchronous automata.

Our approach crucially relies on the development of a *local* and *past-oriented* propositional dynamic logic (LocPastPDL) over traces which is shown to be expressively complete with respect to all regular trace languages. An event-formula of LocPastPDL allows to reason about the causal past of an event and a path-formula of LocPastPDL, localized at a process, allows to march along the sequence of past-events in which that process participates, checking for local regular patterns interspersed with local tests of other event-formulas. We also use additional constant formulas to compare the leading process events from the causal past. The new logic LocPastPDL is of independent interest, and the proof of its expressive completeness is rather subtle.

Finally, we provide a translation of LocPastPDL formulas into local cascade products. More precisely, we show that every LocPastPDL formula can be computed by a restricted local cascade product of the gossip automaton and localized 2-state asynchronous reset automata and localized asynchronous permutation automata.

2012 ACM Subject Classification Theory of computation → Concurrency; Theory of computation → Modal and temporal logics; Theory of computation → Algebraic language theory

Keywords and phrases Mazurkiewicz traces, propositional dynamic logic, regular trace languages, asynchronous automata, cascade product, Krohn Rhodes theorem

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.28

1 Introduction

Mazurkiewicz traces form a well-established model of concurrency [11], allowing in particular to describe distributed processes synchronising via shared actions. The concept of recognizability captures important properties of Mazurkiewicz trace languages and it is natural to ask whether recognizable (or regular) trace languages can always be decomposed as a product of simpler languages – in an appropriate formalism.

This is done in the case of regular languages of finite words by the Krohn-Rhodes theorem [16], which states that every regular language is recognized by a cascade product of simple automata, namely reset and permutation automata. The Krohn-Rhodes theorem has



© Bharat Adsul, Paul Gastin, Saptarshi Sarkar, and Pascal Weil;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 28; pp. 28:1–28:19

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

wide-ranging applications, in particular it offers a path to the proof of certain properties of regular languages or of logical fragments by induction on the length of the cascade product [20]. One of the two main results of this paper is a distributed version of this theorem.

Earlier work by the authors [1, 2] established a similar result, but only for first-order definable trace languages. The main ingredient of that proof was the study of a fragment of local temporal logic on traces. As temporal logic specifies only first-order definable languages, it could not be used to cover all regular trace languages and new ideas had to be introduced: we define here a local and past-oriented propositional dynamic logic (**LocPastPDL**) over traces and our second main result is that this logic is expressively complete with respect to all regular trace languages, which is of independent interest.

Before we say more about our results, let us point out the general philosophy of our work: the remarkable development of the theory of regular languages of finite words has used a triple approach: automata-theoretic, logical and algebraic. An example of this approach is the characterization of star-free languages by counter-free automata, by the aperiodicity of their syntactic monoid, by first-order (**FO**) definability, or by definability in linear temporal logic. See [10] for a survey on first-order definable word languages.

Our work is situated in an effort to apply the same philosophy to the study of regular trace languages. Many results of this sort already exist. In particular, regular trace languages are characterized by Zielonka's asynchronous automata [24] (see Section 5), and by **MSO** (monadic second-order) definability (Thomas [22]). Star-freeness is equivalent to **FO**-definability (Ebinger, Muscholl [12]) and to definability in several global or local temporal logics (Thiagarajan, Walukiewicz [21], Diekert, Gastin [8, 9]). Star-free trace languages are also characterized by the aperiodicity of their syntactic monoids (Guaiana, Restivo, Salemi [15]), and Kufleitner [17] gave algebraic and combinatorial characterizations of certain fragments of local linear temporal logic. A discussion of these developments, and of why there are not more algebraic characterizations of significant classes of regular trace languages can be found in [1, 2].

In this paper, traces are viewed as implemented over a distributed architecture: each action (each letter of the alphabet) is *located* over a non-empty subset of *processes* from a finite set \mathcal{P} , and this location determines which pairs of letters are independent. This view of traces is what informs the definition of asynchronous automata [24]. As in [19, 1, 2], we use asynchronous automata not just as acceptors, using accepting states, but also as machines locally computing relabeling functions for input traces (similar in spirit to the sequential letter-to-letter transducers on words). The composition of these relabeling functions is captured by our notion of a *local cascade product*.

The precise statement of our Krohn-Rhodes theorem for trace languages uses also the notion of a *restricted* local cascade product with the *gossip automaton*. The latter is an asynchronous automaton introduced by Mukund and Sohoni [19], which is entirely determined by the distributed architecture under consideration. The information contributed by the gossip automaton in a *restricted* local cascade product is limited to the event-level and the trace-level comparisons of the order that may exist within the trace of the *latest views* of the different processes.

As indicated above, (local) temporal logic is not suitable to discuss trace languages that are not **FO**-definable and we turn to propositional dynamic logic (**PDL**). This logic was introduced by Fischer and Ladner [13] as a way to reason about programs. Over finite words, an appropriate version called **LDL** was shown to be expressively complete with respect to **MSO** by Giacomo and Vardi [7], see also [23]. **PDL** has been applied in different forms to various structures, e.g., Kripke structures [14], message-sequence charts (**MSC**) and message-passing

systems [6, 18]. Recently, Bollig, Fortin and Gastin [4] introduced a star-free version of PDL interpreted on MSC, and showed its expressively completeness with respect to first-order logic. There are however not many results using PDL on traces (see [5, Chapter 5]).

More specifically we introduce a past-oriented fragment of PDL, called **PastPDL**, and a local fragment of it, called **LocPastPDL**. An event-formula of **LocPastPDL** allows to reason about the causal past of an event and a path-formula of **LocPastPDL**, localized at a process, allows to march along the sequence of past-events in which that process participates, checking for local regular patterns interspersed with local tests of other event-formulas. We also use additional constant formulas to compare the leading events for each process, in the strict causal past of a given event. The proof of the expressive completeness of **PastPDL** and **LocPastPDL** is rather subtle and the result is of independent interest.

The paper is organized as follows. Section 2 lays down the basic notion and terminology for traces over distributed architecture and the PDL fragments **PastPDL** and **LocPastPDL** are introduced in Section 3.

The precise statement on the expressive completeness of **PastPDL** and **LocPastPDL**, Theorem 3, is proved in Section 4. The proof is by induction on the number of processes in the distributed architecture. The case of a single process corresponds to the expressive completeness of linear dynamic logic [7]. Generalizing this to several processes is highly non-trivial. It crucially depends on a lifting lemma which constructs a formula $\text{lift}_P(\varphi)$ from a formula φ in **PastPDL** so that φ holds on a suffix s of a (prime) trace $t = rs$ if and only if $\text{lift}_P(\varphi)$ holds on t – where the suffix s is determined by a subset P of processes.

In Section 5, we briefly describe asynchronous automata, and the important notion of *asynchronous labeling functions* computed by such automata, introduced in [1, 2]. As mentioned above, the latter notion generalizes sequential transducers, and is very close to the locally computable functions of Mukund and Sohoni [19]. In the same section, we explain how the composition of asynchronous labeling functions corresponds to the local cascade product operation on asynchronous automata, and we define the notion of the restricted local cascade products of the gossip automaton and an arbitrary asynchronous automaton.

Our main decomposition theorem, Theorem 16, is proved in Section 6. It shows how any **LocPastPDL** event formula is *computed* by a restricted local cascade product of a copy of the gossip automaton, followed by a cascade product of localized reset and permutation automata. A Krohn-Rhodes-like statement, Corollary 17, follows immediately.

2 Mazurkiewicz traces

We consider (Mazurkiewicz) traces as implemented over a distributed architecture. More precisely, we fix a finite set \mathcal{P} of processes. A *distributed alphabet* over \mathcal{P} is a pair (Σ, loc) where the *location function* $\text{loc}: \Sigma \rightarrow 2^{\mathcal{P}} \setminus \{\emptyset\}$ assigns to each letter $a \in \Sigma$ the set of processes which *participate* in a . For $i \in \mathcal{P}$, we let $\Sigma_i = \{a \in \Sigma \mid i \in \text{loc}(a)\}$. The location function induces an *independence relation* over Σ : letters a and b are *independent* if $\text{loc}(a) \cap \text{loc}(b) = \emptyset$, and they are *dependent* otherwise.

When dealing with posets, and in particular with traces, we use the following notation. If (E, \leq) is a poset and $e \in E$, we let $\downarrow e$ (the *past* of e) be the set $\{f \in E \mid f \leq e\}$, and we let $\downarrow\!-\!e = \downarrow e \setminus \{e\}$ (the *strict past* of e). If $X \subseteq E$, we let $\downarrow X = \bigcup_{e \in X} \downarrow e$.

A *trace* over (Σ, loc) is a triple $t = (E, \leq, \lambda)$ where (E, \leq) is a finite poset and $\lambda: E \rightarrow \Sigma$ is a labeling function, such that

- if $e, e' \in E$ and e' is an *immediate successor* of e (that is, $e < e'$ and $e \leq e'' \leq e'$ implies $e'' = e$ or $e'' = e'$), then $\lambda(e)$ and $\lambda(e')$ are dependent;
- if $e, e' \in E$ and $\lambda(e)$ and $\lambda(e')$ are dependent, then $e \leq e'$ or $e' \leq e$.

The elements of E are traditionally called *events*. Further, if $i \in \mathcal{P}$, E_i denotes the set of *i-events* (i.e., events in which process i participates), namely $E_i = \{e \in E \mid i \in \text{loc}(\lambda(e))\}$. It is clear that E_i is totally ordered by \leq .

We let $\text{Tr}(\Sigma, \text{loc})$ denote the set of all traces over (Σ, loc) . We write simply $\text{Tr}(\Sigma)$ if loc is clear from the context. The empty trace (where $E = \emptyset$) is written ε .

$\text{Tr}(\Sigma)$ is a monoid for the following *concatenation* operation on traces. Let $t = (E, \leq, \lambda)$ and $t' = (E', \leq', \lambda')$ be elements of $\text{Tr}(\Sigma)$. Without loss of generality, we can assume E and E' to be disjoint. We define tt' to be the trace $(E \cup E', \leq'', \lambda'')$ where

- \leq'' is the transitive closure of $\leq \cup \leq' \cup \{(e, e') \in E \times E' \mid \lambda(e) \text{ and } \lambda'(e') \text{ are dependent}\}$,
- $\lambda'': E'' \rightarrow \Sigma$ where $\lambda''(e) = \lambda(e)$ if $e \in E$; otherwise, $\lambda''(e) = \lambda'(e)$.

This operation is associative, with the empty trace ε as unit. Hence, $\text{Tr}(\Sigma)$ is a monoid.

A trace t' is said to be a *prefix* (resp. *suffix*) of a trace t if there exists t'' such that $t = t't''$ (resp. $t = t''t'$). Prefixes of t coincide with restrictions of t to downward-closed subsets of events. Prefixes of the form $\downarrow e$ or $\Downarrow e$ ($e \in E$) are important examples.

A *trace language* over Σ is a subset of $\text{Tr}(\Sigma)$. Regular trace languages are characterized by different classical mechanisms: MSO logic, saturation of regular languages of words, asynchronous automata. In this paper, we say that a trace language L is regular if it is *recognized* by a morphism $\eta: \text{Tr}(\Sigma) \rightarrow M$ to a finite monoid: that is, if $L = \eta^{-1}(\eta(L))$.

3 A propositional dynamic logic for traces

Past propositional dynamic logic. Inspired by the definition of PDL [13] and its version meant to be interpreted on finite words (LDL [7]), we introduce **PastPDL**, *past propositional dynamic logic*, to reason about Mazurkiewicz traces. The syntax of PastPDL is the following.

$$\begin{aligned} \Phi &::= \text{EM } \varphi \mid \mathbf{L}_i \leq \mathbf{L}_j \mid \mathbf{L}_{i,j} \leq \mathbf{L}_k \mid \Phi \vee \Phi \mid \neg \Phi \\ \varphi &::= a \mid \mathbf{Y}_i \leq \mathbf{Y}_j \mid \mathbf{Y}_{i,j} \leq \mathbf{Y}_k \mid \varphi \vee \varphi \mid \neg \varphi \mid \langle \pi \rangle \\ \pi &::= \leftarrow_i \mid \varphi? \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^* \end{aligned}$$

Calling this logic *past* is justified by the fact that we allow only backward \leftarrow_i edges and past-oriented constant formulas $\mathbf{L}_i \leq \mathbf{L}_j$, $\mathbf{L}_{i,j} \leq \mathbf{L}_k$, $\mathbf{Y}_i \leq \mathbf{Y}_j$, $\mathbf{Y}_{i,j} \leq \mathbf{Y}_k$, (semantics below).

Formulas of the form Φ , φ and π are called, respectively, *trace formulas* or *sentences*, *event formulas* and *path formulas*. A trace formula is evaluated on a trace (and hence it defines a trace language), an event formula is evaluated at an event of a trace, and a path formula is evaluated at a pair of events. The semantics of PastPDL is as follows. Let $t = (E, \leq, \lambda)$ be a trace. For each process $i \in \mathcal{P}$, let E_i denote the set of *i-events* of t . We let

$$\begin{aligned} t \models \text{EM } \varphi & \quad \text{if } t, e \models \varphi \text{ for some maximal event } e \text{ in } t, \\ t \models \mathbf{L}_i \leq \mathbf{L}_j & \quad \text{if } E_i \neq \emptyset, E_j \neq \emptyset \text{ and } \max(E_i) \leq \max(E_j), \\ t \models \mathbf{L}_{i,j} \leq \mathbf{L}_k & \quad \text{if } E_i \neq \emptyset, E_j \cap \downarrow E_i \neq \emptyset, E_k \neq \emptyset \text{ and } \max(E_j \cap \downarrow E_i) \leq \max(E_k). \end{aligned}$$

In other words, a trace satisfies $\mathbf{L}_i \leq \mathbf{L}_j$ if the last event on process i is below the last event on process j , and it satisfies $\mathbf{L}_{i,j} \leq \mathbf{L}_k$ when the maximal event on process j which is below some event on process i is below the last event on process k .

Note also that $t \models \text{EM } \varphi$ implies in particular that the trace t is nonempty, so the sentence $\neg \text{EM } \top$ defines the empty trace. Also, $\mathbf{L}_i \leq \mathbf{L}_i$ simply means that $E_i \neq \emptyset$, i.e., the trace contains some *i-event*.

We now turn to event and path formulas. Recall that if $t = (E, \leq, \lambda)$ is a trace and $e \in E$ is an event, then $\Downarrow e$ denotes the strict past of e in t . For a process $i \in \mathcal{P}$, we denote by e_i the unique maximal event of $\Downarrow e \cap E_i$, if it exists, i.e., if $\Downarrow e \cap E_i \neq \emptyset$. If $j \in \mathcal{P}$, we write $e_{i,j}$ for $(e_i)_j$, that is, for the maximal event of $\Downarrow e_i \cap E_j$ if e_i exists and $\Downarrow e_i \cap E_j \neq \emptyset$. If e, f are events of t , we let

$t, e \models a$	if $\lambda(e) = a$
$t, e \models Y_i \leq Y_j$	if e_i, e_j exist and $e_i \leq e_j$
$t, e \models Y_{i,j} \leq Y_k$	if $e_{i,j}, e_k$ exist and $e_{i,j} \leq e_k$
$t, e \models \langle \pi \rangle$	if there exists an event $f \in E$ such that $t, e, f \models \pi$
$t, e, f \models \leftarrow_i$	if f is the immediate predecessor of e on process i
$t, e, f \models \varphi?$	if $e = f$ and $t, e \models \varphi$
$t, e, f \models \pi_1 + \pi_2$	if $t, e, f \models \pi_1$ or $t, e, f \models \pi_2$
$t, e, f \models \pi_1 \cdot \pi_2$	if there is an event g , such that $t, e, g \models \pi_1$ and $t, g, f \models \pi_2$
$t, e, f \models \pi^*$	if there are events $e = e_0, e_1, \dots, e_n = f$ with $n \geq 0$ and $t, e_i, e_{i+1} \models \pi$ for all $0 \leq i < n$

We sometime use $\langle \pi \rangle \varphi$ as a macro for $\langle \pi \cdot \varphi? \rangle$. For instance, we may easily express a *strict since* modality restricted to events on a specified process. More precisely, if $i \in \mathcal{P}$ is a process and φ, ψ are event formulas, then the event formula $\langle (\leftarrow_i \cdot \varphi?)^* \cdot \leftarrow_i \psi? \rangle$, denoted $\varphi \mathcal{S}_i \psi$, holds at some event e of a trace t when there is a sequence $f_n, f_{n-1}, \dots, f_1, f_0 = e$ of *consecutive i -events* ($n > 0$) with $t, f_n \models \psi$ and $t, f_j \models \varphi$ for all $0 < j < n$.

PastPDL is no more expressive than MSO logic.

► **Proposition 1.** *For all PastPDL sentences Φ , event formulas φ and path formulas π , we can construct MSO sentences $\bar{\Phi}$, and formulas $\bar{\varphi}(x), \bar{\pi}(x, y)$ with respectively one or two free first-order variables such that, for all traces t and events e, f in t , we have*

$t \models \Phi$	if and only if	$t \models \bar{\Phi}$
$t, e \models \varphi$	if and only if	$t, x \mapsto e \models \bar{\varphi}(x)$
$t, e, f \models \pi$	if and only if	$t, x \mapsto e, y \mapsto f \models \bar{\pi}(x, y)$

The proof technique is folklore and is an easy structural induction. For the base case(s), we note that the constant formulas $L_i \leq L_j$, $L_{i,j} \leq L_k$, $Y_i \leq Y_j$ and $Y_{i,j} \leq Y_k$ all have first-order definitions. For the Kleene star π^* , the induction step relies on the well-known fact that transitive closure of an MSO-definable relation can be expressed in MSO.

It will be convenient in the sequel to use automata instead of regular expressions to specify path formulas. Define a *path automaton* to be a tuple of the form $\mathcal{A} = (Q, \Delta, I, F)$, where Q is a finite, non-empty set of states, $I, F \subseteq Q$ are, respectively, the sets of initial and final states, and Δ is a finite set of transitions of the form (q, α, q') with $q, q' \in Q$ and

- either $\alpha = \varphi?$ for some event formula φ (a *test* transition),
- or $\alpha \in \{\leftarrow_i \mid i \in \mathcal{P}\}$ (a *move* transition).

A path automaton specifies a path formula with the expected semantics: for a trace $t = (E, \leq, \lambda)$ and two events $e, f \in E$, we have $t, e, f \models \mathcal{A}$ if there exists an accepting run $q_0 \xrightarrow{\alpha_1} q_1 \cdots q_{n-1} \xrightarrow{\alpha_n} q_n$ and a sequence of events $e = e_0, e_1, \dots, e_{n-1}, e_n = f$ such that for all $0 \leq m < n$ we have $t, e_m, e_{m+1} \models \alpha_m$. Notice that if $n = 0$ the condition is simply $e = f$.

If \mathcal{A} is a path automaton, then $\langle \mathcal{A} \rangle$ is an event formula where $t, e \models \langle \mathcal{A} \rangle$ if there exists an event $f \in E$ such that $t, e, f \models \mathcal{A}$.

► **Proposition 2.** *Path formulas and path automata are equally expressive:*

1. For each path formula π , we can construct a path automaton \mathcal{A}_π such that for all traces t and events e, f we have $t, e, f \models \pi$ if and only if $t, e, f \models \mathcal{A}_\pi$.
2. For each path automaton \mathcal{A} , we can construct a path formula $\pi_{\mathcal{A}}$ such that for all traces t and events e, f we have $t, e, f \models \mathcal{A}$ if and only if $t, e, f \models \pi_{\mathcal{A}}$.

This is a direct consequence of the equivalence between regular expressions and finite state automata. Indeed, a path formula π can be seen as a regular expression over the alphabet Γ_π consisting of the moves \leftarrow_i ($i \in \mathcal{P}$) and the tests $\varphi?$ which occur at the top level of π . If \mathcal{A}_π is an automaton accepting the language of Γ_π specified by π , then \mathcal{A}_π is a path automaton which is equivalent to the path formula π . The converse is similarly justified.

Local past propositional dynamic logic. Proposition 2 shows that we may replace $\langle \pi \rangle$ with $\langle \mathcal{A} \rangle$ in the syntax of PastPDL event formulas without changing the expressivity of the logic. Adopting the syntax using path automata allows us to state the definition of LocPastPDL, the *local* fragment of PastPDL. More precisely, say that a path automaton \mathcal{A} is *i-local* for some process $i \in \mathcal{P}$, if all its move transitions are labeled with \leftarrow_i . The automaton \mathcal{A} is *local* if it is *i-local* for some $i \in \mathcal{P}$. The syntax of LocPastPDL is as follows:

$$\begin{aligned} \Phi &::= \text{EM } \varphi \mid \text{L}_i \leq \text{L}_j \mid \text{L}_{i,j} \leq \text{L}_k \mid \Phi \vee \Phi \mid \neg \Phi \\ \varphi &::= a \mid \text{Y}_i \leq \text{Y}_j \mid \text{Y}_{i,j} \leq \text{Y}_k \mid \varphi \vee \varphi \mid \neg \varphi \mid \langle \mathcal{A} \rangle. \end{aligned}$$

where $i, j, k \in \mathcal{P}$, $a \in \Sigma$ and \mathcal{A} ranges over local path automata.

The semantics of LocPastPDL is inherited from PastPDL. We show in Section 4 that both logics are expressively complete with respect to regular trace languages.

4 Expressivity

The main result in this section is the following.

► **Theorem 3.** *PastPDL and LocPastPDL are expressively complete, that is: a trace language is regular if and only if it can be defined by a PastPDL (resp. LocPastPDL) sentence.*

One direction of Theorem 3 is easily taken care of: we saw in Proposition 1 that PastPDL sentences define regular trace languages. Conversely, let L be a regular language, and let η be a morphism from $\text{Tr}(\Sigma)$ to a finite monoid M , recognizing L . Since sentences of LocPastPDL are closed under disjunction, it is enough to show that every trace language of the form $\eta^{-1}(m)$ ($m \in M$) is LocPastPDL-definable.

This is established in two steps. We first deal with prime traces. Recall that a trace $t = (E, \leq, \lambda)$ is *prime* if E has a single maximal event, which we then denote by $\max(t)$. In Theorem 4, we show how to construct a LocPastPDL event formula $\varphi^{(m)}$ such that, if t is a prime trace, then $\eta(t) = m$ if and only if $t, \max(t) \models \varphi^{(m)}$.

Leveraging this partial result to handle all traces – and not just prime traces –, is done in Theorem 7.

4.1 Expressivity of event formulas in LocPastPDL

As announced, we first show that event formulas in LocPastPDL are expressive enough to describe regular sets of prime traces.

► **Theorem 4.** *Let $\eta: \text{Tr}(\Sigma) \rightarrow M$ be a morphism to a finite monoid. For each $m \in M$, we can construct a LocPastPDL event formula $\varphi^{(m)}$ such that, if $t \in \text{Tr}(\Sigma)$ is a prime trace, then $\eta(t) = m$ if and only if $t, \max(t) \models \varphi^{(m)}$.*

The proof of Theorem 4, to be completed at the end of Section 4.1, is by induction on the number of processes. We start with a high-level description of this proof. Let t be a prime trace, let $e = \max(t)$ and let k be a process such that $e \in E_k$. Let $f_1 < f_2 < \dots < f_\ell = e$ be the sequence of events on process k . Then t is equal to the product $t_1 t_2 \dots t_\ell$ with $t_i = \downarrow f_i \setminus \downarrow f_{i-1}$ ($\downarrow f_0 = \emptyset$). Note that t_i is prime, with $\max(t_i) = f_i$, and that t_i has no event on process k apart from f_i : this property of t_i with respect to k opens the door to the usage of the induction hypothesis.

More precisely, we use induction to construct for each $m \in M$ an event formula $\varphi^{(m)}$ such that, for all prime traces $s = \downarrow g$ such that g is the only event on process k (and each t_i is of this form), we have $\eta(s) = m$ if and only if $s, g \models \varphi^{(m)}$.

The next task is to “lift” the formula $\varphi^{(m)}$, meant to be interpreted on the factors t_i , to a formula $\text{lift}_k(\varphi^{(m)})$ to be interpreted on the full trace t . This is done in Lemma 6, in such a way that $t_i, f_i \models \varphi^{(m)}$ if and only if $t, f_i \models \text{lift}_k(\varphi^{(m)})$. The difference is subtle: in one case, past modalities are evaluated on a scope contained in t_i , whereas in the other, their scope may span the full past of f_i in t , i.e., $t_1 \dots t_i$. The lifted formula has to ensure that one never goes below f_{i-1} .

The particular properties of the t_i which make this possible are abstracted out, and generalized, by what we call *residues*. Somewhat informally, if P is a set of processes and g is an event, we let $\text{res}_P(g)$ (the residue of the event g with respect to P) be the largest suffix of $\downarrow g$ which does not contain any event on the processes in P except, perhaps, g itself. Notice that $t_i = \text{res}_{\{k\}}(f_i)$. Lemma 6 proves, by structural induction, that event formulas in PastPDL can be lifted with respect to residues.

Lemma 5, which plays a crucial role in the inductive proof of Lemma 6, shows how the set P of processes may increase to some set P' when one moves from an event g to some previous event g' . The determination of this set P' is possible thanks to the event formulas $Y_i \leq Y_j$ and $Y_{i,j} \leq Y_k$ (primary and secondary comparisons).

The last step of the proof of Theorem 4 uses a k -local path automaton to visit the sequence of events $f_1 < f_2 < \dots < f_\ell = e$ backward, checking along the path the values of the $\eta(t_i)$ with event formulas of the form $\text{lift}_k(\varphi^{(m_i)})$ and storing in its state the value of the product $\eta(t_i) \dots \eta(t_\ell)$: when the automaton has reached f_1 , the first event on process k , it has computed $\eta(t_1)\eta(t_2) \dots \eta(t_\ell) = \eta(t)$.

The precise definition of *residuation* is as follows. For an event e of a trace $t = (E, \leq, \lambda)$ and a process $i \in \mathcal{P}$, recall that $e_i = \max(\downarrow e \cap E_i)$, if it exists, where E_i is the set of i -events in E . By convention, we let $\downarrow e_i = \emptyset$ when e_i does not exist, i.e., when $\downarrow e \cap E_i = \emptyset$.

If $P \subseteq \mathcal{P}$ is a set of processes, the *residue of e by P* is the trace $\text{res}_P(e) = \downarrow e \setminus \bigcup_{i \in P} \downarrow e_i$. In particular, $\text{res}_P(e)$ is a suffix of the trace $\downarrow e$, itself a prefix of t . We will use the following technical result, which makes essential use of the primary and secondary comparison formulas $Y_i \leq Y_j$ and $Y_{i,j} \leq Y_k$.

► **Lemma 5.** *Let t be a trace, $i \in \mathcal{P}$ a process and $P \subseteq \mathcal{P}$ a set of processes. Let e be an event in t such that e_i exists. Then we have*

$$\downarrow e_i \cap \text{res}_P(e) = \begin{cases} \varepsilon & \text{if } t, e \models \bigvee_{k \in P} Y_i \leq Y_k, \\ \text{res}_{P'}(e_i) & \text{otherwise,} \end{cases}$$

where $P' = P \cup \{j \in \mathcal{P} \mid t, e \models Y_{i,j} \leq Y_k \text{ for some } k \in P\}$.

Proof. By definition, we have $\downarrow e_i \cap \text{res}_P(e) = \downarrow e_i \setminus \bigcup_{k \in P} \downarrow e_k$. This is the empty trace if and only if e_i is in one of the $\downarrow e_k$ ($k \in P$), that is, if and only if $t, e \models \bigvee_{k \in P} Y_i \leq Y_k$.

Let us now assume that $t, e \not\models \bigvee_{k \in P} Y_i \leq Y_k$. Note that $\text{res}_{P'}(e_i) = \downarrow e_i \setminus \bigcup_{k \in P'} \downarrow e_{i,k}$. Let $f \in \bigcup_{k \in P'} \downarrow e_{i,k}$. We have $f \leq e_{i,k}$ for some $k \in P'$. If $k \in P$, then $e_{i,k} \leq e_k$. If $k \notin P$, then $e_{i,k} \leq e_j$ for some $j \in P$. In both cases we have $f \in \bigcup_{k \in P} \downarrow e_k$. Therefore $\bigcup_{k \in P'} \downarrow e_{i,k} \subseteq \bigcup_{k \in P} \downarrow e_k$ and hence

$$\downarrow e_i \cap \text{res}_P(e) = \downarrow e_i \setminus \bigcup_{k \in P} \downarrow e_k \subseteq \downarrow e_i \setminus \bigcup_{k \in P'} \downarrow e_{i,k} = \text{res}_{P'}(e_i).$$

Conversely, let $f \in \text{res}_{P'}(e_i)$. In particular, $f \in \downarrow e_i$. Assume that $f \leq e_k$ for some $k \in P$. Since $t, e \not\models Y_i \leq Y_k$, we know that $e_i \not\leq e_k$. If $e_k < e_i$, then $e_{i,k} = e_k$ and we get $f \in \bigcup_{j \in P'} \downarrow e_{i,j}$, a contradiction.

It follows that the events e_i and e_k are concurrent: $e_i \parallel e_k$. Let g be a maximal event in $\uparrow f \cap \downarrow e_i \cap \downarrow e_k$ (this set is not empty since it contains f). Then there exists $\ell \in \text{loc}(g)$ such that $g = e_{i,\ell}$. This implies that $\ell \in P'$ and again $f \in \bigcup_{j \in P'} \downarrow e_{i,j}$, a contradiction.

This concludes the proof that $\text{res}_{P'}(e_i)$ is contained in $\downarrow e_i \setminus \bigcup_{k \in P} \downarrow e_k = \downarrow e_i \cap \text{res}_P(e)$. ◀

We now establish the technical core of the proof of Theorem 4, namely the following *lifting lemma*, which turns an event formula satisfied by a residue of a trace t , into another satisfied by the trace t itself.

► **Lemma 6** (Lifting lemma). *Let $\varphi \in \text{PastPDL}$ be an event formula and $P \subseteq \mathcal{P}$ be a set of processes. We can construct an event formula $\text{lift}_P(\varphi) \in \text{PastPDL}$ such that, for all traces $t = (E, \leq, \lambda)$ and events $e \in E$ in t , we have $\text{res}_P(e), e \models \varphi$ if and only if $t, e \models \text{lift}_P(\varphi)$. Moreover, if $\varphi \in \text{LocPastPDL}$ then $\text{lift}_P(\varphi) \in \text{LocPastPDL}$.*

Proof. The construction is by structural induction on φ . We first let

$$\begin{aligned} \text{lift}_P(a) &= a \text{ for each } a \in \Sigma \\ \text{lift}_P(Y_i \leq Y_j) &= (Y_i \leq Y_j) \wedge \neg \bigvee_{\ell \in P} (Y_i \leq Y_\ell) \vee (Y_j \leq Y_\ell) \\ \text{lift}_P(Y_{i,j} \leq Y_k) &= (Y_{i,j} \leq Y_k) \wedge \neg \bigvee_{\ell \in P} (Y_{i,j} \leq Y_\ell) \vee (Y_k \leq Y_\ell) \end{aligned}$$

The announced statement is easily verified for these atomic formulas. Similarly, boolean combinations of formulas are handled by letting $\text{lift}_P(\varphi \vee \psi) = \text{lift}_P(\varphi) \vee \text{lift}_P(\psi)$ and $\text{lift}_P(\neg \varphi) = \neg \text{lift}_P(\varphi)$.

The last, and more interesting case, is that where $\varphi = \langle \mathcal{A} \rangle$, for a past path automaton $\mathcal{A} = (Q, \Delta, I, F)$. We let $\text{lift}_P(\langle \mathcal{A} \rangle) = \langle \mathcal{A}_P \rangle$, where $\mathcal{A}_P = (Q', \Delta', I', F')$ is the path automaton defined as follows:

- $Q' = Q \times 2^{\mathcal{P}}$, $I' = I \times \{P\}$ and $F' = F \times 2^{\mathcal{P}}$,
- for each test transition $(q_1, \varphi?, q_2) \in \Delta$ of \mathcal{A} and each set $P_1 \subseteq \mathcal{P}$, we define the test transition $((q_1, P_1), \text{lift}_{P_1}(\varphi)?, (q_2, P_1))$ in \mathcal{A}_P ,
- for each move transition $(q_1, \leftarrow_k, q_2) \in \Delta$ of \mathcal{A} and each sets $P_1, P_2 \subseteq \mathcal{P}$ with $P_1 \subseteq P_2$, we define a *test and move*¹ transition $((q_1, P_1), \text{change}_{k, P_1, P_2}?, \leftarrow_k, (q_2, P_2))$ in \mathcal{A}_P where

$$\text{change}_{k, P_1, P_2} = \left(\neg \bigvee_{i \in P_1} Y_k \leq Y_i \right) \wedge \left(\bigwedge_{j \in P_2 \setminus P_1} \bigvee_{i \in P_1} Y_{k,j} \leq Y_i \right) \wedge \left(\bigwedge_{j \notin P_2} \neg \bigvee_{i \in P_1} Y_{k,j} \leq Y_i \right)$$

¹ Formally, in order to comply with the definition of a path automaton, we should split each test and move transition into a test transition followed by a move transition with a new intermediary state in-between.

The formula above characterises the change of context when moving from an event e on process k to the previous event e_k on process k . It is based on Lemma 5. The first conjunct says that $\text{res}_{P_1}(e) \cap \downarrow e_k$ is nonempty. The remaining conjuncts characterises the set P_2 such that $\text{res}_{P_1}(e) \cap \downarrow e_k = \text{res}_{P_2}(e_k)$.

Observe that \mathcal{A}_P is again a *past* automaton and all reachable states (q, P') in \mathcal{A}_P satisfy $P \subseteq P'$. Moreover, if \mathcal{A} is k -local then so is \mathcal{A}_P .

We claim that this construction is correct, *i.e.*, $\text{res}_P(e), e \models \langle \mathcal{A} \rangle$ if and only if $t, e \models \langle \mathcal{A}_P \rangle$. The proof of this claim is in Appendix A. \blacktriangleleft

We can finally complete the proof of Theorem 4, which shows that, as far as prime traces are concerned, LocPastPDL event formulas can express all regular properties.

Proof of Theorem 4. We establish a more precise statement: for $m \in M$ and $P \subseteq \mathcal{P}$, we construct a LocPastPDL event formula $\varphi_P^{(m)}$ such that, if t is a prime trace satisfying $\text{loc}(t \setminus \{\max(t)\}) \subseteq P$, then $\eta(t) = m$ if and only if $t, \max(t) \models \varphi_P^{(m)}$. The statement of the theorem corresponds to the case $P = \mathcal{P}$.

The proof is by induction on the cardinality of P . If $P = \emptyset$, a prime trace t satisfying the condition $\text{loc}(t \setminus \{\max(t)\}) \subseteq P$ consists of the single event $\max(t)$. Therefore, we let

$$\varphi_\emptyset^{(m)} = \bigvee_{a \in \Sigma \text{ s.t. } \eta(a)=m} a.$$

Assume that $P \neq \emptyset$ and consider a prime trace $t = (E, \leq, \lambda)$ satisfying $\text{loc}(t \setminus \{\max(t)\}) \subseteq P$. If $P \cap \text{loc}(\max(t)) = \emptyset$, the primality of t implies that E is a singleton, and we let $\varphi_P^{(m)} = \varphi_\emptyset^{(m)}$.

If $P \cap \text{loc}(\max(t)) \neq \emptyset$, we pick $k \in P \cap \text{loc}(\max(t))$. Let $f_1 < f_2 < \dots < f_\ell$ be the sequence of events in E_k . In particular, $\ell \geq 1$ and $\max(t) = f_\ell$. For each $1 \leq i \leq \ell$, let $t_i = \text{res}_{\{k\}}(f_i)$. Then $t_i = \downarrow f_i \setminus \downarrow f_{i-1}$ (letting $\downarrow f_0 = \emptyset$) and hence, $t = t_1 t_2 \dots t_\ell$ and $\eta(t) = \eta(t_1) \eta(t_2) \dots \eta(t_\ell)$. By construction, $\text{loc}(t_i \setminus \{f_i\}) \subseteq P \setminus \{k\}$ for each $1 \leq i \leq \ell$ and we can use the induction hypothesis: $\eta(t_i) = m'$ if and only if $t_i, f_i \models \varphi_{P \setminus \{k\}}^{(m')}$. Using the lifting lemma (Lemma 6), we then get that $\eta(t_i) = m'$ if and only if $t, f_i \models \text{lift}_{\{k\}}(\varphi_{P \setminus \{k\}}^{(m')})$.

The membership of a trace t in $\eta^{-1}(m)$ – subject to the current assumption that t is prime, $\text{loc}(t \setminus \{\max(t)\}) \subseteq P$ and $k \in P \cap \text{loc}(\max(t))$ – can be computed by a k -local path automaton $\mathcal{A}_{P,k}^{(m)}$ as follows: we let $\mathcal{A}_{P,k}^{(m)} = (M \cup \{\$, \Delta, 1_M, \$\})$, where the initial state is the unit 1_M of the monoid M , the final state is $\$$ and Δ consists of two types of transitions:

1. test and move transitions of the form $(m_1, \text{act}_{m_1, m_2}^? \cdot \leftarrow_k, m_2)$ where $m_1, m_2 \in M$ and

$$\text{act}_{m_1, m_2} = \bigvee_{m' \mid m_2 = m' m_1} \text{lift}_{\{k\}}(\varphi_{P \setminus \{k\}}^{(m')})$$

The intuition is as follows. We use the notations above. Assume that \leftarrow_k moves from f_i to f_{i-1} and that, at f_i , the automaton has already computed $m_1 = \eta(t_{i+1} \dots t_\ell) = \eta(\downarrow f_\ell \setminus \downarrow f_i)$. We have seen that $t, f_i \models \text{lift}_{\{k\}}(\varphi_{P \setminus \{k\}}^{(m')})$ if and only if $\eta(t_i) = m'$. Since the disjunction ranges over all m' with $m_2 = m' m_1$, we deduce that $m_2 = \eta(t_i \dots t_\ell) = \eta(\downarrow f_\ell \setminus \downarrow f_{i-1})$. Therefore, walking down the sequence f_ℓ, \dots, f_1 , the automaton computes the values of η on the suffixes $t_i \dots t_\ell$.

2. (accepting) test transitions of the form $(m_1, \{\text{act}_{m_1, m} \wedge \neg(\leftarrow_k)\}^?, \$)$, where $m_1 \in M$.

To conclude, we let

$$\varphi_P^{(m)} = \left(\bigvee_{a \in \Sigma \mid \eta(a)=m, \text{loc}(a) \cap P = \emptyset} a \right) \vee \left(\bigvee_{k \in P, a \in \Sigma_k} a \wedge \langle \mathcal{A}_{P,k}^{(m)} \rangle \right). \quad \blacktriangleleft$$

4.2 Expressivity of sentences in LocPastPDL

We now generalize Theorem 4 from prime traces to all traces, which concludes the proof of Theorem 3. The precise statement of this generalization is as follows.

► **Theorem 7.** *Let $\eta: \text{Tr}(\Sigma) \rightarrow M$ be a morphism to a finite monoid. For each $m \in M$, we can construct a LocPastPDL sentence $\Phi^{(m)}$ such that, for all traces $t \in \text{Tr}(\Sigma)$, we have $\eta(t) = m$ iff $t \models \Phi^{(m)}$.*

As in Section 4.1, where we dealt with event formulas, we introduce a notion of *trace residuation*. If $t = (E, \leq, \lambda)$ is a trace, $i \in \mathcal{P}$ is a process and $P \subseteq \mathcal{P}$ is a set of processes, we let $\text{res}_{i,P}(t)$, the *residue of process i with respect to P* , be the trace induced by the set of events $\downarrow E_i \setminus \downarrow E_P$. Here E_P denotes the set $\bigcup_{k \in P} E_k$, of all events involving a process in P . We record the following result, analogous to Lemma 5 (proof in Appendix A).

► **Lemma 8.** *Let t be a trace, $i \in \mathcal{P}$ a process and $P \subseteq \mathcal{P}$ a set of processes. Then we have*

$$\text{res}_{i,P}(t) = \begin{cases} \varepsilon & \text{if } t \models \neg(L_i \leq L_i) \vee \bigvee_{j \in P} (L_i \leq L_j), \\ \text{res}_{P'}(e) & \text{otherwise,} \end{cases}$$

where $e = \max E_i$ and $P' = \{j \in \mathcal{P} \mid t \models \bigvee_{k \in P} L_{i,j} \leq L_k\}$.

Proof of Theorem 7. The proof consists in identifying a particular, LocPastPDL-definable decomposition of a trace t as a product of prime traces, and using Theorem 4 to handle its factors.

Let $t = (E, \leq, \lambda)$ be a non-empty trace and let e_1, \dots, e_ℓ be its maximal events. We choose a process $i_k \in \text{loc}(e_k)$ for each maximal event. As maximal events are pairwise concurrent, the i_k are pairwise distinct. We let $t_1 = \downarrow e_1$ and, for $1 < k \leq \ell$, we let $Q_k = \{i_1, \dots, i_{k-1}\}$ and $t_k = \text{res}_{i_k, Q_k}(t)$. In particular, each t_k is a non-empty prime trace and $t = t_1 \cdot t_2 \cdots t_\ell$.

For each tuple i_1, \dots, i_ℓ of pairwise distinct processes, the following LocPastPDL-sentence checks that a trace has ℓ maximal events located on processes i_1, \dots, i_ℓ (we use i as an abbreviation for the event formula $\bigvee_{a \in \Sigma_i} a$):

$$\text{MAX}_{i_1, \dots, i_\ell} = \left(\bigwedge_{1 \leq k \leq \ell} \text{EM } i_k \right) \wedge \neg \text{EM} \neg \left(\bigvee_{1 \leq k \leq \ell} i_k \right) \wedge \neg \text{EM} \left(\bigvee_{1 \leq k, k' \leq \ell, k \neq k'} i_k \wedge i_{k'} \right)$$

Letting $P_1 = \emptyset$ we have $t_1 = \text{res}_{P_1}(e_1)$. Using Lemma 8, we find subsets $P_k \subseteq \mathcal{P}$ such that $t_k = \text{res}_{P_k}(e_k)$ for each $1 < k \leq \ell$. We note that Lemma 8 also justifies the following specification of the sets P_k : assuming that $t \models \text{MAX}_{i_1, \dots, i_\ell}$, these sets P_k are characterized by the sentence

$$\bigwedge_{1 \leq k \leq \ell} \text{RES}_{i_k, \{i_1, \dots, i_{k-1}\}}^{P_k}$$

where

$$\text{RES}_{i,P}^{P'} = \left(\bigwedge_{j \in P'} \bigvee_{k \in P} L_{i,j} \leq L_k \right) \wedge \left(\bigwedge_{j \notin P'} \neg \bigvee_{k \in P} L_{i,j} \leq L_k \right).$$

Finally, once the sequence i_1, \dots, i_ℓ and the sets P_1, \dots, P_ℓ are fixed, the equality $\eta(t) = m$ is checked by the sentence

$$\bigvee_{m=m_1 \cdots m_\ell} \bigwedge_{1 \leq k \leq \ell} \text{EM} \left(i_k \wedge \text{lift}_{P_k}(\varphi^{(m_k)}) \right),$$

where the $\varphi^{(m_k)}$ ($1 \leq k \leq \ell$) are given by Theorem 4.

To conclude the proof of the theorem, if $m \neq 1_M$, we let $\Phi^{(m)}$ be the sentence

$$\bigvee_{\substack{i_1, \dots, i_\ell \\ P_1, \dots, P_\ell}} \text{MAX}_{i_1, \dots, i_\ell} \wedge \bigwedge_{1 \leq k \leq \ell} \text{RES}_{i_k, \{i_1, \dots, i_{k-1}\}}^{P_k} \wedge \bigvee_{m=m_1 \dots m_\ell} \bigwedge_{1 \leq k \leq \ell} \text{EM} \left(i_k \wedge \text{lift}_{P_k}(\varphi^{(m_k)}) \right). \quad (1)$$

Note that, the empty trace does not satisfy $\text{MAX}_{i_1, \dots, i_\ell}$ for any tuple (i_1, \dots, i_ℓ) with $\ell > 0$, and hence it does not satisfy the formula in Equation (1), with $m = 1_M$. Therefore, we let $\Phi^{(1_M)}$ be the disjunction of $\neg \text{EM} \top$ (which specifies the empty trace) and the formula in Equation (1), with $m = 1_M$. ◀

5 Asynchronous automata and local cascade products

In Section 6, we exploit the expressive completeness of LocPastPDL established above to give a Krohn-Rhodes style decomposition result for regular trace languages. Here, we first review the distributed model of asynchronous automata (Zielonka, [24]), seen both as acceptors of trace languages and as letter-to-letter trace transducers, and the related cascade product.

Asynchronous automata. work in a concurrent manner on traces over a distributed alphabet, say (Σ, loc) . They have local states, for each process in \mathcal{P} , and their transitions on a letter $a \in \Sigma$ read and update only the states that are local to a process in $\text{loc}(a)$. Formally, an *asynchronous automaton* A over (Σ, loc) is a tuple $(\{S_i\}_{i \in \mathcal{P}}, \{\delta_a\}_{a \in \Sigma}, s_{\text{in}})$ where

- S_i is a finite non-empty set of *local i -states* for each process i ;
- For $a \in \Sigma$, let $S_a = \prod_{i \in \text{loc}(a)} S_i$ be called the set of *a -states*. Then $\delta_a: S_a \rightarrow S_a$ is a (deterministic and complete) *transition function on a -states*;
- $s_{\text{in}} \in S$ (where $S = \prod_{i \in \mathcal{P}} S_i$ is called the set of *global states*) is the *initial global state*.

If s is a global state, we write s_a for its projection on S_a and s_{-a} for its projection on the remaining processes. It is convenient to write $s = (s_a, s_{-a})$.

For $a \in \Sigma$, let $\Delta_a: S \rightarrow S$ be the *global transition function* defined by $\Delta_a((s_a, s_{-a})) = (\delta_a(s_a), s_{-a})$. Composing these functions defines the global transition Δ_t of any trace $t \in \text{Tr}(\Sigma)$: we let Δ_ε be the identity function and, if $t = t'a$, then $\Delta_t = \Delta_a \circ \Delta_{t'}$. We denote by $A(t)$ the global state reached when running A on t , that is, $A(t) = \Delta_t(s_{\text{in}})$.

Zielonka's fundamental theorem [24] states that a trace language is recognizable if and only if it is accepted by some asynchronous automaton A , that is, if there exists a subset $S_{\text{fin}} \subseteq S$ of *final global states* such that $L = \{t \in \text{Tr}(\Sigma) \mid A(t) \in S_{\text{fin}}\}$.

Asynchronous labeling functions. Asynchronous automata can be used not only as acceptors, as above, but also as devices to compute certain functions on traces: maps which, given a trace $t = (E, \leq, \lambda)$, compute a trace with the same underlying poset structure (E, \leq) , and with a richer labeling function.

That point of view, which was developed by the authors in [1, 2], generalizes the notion of sequential letter-to-letter word transducers, and is closely related to the *locally computable functions* defined in [19].

Formally, let (Σ, loc) be a distributed alphabet and Γ be a finite non-empty set. Then $\Sigma \times \Gamma$ is a distributed alphabet (over the same set \mathcal{P} of processes as (Σ, loc)) for the location function given by $\text{loc}(a, \gamma) = \text{loc}(a)$ for every $(a, \gamma) \in \Sigma \times \Gamma$. A map $\theta: \text{Tr}(\Sigma) \rightarrow \text{Tr}(\Sigma \times \Gamma)$ is called a Γ -*labeling function* if, for each $t = (E, \leq, \lambda) \in \text{Tr}(\Sigma)$, we have $\theta(t) = (E, \leq, (\lambda, \mu))$, *i.e.*, θ adds a new label $\mu(e) \in \Gamma$ to each event e in t .

► **Example 9.** Let F be a finite set of LocPastPDL event formulas and $\Gamma_F = \{0, 1\}^F$. For each trace $t \in \text{Tr}(\Sigma)$ and event e of t , we let $\mu_F(e)$ be the tuple of truth values of each $\varphi \in F$ at e . We then let θ^F be the Γ_F -labeling function which maps a trace $t = (E, \leq, \lambda) \in \text{Tr}(\Sigma)$ to the trace $(E, \leq, (\lambda, \mu_F)) \in \text{Tr}(\Sigma \times \Gamma_F)$.

An *asynchronous (letter-to-letter) Γ -transducer* over (Σ, loc) is a tuple $\hat{A} = (A, \{\mu_a\})$ where $A = (\{S_i\}, \{\delta_a\}, s_{\text{in}})$ is an asynchronous automaton and each μ_a ($a \in \Sigma$) is a map $\mu_a: S_a \rightarrow \Gamma$. We associate with \hat{A} the Γ -labeling function, also denoted by \hat{A} , from $\text{Tr}(\Sigma)$ to $\text{Tr}(\Sigma \times \Gamma)$, which maps $t = (E, \leq, \lambda)$ to $\hat{A}(t) = (E, \leq, (\lambda, \mu))$ in such a way that, for every event $e \in E$ with $\lambda(e) = a$ and $s = A(\downarrow e)$, we have $\mu(e) = \mu_a(s_a)$. We say that \hat{A} *computes* (or *implements*) the Γ -labeling function \hat{A} . We also say that an asynchronous automaton $A = (\{S_i\}, \{\delta_a\}, s_{\text{in}})$ *computes* a Γ -labeling function θ if there are maps $\mu_a: S_a \rightarrow \Gamma$ such that $\theta = \hat{A}$, with $\hat{A} = (A, \{\mu_a\})$.

Notice that a Γ -labeling function is defined on every input trace, hence an asynchronous transducer admits a run on all traces and it does not use an acceptance condition.

► **Example 10.** Let $i \in \mathcal{P}$ be a process, let $\varphi_i = (\mathbf{Y}_i \leq \mathbf{Y}_i)$ be the LocPastPDL event formula which states that there is an event on process i in the strict past of the current event. With reference to Example 9, $\Gamma_F = \{0, 1\}$ and the Γ_F -labeling function θ^F is computed by the following asynchronous transducer. For each process j , the set of j -states is $\{0, 1\}$, and the global initial state has every process start in state 0. When the first event e occurs on process i , all processes in $\text{loc}(e)$ switch to state 1: for every $a \in \Sigma_i$, δ_a is the constant map sending all states in S_a to $(1, \dots, 1)$. This information is then propagated via synchronizing events: for every $b \in \Sigma \setminus \Sigma_i$, the map δ_b sends $(0, \dots, 0)$ to itself and every other state to $(1, \dots, 1)$. It is easy to add output functions $\{\mu_a\}_{a \in \Sigma}$ in order to compute θ^F .

Local cascade product. It turns out that the composition of labeling functions computed by asynchronous transducers, can also be computed by an asynchronous transducer. This asynchronous transducer is the result of the *local cascade product* operation defined below.

► **Definition 11.** Let $\hat{A} = (\{S_i\}, \{\delta_a\}, s_{\text{in}}, \{\mu_a\})$ be a Γ -labeling asynchronous transducer over (Σ, loc) , and let $\hat{B} = (\{Q_i\}, \{\delta_{(a,\gamma)}\}, q_{\text{in}}, \{\nu_{(a,\gamma)}\})$ be a Π -labeling asynchronous transducer over $(\Sigma \times \Gamma, \text{loc})$. We define the local cascade product of \hat{A} and \hat{B} to be the $(\Gamma \times \Pi)$ -labeling asynchronous transducer $\hat{A} \circ_{\ell} \hat{B} = (\{S_i \times Q_i\}, \{\nabla_a\}, (s_{\text{in}}, q_{\text{in}}), \{\tau_a\})$ where $\nabla_a((s_a, q_a)) = (\delta_a(s_a), \delta_{(a, \mu_a(s_a))}(q_a))$ and $\tau_a: S_a \times Q_a \rightarrow \Gamma \times \Pi$ is defined by $\tau_a((s_a, q_a)) = (\mu_a(s_a), \nu_{(a, \mu_a(s_a))}(q_a))$.

► **Remark 12.** It is directly verified that, with the notation of Definition 11, if \hat{A} implements $f_A: \text{Tr}(\Sigma) \rightarrow \text{Tr}(\Sigma \times \Gamma)$ and \hat{B} implements $f_B: \text{Tr}(\Sigma \times \Gamma) \rightarrow \text{Tr}(\Sigma \times \Gamma \times \Pi)$ then the local cascade product $\hat{A} \circ_{\ell} \hat{B}$ implements the composition $f_B \circ f_A: \text{Tr}(\Sigma) \rightarrow \text{Tr}(\Sigma \times \Gamma \times \Pi)$.

In the sequential case, that is, when $|\mathcal{P}| = 1$, the local cascade product coincides with the well-known operation of cascade product of sequential letter-to-letter transducers.

Slightly abusing language, we view a local cascade product also as an asynchronous automaton (forgetting the local labeling functions) and we can use it to accept trace languages as well. The celebrated theorem by Krohn and Rhodes [16] characterizes regular word languages as those accepted by cascade products of two simple kinds of automata:

- 2-state reset automata, where the transition function of each letter is either the identity function or constant;
- permutation automata, where each letter induces a permutation of the state set.

► **Theorem 13** (Krohn-Rhodes [16]). *Any regular word language is accepted by a cascade product of 2-state reset automata and permutation automata.*

In the setting of traces, we consider distributed analogues of reset and permutation automata. If $i \in \mathcal{P}$ is a process, a *2-state reset automaton localized at i* is an asynchronous automaton with two i -local states, where all other local state sets are singletons and the local transition induced by each letter is either the identity function, or a constant function. Similarly, a *permutation automaton localized at i* is an asynchronous automaton where each set of j -states ($j \neq i$) is a singleton and the local transition by any letter is a permutation.

Another important asynchronous automaton is Mukund and Sohoni's *gossip (asynchronous) automaton \mathcal{G}* , see [19]. Its main purpose is to compute the primary and secondary comparisons, as stated in the theorem below.

► **Theorem 14** (Mukund-Sohoni [19]). *Let $Y = \{Y_i \leq Y_j, Y_{i,j} \leq Y_k \mid i, j, k \in \mathcal{P}\}$ be the set of all constant event formulas of LocPastPDL and let θ^Y be the corresponding labeling function. The gossip automaton \mathcal{G} computes θ^Y .*

We say that a local cascade product $\hat{\mathcal{G}} \circ_{\ell} \hat{B}$ is *restricted* (or θ^Y -*restricted*) if the labeling function computed by $\hat{\mathcal{G}}$ is θ^Y , i.e., the information passed to \hat{B} by $\hat{\mathcal{G}}$ is restricted to the truth values of the event formulas in Y .

► **Remark 15.** The gossip automaton \mathcal{G} also computes the truth values of the constant trace sentences $L = \{L_i \leq L_j, L_{i,j} \leq L_k \mid i, j, k \in \mathcal{P}\}$, this time *globally*. More precisely, if S is the global state set of \mathcal{G} , then there is a map $\zeta: S \rightarrow \{0, 1\}^L$ such that, for every trace t , and sentence $\Phi \in L$, $t \models \Phi$ if and only if the Φ -component of $\zeta(\mathcal{G}(t))$ is 1.

6 Cascade decomposition

The following is the main result of this section.

► **Theorem 16.** *Let φ be a LocPastPDL event formula and θ^φ (for $\theta^{\{\varphi\}}$) be the corresponding $\{0, 1\}$ -labeling function. One can construct a restricted cascade product of the gossip automaton followed by a local cascade product of localized reset and permutation automata, which computes θ^φ .*

Before we prove Theorem 16, we establish an important corollary.

► **Corollary 17.** *Any regular trace language is accepted by a restricted local cascade product of the gossip automaton and a local cascade product of localized reset automata and localized permutation automata.*

Proof. By Theorem 3, any regular trace language L is defined by a sentence Φ in LocPastPDL. If Φ is of the form $L_i \leq L_j$ or $L_{i,j} \leq L_k$, then L is accepted by the gossip automaton, by Remark 15. The case where Φ is a non-trivial boolean combination is easily handled, and we are left with sentences of the form $EM\varphi$.

If $i \in \mathcal{P}$ is a process, let $EM_i\varphi$ be the sentence which expresses that a trace has at least one i -event, and that its maximum i -event satisfies φ . Then $EM\varphi$ is equivalent to the disjunction $\bigvee_{i \in \mathcal{P}} \left(EM_i\varphi \wedge \neg(\bigvee_{j \in \mathcal{P}} L_i < L_j) \right)$, where $L_i < L_j = (L_i \leq L_j) \wedge \neg(L_j \leq L_i)$, so we only need to deal with sentences of the form $EM_i\varphi$.

By Theorem 16, the labeling function θ^φ is computed by an asynchronous transducer A_φ of the required local cascade form. Let B be the localized reset automaton with local i -states $\{q_0, q_1\}$ (and other local state sets singletons), initial state q_0 , on alphabet $\Sigma \times \{0, 1\}$, with

the following transitions. The transition induced by a letter $(a, 1)$ such that $i \in \text{loc}(a)$ is a constant map to q_1 . Other transitions labeled $(a, 0)$ with $i \in \text{loc}(a)$ are the constant map to q_0 . Then $A_\varphi \circ_\ell B$ recognizes $\text{EM}_i \varphi$ when the global final state of the B component is q_1 . Notice that B only needs to check if the maximal i -event of t satisfies φ , an information which is already added to the label of this event by A_φ (as 0 or 1). ◀

We now move towards the proof of Theorem 16. We first associate with each local path automaton a regular word language over a decorated alphabet. Specifically, let \mathcal{A} be an i -local path automaton and let F be the set of event formulas in its test transitions. Recall that, if $t \in \text{Tr}(\Sigma)$ and e, f are events in t , we have $t, e, f \models \mathcal{A}$ if there is an accepting run $q_0 \xrightarrow{\alpha_1} q_1 \cdots q_{n-1} \xrightarrow{\alpha_n} q_n$ and a sequence of events $e = e_0, e_1, \dots, e_{n-1}, e_n = f$ such that for all $0 \leq m < n$ we have $t, e_m, e_{m+1} \models \alpha_m$. Checking whether $t, e_m, e_{m+1} \models \varphi?$ for some $\varphi \in F$ is done by a simple inspection of the label of event $e_m = e_{m+1}$ in $\theta^F(t) \in \text{Tr}(\Sigma \times \Gamma_F)$. Observe also that, since \mathcal{A} is i -local, all the e_m are i -events. This leads to the definition of a word language $\mathcal{L}_F(\mathcal{A})$ over the alphabet $\Sigma_i \times \Gamma_F$. Each word w in $\mathcal{L}_F(\mathcal{A})$ is induced by a trace t and a pair of events e, f such that $t, e, f \models \mathcal{A}$, and consists of the sequence of labels in $\theta^F(t)$ of the i -events from f to e .

$$\mathcal{L}_F(\mathcal{A}) = \{\theta^F(t) \cap E_i \cap \downarrow e \cap \uparrow f \mid t, e, f \models \mathcal{A}\} \subseteq (\Sigma_i \times \Gamma_F)^*.$$

► **Lemma 18.** *Let \mathcal{A} be an i -local path automaton and let F be the set of event formulas in its test transitions. Then $\mathcal{L}_F(\mathcal{A})$ is a regular language.*

Proof. The automaton \mathcal{A} accepts a regular language over the alphabet $\{\varphi? \mid \varphi \in F\} \cup \{\leftarrow_i\}$. Processing a letter from the alphabet translates to a move to a different event in the trace (see the semantics of path automata recalled above) if that letter is \leftarrow_i , but not if it is of the form $\varphi?$. To smooth out this difference, we modify \mathcal{A} to an automaton \mathcal{B} with the same semantics (in the sense that $t, e, f \models \mathcal{A}$ if and only if $t, e, f \models \mathcal{B}$), where transitions to an accepting state have labels of the form $\psi?$ and all other transitions have a label of the form $\psi'? \leftarrow_i$, where ψ, ψ' are conjunctions of formulas in F (we talk of *test-and-move* transitions).

Let Q be the set of states of \mathcal{A} and let $q_a \notin Q$ be a new state. The set of states of \mathcal{B} is $Q' = Q \cup \{q_a\}$, \mathcal{B} has the same initial states as \mathcal{A} , and q_a is the only accepting state of \mathcal{B} . The transitions of \mathcal{B} are as follows.

1. Let $q, q' \in Q$ and let ψ be a conjunction of formulas in F . \mathcal{B} has a test-and-move transition from q to q' labeled $\psi? \cdot \leftarrow_i$ in \mathcal{B} if there is a path from q to q' in \mathcal{A} starting with a sequence of test transitions using exactly all the conjuncts of ψ and ending with a move transition (labeled \leftarrow_i).
2. Let $q \in Q$ and let ψ be a conjunction of formulas in F . \mathcal{B} has a test transition from q to q_a labeled $\psi?$ if there is a path in \mathcal{A} from q to some accepting state q' of \mathcal{A} consisting of test transitions using exactly all the conjuncts of ψ .

It is not difficult to see that \mathcal{A} and \mathcal{B} have the same semantics. For each $\varphi \in F$, let Δ_φ be the set of letters $(a, \gamma) \in \Sigma_i \times \Gamma_F$ such that the φ -component of γ is 1. We now modify \mathcal{B} into a new automaton \mathcal{B}' by changing the labels of transitions: for each edge labeled by $\bigwedge_{j=1}^k \varphi_j? \leftarrow_i$ (resp. $\bigwedge_{j=1}^k \varphi_j?$), replace the label with $\bigcap_{j=1}^k \Delta_{\varphi_j}$. The automaton \mathcal{B}' , over $\Sigma_i \times \Gamma_F$ is easily seen to accept the reverse language of $\mathcal{L}_F(\mathcal{A})$, so $\mathcal{L}_F(\mathcal{A})$ itself is regular. ◀

We can finally prove Theorem 16, the last missing element of this paper.

Proof of Theorem 16. The proof is by structural induction on the LocPastPDL event formula φ . If $\varphi = a$ ($a \in \Sigma$), A_a is the asynchronous transducer where each set of local states is a singleton. The labeling function is defined by $\mu_b(s) = 1$ if $b = a$ and 0 otherwise. If $\varphi = Y_i \leq Y_k$ or $\varphi = Y_{i,j} \leq Y_k$, Theorem 14 shows that we can use the gossip automaton as A_φ . Boolean combination of event formulae are easily handled.

The case where $\varphi = \langle \mathcal{A} \rangle$, for some i -local path automaton \mathcal{A} , is non-trivial. Let F be the set of event formulas in the test transitions of \mathcal{A} . By induction hypothesis, for each $\psi \in F$, we have an asynchronous transducer A_ψ in the required local cascade form which computes θ^ψ . By the usual direct product construction, which can be subsumed by a local cascade product (factorizing the gossip automaton), we then have an asynchronous transducer A_F in the required form which computes θ^F . We then construct A_φ in the form of a local cascade product $A_F \circ_\ell B$ for an appropriate asynchronous transducer B on alphabet $\Sigma \times \Gamma_F$.

Let $\mathcal{L}_F(A)$ be the language (over alphabet $\Sigma_i \times \Gamma_F$) defined above, which is regular by Lemma 18, and let C be an automaton accepting $(\Sigma_i \times \Gamma_F)^* \cdot \mathcal{L}_F(A)$. By Krohn-Rhodes's theorem (Theorem 13 above), C can be chosen to be a cascade product of 2-state reset automata and permutation automata. Let us *localize* each of these automata at process i , by adding singleton local state sets for each process $j \neq i$. The resulting local cascade product (of localized reset and permutation asynchronous transducers) allows us to check whether an i -local state is final in C or not. It is easily verified that a labeling function can then be imposed on B , by which $A_F \circ_\ell B$ computes θ^φ .

This completes the proof of Theorem 16. ◀

7 Conclusion

We have shown that LocPastPDL is expressively complete. Recall that a basic trace formula of LocPastPDL is either of the form $EM \varphi$ or a constant comparison formula such as $L_i \leq L_j$ or $L_{i,j} \leq L_k$. We could instead use basic trace formula $EM_i \varphi$ which asserts that the maximum i -event exists and satisfies the event formula φ . It follows from the results in [3] that boolean combinations of $EM_i \varphi$ suffice to arrive at an expressively complete logic. Note that our expressive-completeness proof of LocPastPDL is direct and self-contained. In view of this, it would be interesting to directly express $L_i \leq L_j$ and $L_{i,j} \leq L_k$ using only basic formulas of the form $EM_i \varphi$. Another exciting question concerns the necessity of the primary and secondary event comparison formulas $Y_i \leq Y_j$ and $Y_{i,j} \leq Y_k$ for the expressive completeness result. This is also intimately related to the necessity of the gossip automaton in our distributed Krohn-Rhodes theorem. It would be also interesting to identify a natural fragment of LocPastPDL which matches first-order logic in expressive power, and also extend the results in this work to infinite traces.

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. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPICs*, pages 19:1–19:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
- 2 Bharat Adsul, Paul Gastin, Saptarshi Sarkar, and Pascal Weil. Asynchronous wreath product and cascade decompositions for concurrent behaviours. *Log. Methods Comput. Sci.*, 18, 2022.
- 3 Bharat Adsul and Milind A. Sohoni. Local normal forms for logics over traces. In Manindra Agrawal and Anil Seth, editors, *FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science, 22nd Conference Kanpur, India, December 12-14, 2002, Proceedings*, volume 2556 of *Lecture Notes in Computer Science*, pages 47–58. Springer, 2002.

- 4 Benedikt Bollig, Marie Fortin, and Paul Gastin. Communicating finite-state machines, first-order logic, and star-free propositional dynamic logic. *J. Comput. Syst. Sci.*, 115:22–53, 2021. doi:10.1016/j.jcss.2020.06.006.
- 5 Benedikt Bollig and Paul Gastin. Non-sequential theory of distributed systems. *CoRR*, abs/1904.06942, 2019. arXiv:1904.06942.
- 6 Benedikt Bollig, Dietrich Kuske, and Ingmar Meinecke. Propositional dynamic logic for message-passing systems. *Log. Methods Comput. Sci.*, 6(3), 2010. arXiv:1007.4764.
- 7 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.
- 8 Volker Diekert and Paul Gastin. LTL is expressively complete for Mazurkiewicz traces. *Journal of Computer and System Sciences*, 64(2):396–418, March 2002.
- 9 Volker Diekert and Paul Gastin. Pure future local temporal logics are expressively complete for mazurkiewicz traces. *Information and Computation*, 204(11):1597–1619, November 2006.
- 10 Volker Diekert and Paul Gastin. First-order definable languages. In Jörg Flum, Erich Grädel, and Thomas Wilke, editors, *Logic and Automata: History and Perspectives*, volume 2 of *Texts in Logic and Games*, pages 261–306. Amsterdam University Press, 2008.
- 11 Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces*. World Scientific, 1995. doi:10.1142/2563.
- 12 Werner Ebinger and Anca Muscholl. Logical definability on infinite traces. *Theor. Comput. Sci.*, 154(1):67–84, 1996.
- 13 Michael J Fischer and Richard E Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979.
- 14 Stefan Göller, Markus Lohrey, and Carsten Lutz. PDL with intersection and converse: satisfiability and infinite-state model checking. *J. Symb. Log.*, 74(1):279–314, 2009. doi:10.2178/js1/1231082313.
- 15 Giovanna Guaiana, Antonio Restivo, and Sergio Salemi. Star-free trace languages. *Theor. Comput. Sci.*, 97(2):301–311, 1992.
- 16 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.
- 17 Manfred Kufleitner. Polynomials, fragments of temporal logic and the variety DA over traces. *Theoretical Computer Science*, 376:89–100, 2007. Special issue DLT 2006.
- 18 Roy Mennicke. Propositional dynamic logic with converse and repeat for message-passing systems. *Log. Methods Comput. Sci.*, 9(2), 2013. doi:10.2168/LMCS-9(2:12)2013.
- 19 Madhavan Mukund and Milind A. Sohoni. Keeping track of the latest gossip in a distributed system. *Distributed Comput.*, 10(3):137–148, 1997. doi:10.1007/s004460050031.
- 20 Howard Straubing. *Finite automata, formal logic, and circuit complexity*. Birkhäuser Verlag, Basel, Switzerland, 1994.
- 21 P.S. Thiagarajan and I. Walukiewicz. An expressively complete linear time temporal logic for Mazurkiewicz traces. *Information and Computation*, 179(2):230–249, December 2002.
- 22 Wolfgang Thomas. On logical definability of trace languages. In V. Diekert, editor, *Proceedings of a workshop of the ESPRIT Basic Research Action No 3166: Algebraic and Syntactic Methods in Computer Science (ASMICS), Kochel am See, Bavaria, FRG (1989)*, Report TUM-I9002, Technical University of Munich, pages 172–182, 1990.
- 23 Nicolas Troquard and Philippe Balbiani. Propositional Dynamic Logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2019 edition, 2019. URL: <https://plato.stanford.edu/archives/spr2019/entries/logic-dynamic/>.
- 24 Wiesław Zielonka. Notes on finite asynchronous automata. *RAIRO Theor. Informatics Appl.*, 21(2):99–135, 1987. doi:10.1051/ita/1987210200991.

A Appendix for Section 4

► **Lemma 6** (Lifting lemma). *Let $\varphi \in \text{PastPDL}$ be an event formula and $P \subseteq \mathcal{P}$ be a set of processes. We can construct an event formula $\text{lift}_P(\varphi) \in \text{PastPDL}$ such that, for all traces $t = (E, \leq, \lambda)$ and events $e \in E$ in t , we have $\text{res}_P(e), e \models \varphi$ if and only if $t, e \models \text{lift}_P(\varphi)$. Moreover, if $\varphi \in \text{LocPastPDL}$ then $\text{lift}_P(\varphi) \in \text{LocPastPDL}$.*

Proof. The construction is by structural induction on φ . We first let

$$\begin{aligned} \text{lift}_P(a) &= a \text{ for each } a \in \Sigma \\ \text{lift}_P(Y_i \leq Y_j) &= (Y_i \leq Y_j) \wedge \neg \bigvee_{\ell \in P} (Y_i \leq Y_\ell) \vee (Y_j \leq Y_\ell) \\ \text{lift}_P(Y_{i,j} \leq Y_k) &= (Y_{i,j} \leq Y_k) \wedge \neg \bigvee_{\ell \in P} (Y_{i,j} \leq Y_\ell) \vee (Y_k \leq Y_\ell) \end{aligned}$$

The announced statement is easily verified for these atomic formulas. Similarly, boolean combinations of formulas are handled by letting $\text{lift}_P(\varphi \vee \psi) = \text{lift}_P(\varphi) \vee \text{lift}_P(\psi)$ and $\text{lift}_P(\neg\varphi) = \neg\text{lift}_P(\varphi)$.

The last, and more interesting case, is that where $\varphi = \langle \mathcal{A} \rangle$, for a past path automaton $\mathcal{A} = (Q, \Delta, I, F)$. We let $\text{lift}_P(\langle \mathcal{A} \rangle) = \langle \mathcal{A}_P \rangle$, where $\mathcal{A}_P = (Q', \Delta', I', F')$ is the path automaton defined as follows:

- $Q' = Q \times 2^{\mathcal{P}}$, $I' = I \times \{P\}$ and $F' = F \times 2^{\mathcal{P}}$,
- for each test transition $(q_1, \varphi?, q_2) \in \Delta$ of \mathcal{A} and each set $P_1 \subseteq \mathcal{P}$, we define the test transition $((q_1, P_1), \text{lift}_{P_1}(\varphi)?, (q_2, P_1))$ in \mathcal{A}_P ,
- for each move transition $(q_1, \leftarrow_k, q_2) \in \Delta$ of \mathcal{A} and each sets $P_1, P_2 \subseteq \mathcal{P}$ with $P_1 \subseteq P_2$, we define a *test and move*² transition $((q_1, P_1), \text{change}_{k, P_1, P_2}? \cdot \leftarrow_k, (q_2, P_2))$ in \mathcal{A}_P where

$$\text{change}_{k, P_1, P_2} = \left(\neg \bigvee_{i \in P_1} Y_k \leq Y_i \right) \wedge \left(\bigwedge_{j \in P_2 \setminus P_1} \bigvee_{i \in P_1} Y_{k,j} \leq Y_i \right) \wedge \left(\bigwedge_{j \notin P_2} \neg \bigvee_{i \in P_1} Y_{k,j} \leq Y_i \right)$$

The formula above characterises the change of context when moving from an event e on process k to the previous event e_k on process k . It is based on Lemma 5. The first conjunct says that $\text{res}_{P_1}(e) \cap \downarrow e_k$ is nonempty. The remaining conjuncts characterises the set P_2 such that $\text{res}_{P_1}(e) \cap \downarrow e_k = \text{res}_{P_2}(e_k)$.

Observe that \mathcal{A}_P is again a *past* automaton and all reachable states (q, P') in \mathcal{A}_P satisfy $P \subseteq P'$. Moreover, if \mathcal{A} is k -local then so is \mathcal{A}_P .

We claim that this construction is correct, *i.e.*, $\text{res}_P(e), e \models \langle \mathcal{A} \rangle$ if and only if $t, e \models \langle \mathcal{A}_P \rangle$.

Let us first assume that $\text{res}_P(e), e \models \langle \mathcal{A} \rangle$. There is an accepting run $q_0 \xrightarrow{\alpha_1} q_1 \cdots q_{n-1} \xrightarrow{\alpha_n} q_n$ of \mathcal{A} and a sequence of events $e = e_0, e_1, \dots, e_{n-1}, e_n$ such that for all $1 \leq m \leq n$ we have $\text{res}_P(e), e_{m-1}, e_m \models \alpha_m$. We construct inductively

- a sequence $P_0, P_1, \dots, P_n \subseteq \mathcal{P}$ so that $\downarrow e_m \cap \text{res}_P(e) = \text{res}_{P_m}(e_m)$ for all $0 \leq m \leq n$,
- an accepting run $(q_0, P_0) \xrightarrow{\beta_1} (q_1, P_1) \cdots (q_{n-1}, P_{n-1}) \xrightarrow{\beta_n} (q_n, P_n)$ of \mathcal{A}_P such that $t, e_{m-1}, e_m \models \beta_m$ for all $1 \leq m \leq n$.

We start with $P_0 = P$ so that $\downarrow e_0 \cap \text{res}_P(e) = \text{res}_{P_0}(e_0)$ and (q_0, P_0) is initial in \mathcal{A}_P . Now, let $0 < m \leq n$ and assume that we have constructed the sequence of sets and the run up to $m-1$. There are two cases.

² Formally, in order to comply with the definition of a path automaton, we should split each test and move transition into a test transition followed by a move transition with a new intermediary state in-between.

1. If $\alpha_m = \psi?$ is a test, we have $e_m = e_{m-1}$ and $\text{res}_P(e), e_{m-1} \models \psi$. Since ψ is a past formula and $\downarrow e_{m-1} \cap \text{res}_P(e) = \text{res}_{P_{m-1}}(e_{m-1})$, we deduce that $\text{res}_{P_{m-1}}(e_{m-1}), e_{m-1} \models \psi$. By our induction hypothesis, we then have $t, e_{m-1} \models \text{lift}_{P_{m-1}}(\psi)$, and we let $\beta_m = \text{lift}_{P_{m-1}}(\psi)?$ and $P_m = P_{m-1}$. With this definition, the required conditions are satisfied: $\downarrow e_m \cap \text{res}_P(e) = \text{res}_{P_m}(e_m)$, $((q_{m-1}, P_{m-1}), \beta_m, (q_m, P_m))$ is a transition in \mathcal{A}_P , and $t, e_{m-1}, e_m \models \beta_m$.
2. If $\alpha_m = \leftarrow_k$ is a left move, we have $e_{m-1}, e_m \in E_k$ and e_m is the predecessor of e_{m-1} on process k . In particular, e_m is the maximal event in $\downarrow e_{m-1} \cap E_k$. We apply Lemma 5: first, $\downarrow e_m \cap \text{res}_{P_{m-1}}(e_{m-1}) = \downarrow e_m \cap \text{res}_P(e)$ is nonempty since it contains e_m ; it follows that $t, e_{m-1} \models \neg \bigvee_{i \in P_{m-1}} Y_k \leq Y_i$. Let $P_m = P_{m-1} \cup \{j \in \mathcal{P} \mid t, e_{m-1} \models Y_{k,j} \leq Y_i \text{ for some } i \in P_{m-1}\}$. By Lemma 5, we have $\downarrow e_m \cap \text{res}_P(e) = \downarrow e_m \cap \text{res}_{P_{m-1}}(e) = \text{res}_{P_m}(e_m)$. By definition of P_m , we get $t, e_{m-1} \models \text{change}_{k, P_{m-1}, P_m}$. We then let $\beta_m = \text{change}_{k, P_{m-1}, P_m} \cdot \leftarrow_k$ so that $((q_{m-1}, P_{m-1}), \beta_m, (q_m, P_m))$ is a transition in \mathcal{A}_P , and $t, e_{m-1}, e_m \models \beta_m$.

Using the constructed run in \mathcal{A}_P and the same sequence of events $e = e_0, e_1, \dots, e_{n-1}, e_n$, we find that $t, e \models \langle \mathcal{A}_P \rangle$.

Conversely, assume that $t, e \models \text{lift}_P(\langle \mathcal{A} \rangle) = \langle \mathcal{A}_P \rangle$. There is an accepting run $(q_0, P_0) \xrightarrow{\beta_1} (q_1, P_1) \cdots (q_{n-1}, P_{n-1}) \xrightarrow{\beta_n} (q_n, P_n)$ of \mathcal{A}_P and a sequence of events $e = e_0, e_1, \dots, e_{n-1}, e_n$ such that $t, e_{m-1}, e_m \models \beta_m$ for all $1 \leq m \leq n$. We show by induction that $\downarrow e_m \cap \text{res}_P(e) = \text{res}_{P_m}(e_m)$ for all $0 \leq m \leq n$. We construct simultaneously a sequence $\alpha_1, \dots, \alpha_n$ such that $q_0 \xrightarrow{\alpha_1} q_1 \cdots q_{n-1} \xrightarrow{\alpha_n} q_n$ is an accepting run of \mathcal{A} and $\text{res}_P(e), e_{m-1}, e_m \models \alpha_m$ for all $1 \leq m \leq n$.

Since (q_0, P_0) is initial in \mathcal{A}_P , we have $P_0 = P$. Using $e_0 = e$, we get $\downarrow e_0 \cap \text{res}_P(e) = \text{res}_{P_0}(e_0)$. Now, assume that our properties hold up to $m-1$. There are two cases.

1. If $\beta_m = \text{lift}_{P_{m-1}}(\psi)?$ is a test, then $P_m = P_{m-1}$ and $e_m = e_{m-1}$. In particular, $\downarrow e_m \cap \text{res}_P(e) = \text{res}_{P_m}(e_m)$. Let $\alpha_m = \psi?$. By definition of \mathcal{A}_P , we know that (q_{m-1}, α_m, q_m) is a transition of \mathcal{A} . From $t, e_{m-1}, e_m \models \beta_m = \text{lift}_{P_{m-1}}(\psi)?$, we get $t, e_{m-1} \models \text{lift}_{P_{m-1}}(\psi)$ and, by the induction hypothesis, we obtain $\text{res}_{P_{m-1}}(e_{m-1}), e_{m-1} \models \psi$. Since ψ is a past formula and $\text{res}_{P_{m-1}}(e_{m-1}) = \downarrow e_{m-1} \cap \text{res}_P(e)$, we deduce that $\text{res}_P(e), e_{m-1} \models \psi$ and finally $\text{res}_P(e), e_{m-1}, e_m \models \alpha_m = \psi?$.
2. If $\beta_m = \text{change}_{k, P_{m-1}, P_m} \cdot \leftarrow_k$ is a test-and-move, we let $\alpha_m = \leftarrow_k$. Then $t, e_{m-1} \models \text{change}_{k, P_{m-1}, P_m}$ and (q_{m-1}, α_m, q_m) is a transition of \mathcal{A} . Moreover, $t, e_{m-1}, e_m \models \leftarrow_k$. Using the fact that $t, e_{m-1} \models \neg \bigvee_{i \in P_{m-1}} Y_k \leq Y_i$, Lemma 5 shows that $e_m \in \text{res}_{P_{m-1}}(e_{m-1}) = \downarrow e_{m-1} \cap \text{res}_P(e)$. Therefore, $\text{res}_P(e), e_{m-1}, e_m \models \leftarrow_k$. Finally, using Lemma 5 again and the fact that $t, e_{m-1} \models \text{change}_{k, P_{m-1}, P_m}$, we get $\text{res}_{P_m}(e_m) = \downarrow e_m \cap \text{res}_{P_{m-1}}(e_{m-1}) = \downarrow e_m \cap \text{res}_P(e)$.

Thus $\text{res}_P(e), e \models \langle \mathcal{A} \rangle$, and this concludes the proof. \blacktriangleleft

► **Lemma 8.** *Let t be a trace, $i \in \mathcal{P}$ a process and $P \subseteq \mathcal{P}$ a set of processes. Then we have*

$$\text{res}_{i,P}(t) = \begin{cases} \varepsilon & \text{if } t \models \neg(\mathbb{L}_i \leq \mathbb{L}_i) \vee \bigvee_{j \in P} (\mathbb{L}_i \leq \mathbb{L}_j), \\ \text{res}_{P'}(e) & \text{otherwise,} \end{cases}$$

where $e = \max E_i$ and $P' = \{j \in \mathcal{P} \mid t \models \bigvee_{k \in P} \mathbb{L}_{i,j} \leq \mathbb{L}_k\}$.

Proof. Observe that $\text{res}_{i,P}(t)$ is the empty trace if E_i is empty or if the maximal i -event is below a j -event for some $j \in P$. The first condition is exactly captured by $\neg(\mathbb{L}_i \leq \mathbb{L}_i)$, and the second one by $\bigvee_{j \in P} (\mathbb{L}_i \leq \mathbb{L}_j)$.

Let us now assume that $\text{res}_{i,P}(t)$ is not the empty trace. Then $e = \max E_i$ exists. Moreover, by definition, $\text{res}_{i,P}(t) = \downarrow e \setminus \downarrow E_P$ and $\text{res}_{P'}(e) = \downarrow e \setminus \bigcup_{j \in P'} \downarrow e_j$ where e_j is the maximal event in $E_j \cap \downarrow e$, if it exists.

Let f be an event in $\text{res}_{i,P}(t)$. Then $f \leq e$. Suppose that $f \leq e_j$ for some $j \in P'$. By definition of P' , there exists $k \in P$ such that $t \models \mathbb{L}_{i,j} \leq \mathbb{L}_k$. Then

$$f \leq e_j = \max(E_j \cap \downarrow e) \leq \max(E_j \cap \downarrow e) \leq \max(E_k).$$

In particular, $f \in \downarrow E_P$, a contradiction since $f \in \text{res}_{i,P}(t) = \downarrow e \setminus \downarrow E_P$. Therefore $\text{res}_{i,P}(t)$ is contained in $\downarrow e \setminus \bigcup_{j \in P'} \downarrow e_j = \text{res}_{P'}(e)$.

Conversely, suppose that $f \in \text{res}_{P'}(e)$. Then, again, $f \leq e$. Suppose that $f \in \downarrow E_P$, i.e., $f \leq e' = \max(E_k)$ for some $k \in P$. If $e' < e$, then $f \leq e' = \max(E_k \cap \downarrow e) = e_k$. Also, $E_k \subseteq \downarrow e$, so $t \models \mathbb{L}_{i,k} \leq \mathbb{L}_k$ and hence $k \in P'$, which is impossible since $f \in \text{res}_{P'}(e)$.

We cannot have $e \leq e'$ either, since $t \not\models (\mathbb{L}_i \leq \mathbb{L}_k)$. Therefore e and e' are concurrent events. Let g be a maximal event in $\uparrow f \cap \downarrow e \cap \downarrow e'$. There exists $j \in \text{loc}(g)$ such that $e_j = g = \max(E_j \cap \downarrow e)$. Again this implies that $j \in P'$ and $f \in \bigcup_{j \in P'} \downarrow e_j$, a contradiction. It follows that $\text{res}_{P'}(e)$ is contained in $\text{res}_{i,P}(e)$, which concludes the proof. \blacktriangleleft

A Kleene Theorem for Higher-Dimensional Automata

Uli Fahrenberg

EPITA Research and Development Laboratory (LRDE), Paris, France

Christian Johansen

NTNU Gjøvik, Norway

Georg Struth

University of Sheffield, UK

Collegium de Lyon, France

Krzysztof Ziemiański

University of Warsaw, Poland

Abstract

We prove a Kleene theorem for higher-dimensional automata (HDAs). It states that the languages they recognise are precisely the rational subsumption-closed sets of interval pomsets. The rational operations include a gluing composition, for which we equip pomsets with interfaces. For our proof, we introduce HDAs with interfaces as presheaves over labelled precube categories and use tools inspired by algebraic topology, such as cylinders and (co)fibrations. HDAs are a general model of non-interleaving concurrency, which subsumes many other models in this field. Interval orders are used as models for concurrent or distributed systems where events extend in time. Our tools and techniques may therefore yield templates for Kleene theorems in various models and applications.

2012 ACM Subject Classification Theory of computation → Automata extensions

Keywords and phrases higher-dimensional automata, interval posets, Kleene theorem, concurrency theory, labelled precube categories

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.29

Related Version *Full Version:* <https://arxiv.org/abs/2202.03791>

1 Introduction

Higher-dimensional automata (HDAs) were introduced by Pratt and van Glabbeek as a general geometric model for non-interleaving concurrency [21,23]. HDAs support autoconcurrency and events with duration or structure, whereas events in interleaving models must be instantaneous. They subsume, for example, event structures and safe Petri nets [24]. Asynchronous transition systems and standard automata are two- and one-dimensional HDAs, respectively [12]. We have recently used van Glabbeek’s (execution) paths [24] to relate HDAs with certain languages of interval posets [6]. Yet a precise description of these languages in terms of a Kleene theorem has so far been missing. Our main contribution is the formalisation and proof of such a theorem.

HDAs consist of cells and lists of events that are active in them. Zero-dimensional cells represent states in which no event is active; 1-dimensional cells represent transitions in which exactly one event is active – as in standard automata. Higher n -dimensional cells model concurrent behaviours with n active events. As an example, Fig. 1 shows an HDA with cells of dimension ≤ 2 . The cells x and y , for instance, have active events $[a/b]$ and $[a/c]$, respectively. Cells at any dimension may serve as start and accept cells. In Fig. 1, these are marked with incoming and outgoing arrows. Lower dimensional cells or faces are attached to higher dimensional ones using lower and upper face maps. These indicate further when individual



© Uli Fahrenberg, Christian Johansen, Georg Struth, and Krzysztof Ziemiański;
licensed under Creative Commons License CC-BY 4.0

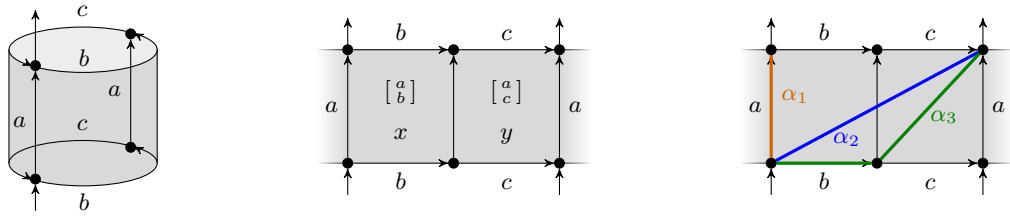
33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Sławomir Lasota, and Anca Muscholl; Article No. 29; pp. 29:1–29:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** HDA with two 2-dimensional cells x and y connected along transitions and modelling parallel execution of a and $(bc)^*$. Middle: unfolded view; right: three accepting paths.

events start or terminate. In Fig. 1, the lower face $\delta_a^0(x)$ of x is the lower b -transition in which a is not yet active; its upper face $\delta_a^1(x)$ is the upper b -transition in which a is no longer active. Further, $\delta_b^1(x) = \delta_c^0(y)$ and $\delta_c^1(y) = \delta_b^0(x)$.

Executions of HDAs are (higher-dimensional) paths [24]: sequences of cells connected with operations of starting and terminating events. Every path α is characterised by ordering the events $\text{ev}(\alpha)$ that occur in it with respect to precedence. This always yields interval orders. In addition, $\text{ev}(\alpha)$ is equipped with source and target interfaces, which model events active in the initial and final cell of α , respectively, and a secondary event order, which captures the list structure of events in cells. We call (isomorphism classes of) such labelled posets with interfaces and an event order *ipomsets*. The language of an HDA is then related to the set of (interval) ipomsets associated with all its accepting paths – from start to accept cells [6]. Languages of HDAs must in particular be down-closed with respect to less concurrent executions, modelled by a subsumption preorder. This motivates the definition of (interval ipomset) languages as subsumption-closed sets of interval ipomsets.

Kleene theorems usually require a notion of *rational* language. Here it is based on the union \cup , gluing (serial) composition $*$, parallel composition \parallel , and (serial) Kleene plus $^+$ of languages. These definitions are not entirely straightforward, as down-closure and the interval property must be preserved. In particular, $*$ is more complicated than, for instance, the standard series composition of pomsets due to interfaces. Without interfaces, it would reduce to the latter. We consider finite HDAs only and thus can neither include the parallel Kleene star nor the full serial Kleene star as a rational operation. The latter contains the identity language, which requires an HDA of infinite dimension.

Our Kleene theorem shows that the rational languages are precisely the regular languages (recognised by finite HDAs). To show that regular languages are rational, we translate the cells of an HDA into a standard automaton and reuse one direction of the standard Kleene theorem. Proving that rational languages are regular is harder. Regularity of \cup is straightforward, and for \parallel , the corresponding operation on HDAs is simply a tensor product. But $*$ and $^+$ require an intricate gluing operation on HDAs along higher-dimensional cells.

Beyond the Kleene theorem, three further contributions seem of independent interest. We model HDAs as presheaves on novel precube categories that feature events and labels in the base category. These are equivalent to standard HDAs [24], but constructions become simpler and the relation between iposets and precubical sets clearer.

We also introduce iHDAs – HDAs with interfaces – which may assign events to source or target interfaces. Target events cannot be terminated: they either remain active at the end of an execution or do not appear at all. By opposition, source events cannot be “unstarted”: they are either active at the beginning of an execution or do not appear at all. Additionally, all events of start cells are source events and all events of accept cells, target events. Every

HDA can be converted into an equivalent iHDA (recognising the same language) and vice versa, using operations of resolution and closure. Both models play a technical role in our proofs, and we frequently switch between them.

Another tool used in the Kleene theorem is motivated by algebraic topology. We introduce cylinder objects and show that every map between (i)HDAs can be decomposed into an (initial or final) inclusion followed by a (future or past) path-lifting map. This allows us to “pull apart” start and accept cells of iHDAs when dealing with serial compositions and loops.

In this paper we introduce the concepts needed for formulating and proving the Kleene theorem, and we outline its proof. Most technical details can be found in the long version [7].

2 Higher-Dimensional Automata

HDAs and iHDAs are particular (pre)cubical sets. Like simplicial sets, they are typically modelled as presheaves on certain base categories. Here we introduce new labelled precube categories and variants with interfaces, which tame the combinatorics of concurrent events in well-structured ways. Their objects are lo-sets, which model concurrent events that are active in some cell of an HDA, or ilo-sets, which equip lo-sets with interfaces. Their morphisms are coface maps (opposites of face maps), which insert events into lo-sets and preserve interfaces if present. Some constructions require isomorphism classes of labelled precube categories (with interfaces), and we define these as well. Finally, we briefly introduce resolution and closure functors that translate between HDAs and iHDAs, in preparation for the summary of our Kleene theorem in Section 5. We contextualise our approach in App. A. Throughout the paper, we fix an alphabet Σ , finite or infinite.

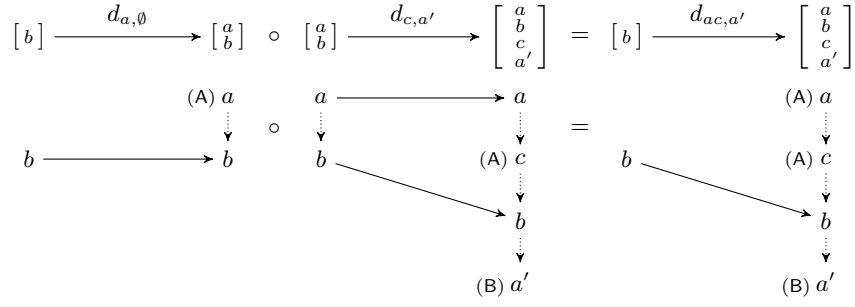
Lo-sets and ilo-sets. A *lo-set* $(U, \dashrightarrow, \lambda)$ is a finite set U totally ordered by the strict order \dashrightarrow and labelled by $\lambda : U \rightarrow \Sigma$. A *lo-set with interfaces* (*ilo-set*) is a triple (S, U, T) of a lo-set U , a *source interface* $S \subseteq U$ and a *target interface* $T \subseteq U$. Lo-sets are regarded as ilo-sets with empty interfaces. We write ${}_S U_T$ or just U for ilo-sets.

The labels of ilo-sets indicate the actions associated with events. The *event order* \dashrightarrow captures their list structure, which is convenient for regarding ilo-sets as ipomsets in Section 3.

A *lo-map* is an order and label preserving function between lo-sets. Lo-maps are strict order embeddings and thus injective, as \dashrightarrow is total. A *lo-isomorphism* is therefore a surjective lo-map. An *ilo-map* $f : U \rightarrow V$ must also satisfy $S_U = f^{-1}(S_V)$ and $T_U = f^{-1}(T_V)$ (and $f(S_U) = S_V$ and $f(T_U) = T_V$ if it is an ilo-isomorphism). We write $U \simeq V$ if (i)lo-sets U and V are isomorphic – there is at most one isomorphism between (i)lo-sets. Isomorphism classes of lo-sets are words over Σ . Those of ilo-sets are words over an extended alphabet $\Sigma_\bullet = \{a, \bullet a, a \bullet, \bullet a \bullet \mid a \in \Sigma\}$, where $\bullet a$, for example, indicates membership in a source interface. As in Fig. 1, we represent such words as column vectors.

Each (i)lo-map $f : U \hookrightarrow V$ defines and is defined by a unique $A = V \setminus f(U) \subseteq V$. We write ∂_A for this map; it is an isomorphism iff $A = \emptyset$. The composite of (i)lo-maps $\partial_A : U \hookrightarrow V$, $\partial_B : V \hookrightarrow W$ is $\partial_{\partial_B(A) \cup B} : U \hookrightarrow W$. This data defines categories of (i)lo-sets. Each $\partial_A : U \rightarrow V$ constructs V by inserting the events of A into U , while the face maps in the introduction delete events and thus map in the opposite direction, see Fig. 2. Linking (i)lo-maps with face maps and initial and final cells of HDAs requires refinements.

Labelled precube categories. Labelled precube categories account for the active, unstarted and terminated events in HDAs and equip them with interfaces. The arrows of labelled precube categories are coface maps. These map in the opposite direction of face maps.



■ **Figure 2** Composition of morphisms in \square . Letters (A) and (B) indicate events that become unstarted (A) or terminated (B), as defined in the triple $(\partial_{A \cup B}, A, B)$.

The *full labelled precube category with interfaces*, $\mathbf{I}\square$, has ilo-sets as objects and *coface maps* $d_{A,B} : U \rightarrow V$ as arrows. Each $d_{A,B}$ is a triple $(\partial_{A \cup B}, A, B)$ with $A, B \subseteq V$ disjoint, $A \cap S_V = \emptyset = B \cap T_V$, and $\partial_{A \cup B} : U \hookrightarrow V$ an ilo-map. The composite of $d_{A,B} : U \rightarrow V$ and $d_{C,D} : V \rightarrow W$ is defined as $d_{C,D} \circ d_{A,B} = (\partial_{C \cup D} \circ \partial_{A \cup B}, \partial_{C \cup D}(A) \cup C, \partial_{C \cup D}(B) \cup D)$.

As an instance, the *full labelled precube category* \square with lo-sets as objects and coface maps $d_{A,B}$ based on lo-maps $\partial_{A \cup B}$ is trivially isomorphic to the full subcategory of $\mathbf{I}\square$ with empty interfaces S_U, T_U , for each ilo-set U .

We only need coface map compositions with pairwise disjoint $A, B, C, D \subseteq W$ and $V = W \setminus (C \cup D)$. In this case,

$$d_{C,D} \circ d_{A,B} = d_{A \cup C, B \cup D}. \quad (1)$$

See Fig 2 for an example. We write d_A^0 for $d_{A, \emptyset}$ and d_B^1 for $d_{\emptyset, B}$. Intuitively, $d_A^0 : U \rightarrow V$ inserts events in A into U that are unstarted in U , whereas $d_B^1 : U \rightarrow V$ inserts events B into U that are terminated in U . See again Fig. 2 for an example. With d_B^1 , events in T_V cannot terminate in U as $T_V \cap B = \emptyset$; with d_A^0 , those in S_V cannot be unstarted in U as $S_V \cap A = \emptyset$. Both interfaces are preserved in pre-images of $\partial_{A \cup B}$ and thus by coface map compositions.

It is standard to work with equivalence classes of events. The *labelled precube category with interfaces* $\mathbf{I}\square$ is the quotient of $\mathbf{I}\square$ with respect to \simeq . Its objects are words over Σ_\bullet . Its coface maps are equivalence classes of coface maps in $\mathbf{I}\square$, where $d_{A,B} : U \rightarrow V$, $d_{A',B'} : U' \rightarrow V'$ are equivalent in $\mathbf{I}\square$ if $U \simeq U'$ and $V \simeq V'$ (hence also $A \simeq A'$ and $B \simeq B'$). They insert letters into words, while remembering whether they correspond to unstarted or terminated actions.

The *labelled precube category* \square is obtained from $\mathbf{I}\square$ in a similar way. The categories $\mathbf{I}\square$ and \square are skeletal, and because isomorphisms are unique, the quotient functors $\mathbf{I}\square \rightarrow \mathbf{I}\square$ and $\square \rightarrow \square$ are equivalences of categories. We thus switch freely between full categories and their skeletons and identify morphisms $[U] \rightarrow [V]$ with representatives $d_{A,B} : U \rightarrow V$.

We often use the involutive *reversal* functor on these categories. It maps ${}_S U_T$ to ${}_T U_S$ and $d_{A,B}$ to $d_{B,A}$, thus swapping unstarted and terminated events.

Higher-dimensional automata. A *precubical set with interfaces* (*ipc-set*) is a presheaf on $\mathbf{I}\square$, that is, a functor $X : \mathbf{I}\square^{\text{op}} \rightarrow \mathbf{Set}$. We write $X[U]$ for the value of X on object U of $\mathbf{I}\square$. We write $\delta_{A,B} = X[d_{A,B}] : X[U] \rightarrow X[U \setminus (A \cup B)]$ for the *face map* associated to coface map $d_{A,B} : U \setminus (A \cup B) \rightarrow U$. Elements of $X[U]$ are *cells* of X – we often view X as a set of cells.

We write $\text{iev}(x) = {}_S U_T$ and $\text{ev}(x) = U$ if $x \in X[{}_S U_T]$, as well as $\delta_A^0 = X[d_A^0]$ and $\delta_B^1 = X[d_B^1]$. Such face maps attach lower and upper faces to the cells in $X[U]$.

A *precubical set* is defined analogously as a presheaf X on \square . We may view it as an ipc-set X such that $X[_S U_T] = \emptyset$ whenever $S \neq \emptyset$ or $T \neq \emptyset$. A precubical set or ipc-set X is *finite* if it has finitely many cells. The *dimension* of a cell $x \in X[U]$ is $|U|$. The *dimension* of X is the maximal dimension among its cells.

A *higher-dimensional automaton with interfaces (iHDA)* is a finite ipc-set X with subsets X_\perp of *start cells* and X^\top of *accept cells*. These are required to satisfy $S = U$ for all $x \in X_\perp$ with $\text{iev}(x) = {}_S U_T$, and $T = U$ for all $x \in X^\top$ with $\text{iev}(x) = {}_S U_T$. X_\perp and X^\top are not necessarily precubical subsets.

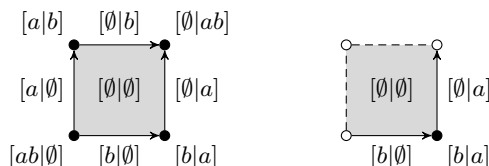
Analogously, a *higher-dimensional automaton (HDA)* is a finite precubical set X equipped with subsets X_\perp and X^\top of start and accept cells. HDAs are not simply special cases of iHDAs due to the above requirements on interfaces.

An *ipc-map* is a natural transformation $f : X \rightarrow Y$ of ipc-sets X, Y , an *iHDA-map* must preserve start and accept cells as well: $f(X_\perp) \subseteq Y_\perp$ and $f(X^\top) \subseteq Y^\top$. Precubical maps and HDA-maps are defined analogously. We write $\square\mathbf{Set}$, $\mathbb{I}\square\mathbf{Set}$, \mathbf{HDA} and \mathbf{iHDA} for the resulting categories (of precubical sets, ipc-sets, HDAs and iHDAs, respectively).

The reversal on \square translates to ipc-sets, precubical sets, iHDAs and HDAs. It maps $\delta_{A,B}$ to $\delta_{B,A}$ and exchanges start and accept cells if present. The relationship between HDAs and iHDAs is studied in [7].

Standard cubes. The *standard U -cube* \square^U of lo-set U is the precubical set represented by U (the Yoneda embedding): $\square^U = \square(-, U)$. For each lo-set V , $\square^U[V]$ is the set of all $d_{A,B} : V \rightarrow U$ with $A, B \subseteq U$. We write $[A|B]$ for such a cell. For $d_{A,B} : U \rightarrow V$ we denote by $\square^{d_{A,B}} : \square^U \rightarrow \square^V$ the induced map given by $\square^{d_{A,B}}([C|D]) = [A \cup C | B \cup D]$. The definition of $\mathbb{I}\square^U$ for an ilo-set U is analogous.

► **Example 1.** For $U = \begin{bmatrix} a \\ b \end{bmatrix}$, \square^U has the cells occurring as pairs $[-|-]$ in the left-hand cube below. The first coordinate of the pair lists the unstarted events associated to a cell, the second one the terminated ones.



The maps $\square^{d_{A,B}}$ attach faces to cells. The lower face map $\square^{d_a^0}$, for instance, attaches $[a|\emptyset]$ as the left 1-cell to the 2-cell $[\emptyset|\emptyset]$ and $[ab|\emptyset]$ as the left 0-cell to the 1-cell $[b|\emptyset]$. Notation $[a|\emptyset]$ indicates that a has not yet started and no element has terminated in the associated face, while b is active. An analogous construction of the standard cube $\mathbb{I}\square^V$, for $V = \begin{bmatrix} \bullet a \\ b \bullet \end{bmatrix}$, is shown in the right-hand diagram above. Here $\bullet a$ and $b \bullet$ prevent a from starting and b from terminating on faces. All lower faces in the left-hand cube containing a on the left of the $|$ and all upper faces containing b on the right of the $|$ must thus be removed. (We have omitted some set braces and likewise.)

The notation $[A|B]$ for the cells of \square^U is useful in later proofs. For any (i)HDA X and $x \in X$, there exists a unique precubical map, the Yoneda embedding $\iota_x : \square^U \rightarrow X$ ($\mathbb{I}\square^U \rightarrow X$), such that $\iota_x([\emptyset|\emptyset]) = x$.

Standard automata. One-dimensional HDAs X are slight generalisations of standard finite automata. Cells in $X[\emptyset]$ are states of the automaton, those in $X[a]$ are a -labelled transitions. The face maps $\delta_a^0, \delta_a^1 : X[a] \rightarrow X[\emptyset]$ attach sources and targets to transitions. Yet transitions may serve as start and accept cells in X .

Resolution and closure. The forgetful functor $F : \square \rightarrow \square$, $sU_T \mapsto U$ induces two functors $\text{Res} : \square \mathbf{Set} \rightarrow \square \mathbf{Set}$ and $\text{Cl} : \square \mathbf{Set} \rightarrow \square \mathbf{Set}$. *Resolution* Res is given by $\text{Res}(X) = X \circ F$: the free functor induced by F , and *closure* Cl is left adjoint to Res . We extend them to functors between **HDA** and **iHDA** in a natural way.

Intuitively, closure fills in missing cells of ipc-sets: the standard cube \square^U of Example 1, for instance, is the closure of \square^V . The cells of $\text{Res}(X)$ are triples (x, S, T) , where x is a cell of X and the subsets $S, T \subseteq \text{ev}(x)$ are all possible assignments of interfaces. Every cell $x \in X[U]$ thus produces $4^{|U|}$ cells in $\text{Res}(X)$, for example,

$$\text{Res} \left(\begin{array}{c} \bullet \\ \nearrow \\ \bullet \end{array} \right) = \begin{array}{c} \circ \longrightarrow \bullet \\ \bullet \longrightarrow \circ \\ \circ \longrightarrow \circ \end{array}$$

We give a detailed description of Res and Cl in [7].

3 Ipomsets

Pomsets are a standard model of non-interleaving concurrency. Ipomsets have been introduced to model the behaviours of a restricted class of HDAs [6]. Here we recall the basic definitions and adapt them to general HDAs. A notion of *rational ipomset language* is related with HDAs in the Kleene theorem of Section 5.

Ipomsets. A *labelled iposet* $(P, <, \dashrightarrow, S, T, \lambda)$ consists of a finite set P with two strict (partial) orders: the *precedence order* $<$ and the *event order* \dashrightarrow such that each pair in P is comparable by either \leq or \dashrightarrow . $\lambda : P \rightarrow \Sigma$ is a labelling function, and $S, T \subseteq P$ are the source and target *interfaces* of P . Elements of S must be $<$ -minimal and those of T $<$ -maximal, hence S and T are lo-sets. We write ε for the empty iposet.

A *subsumption* of labelled iposets P, Q is a bijection $f : P \rightarrow Q$ for which $f(S_P) = S_Q$, $f(T_P) = T_Q$, $f(x) <_Q f(y)$ implies $x <_P y$, and $x \dashrightarrow_P y$, $x \not<_P y$ and $y \not<_P x$ imply $f(x) \dashrightarrow_Q f(y)$. Subsumptions thus respect interfaces, reflect precedence and preserve essential event order. Our definition adapts the standard one [13] to the presence of event orders; intuitively, P has more order, and less concurrency, than Q .

An *isomorphism* of labelled iposets is a subsumption that is an order isomorphism. The event order makes isomorphisms unique [6, Lem. 34]. We write $P \sqsubseteq Q$ if there is a subsumption $P \rightarrow Q$ (Q subsumes P) and $P \cong Q$ if P and Q are isomorphic. Isomorphic iposets have the same order and label structure. An *ipomset* is an isomorphism class of labelled iposets. We switch freely between ipomsets and labelled iposets, which is safe due to uniqueness of isomorphisms.

An ipomset P is *discrete* if $<$ is empty and hence \dashrightarrow total. It is an *identity* if $S = P = T$. Discrete ipomsets can therefore be identified with isomorphism classes of ilo-sets. The *singleton ipomsets* are the discrete ilo-sets $[a]$, $[\bullet a]$, $[a \bullet]$ and $[\bullet a \bullet]$ for all $a \in \Sigma$. They generate the rational ipomset languages in Section 5.

An ipomset P is an *interval ipomset* if it admits an interval representation [11]: a pair $b, e : P \rightarrow \mathbb{R}$ such that $b(x) \leq e(x)$ for all $x \in P$ and $x <_P y$ iff $e(x) < b(y)$ for all $x, y \in P$. We write iPoms and iiPoms for the sets of ipomsets and interval ipomsets, respectively.

Ipomset compositions. The *parallel composition* $P \parallel Q$ of labelled iposets P, Q has the disjoint union $P \sqcup Q$ as carrier set, $S_{P \parallel Q} = S_P \cup S_Q$, $T_{P \parallel Q} = T_P \cup T_Q$, $<_{P \parallel Q} = <_P \cup <_Q$, and $x \dashrightarrow_{P \parallel Q} y$ iff $x \dashrightarrow_P y$, $x \dashrightarrow_Q y$, or $x \in P$ and $y \in Q$. It is a straightforward generalisation of the pomset case, a coproduct of P and Q that extends \dashrightarrow_P and \dashrightarrow_Q .

The *gluing composition* $P * Q$ is only defined if $T_P \simeq S_Q$. Its carrier set is the quotient $(P \sqcup Q)_{/x \equiv f(x)}$, where $f : T_P \rightarrow S_Q$ is the unique isomorphism. The interfaces are $S_{P * Q} = S_P$ and $T_{P * Q} = T_Q$, $\dashrightarrow_{P * Q}$ is the transitive closure of $\dashrightarrow_P \cup \dashrightarrow_Q$, and $x <_{P * Q} y$ iff $x <_P y$, $x <_Q y$, or $x \in P \setminus T_P$ and $y \in Q \setminus S_Q$. The structural inclusions $P \hookrightarrow P * Q \hookleftarrow Q$ preserve both precedence and event orders.

For ipomsets with empty interfaces, $*$ is serial pomset composition [13]. In general, matching interface points are glued, see [6,8] for examples. Both $*$ and \parallel respect isomorphisms and lift to associative, non-commutative operations on ipomsets (\parallel is non-commutative due to the event order). Ipomsets form a category with lo-sets as objects, ipomsets as arrows and $*$ as composition. Interval ipomsets are closed under $*$ [6], but not under \parallel ($a \rightarrow b$ is an interval ipomset; $(a \rightarrow b) \parallel (a \rightarrow b)$ is not). Note that all (isomorphism classes of) interval ipomsets can be generated by gluing ilo-sets [6].

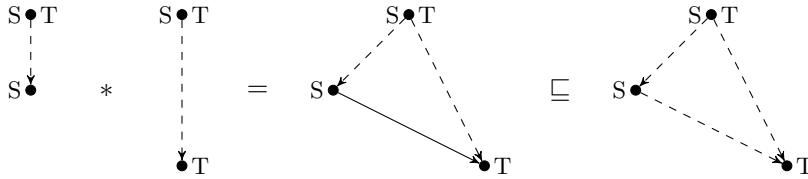
The *width* $w(P)$ of ipomset P is the cardinality of a maximal $<$ -antichain; its *size* is $\#(P) = |P| - \frac{1}{2}(|S| + |T|)$. We glue ipomsets along interfaces below and hence remove half of the interfaces when computing $\#$, which thus may be fractional.

► **Lemma 2.** *Let P and Q be ipomsets. Then*

- (a) $w(P \parallel Q) = w(P) + w(Q)$ and $\#(P \parallel Q) = \#(P) + \#(Q)$,
- (b) if $T_P = S_Q$, then also $w(P * Q) = \max(w(P), w(Q))$ and $\#(P * Q) = \#(P) + \#(Q)$,
- (c) if $P \sqsubseteq Q$, then $w(P) \leq w(Q)$.

► **Lemma 3.** *For lo-sets $W \subseteq V \subseteq U$, ${}_W V_V * {}_V U_U = {}_W U_U$. For lo-sets $Z \subseteq U, V \subseteq W$ with $Z = U \cap V$ and $W = U \cup V$, ${}_U U_Z * {}_Z V_V \sqsubseteq {}_U W_V$. ◀*

The second fact is illustrated by the following picture.



Languages. An *interval ipomset language* (a *language* for short) is a subset $L \subseteq \text{iiPoms}$ that is *down-closed*: if $P \sqsubseteq Q$ and $Q \in L$, then $P \in L$. We introduce the rational operations \cup , $*$, \parallel and (Kleene plus) $^+$ for languages:

$$L * M = \{P * Q \mid P \in L, Q \in M, T_P = S_Q\} \downarrow, \quad L \parallel M = \{P \parallel Q \mid P \in L, Q \in M\} \downarrow,$$

$$L^+ = \bigcup_{n \geq 1} L^n, \quad \text{for } L^1 = L, L^{n+1} = L * L^n.$$

We write $A \downarrow = \{P \in \text{iiPoms} \mid \exists Q \in A. P \sqsubseteq Q\}$ for the *down-closure* of a set $A \subseteq \text{iPoms}$. Down-closure is needed because parallel compositions of interval ipomsets may not be interval and gluing and parallel compositions of down-closed languages may not be down-closed.

► **Example 4.** $\{[a] \parallel [b]\} = \{[\begin{smallmatrix} a \\ b \end{smallmatrix}]\} = \{[\begin{smallmatrix} a & \bullet \\ b & \bullet \end{smallmatrix}]\} * \{[\begin{smallmatrix} \bullet & a \\ \bullet & b \end{smallmatrix}]\}$ is not down-closed.

Both $*$ and \parallel are associative and non-commutative. The identity of \parallel is $\{\varepsilon\}$, that of $*$ is the *identity language* $\text{ld} = \{{}_U U_U \mid U \in \square\}$.

The class of *rational languages* is the smallest class that contains the empty language, the empty-pomset language and the singleton pomset languages

$$\emptyset, \{\varepsilon\}, \{[a]\}, \{[\bullet a]\}, \{[a \bullet]\}, \{[\bullet a \bullet]\}, \quad a \in \Sigma, \quad (2)$$

and that is closed under the rational operations \cup , $*$, \parallel and $^+$.

The *width* of a language is the maximal width among its elements. Lemma 2 shows that rational languages have finite width; ld has infinite width and is thus not rational.

4 Paths and Languages of iHDAs

Computations or runs of (i)HDAs are higher-dimensional paths that keep track of the cells and face maps traversed [24]. In this section we recall their definition. As an important stepping stone towards a Kleene theorem we then relate paths of (i)HDAs with ipomsets – for a more general class than [6]. We also introduce notions of path equivalence and subsumption. The latter corresponds to ipomset subsumption. Finally, we introduce regular languages.

Paths. A *path* of length n in $X \in \square \mathbf{Set}$ is a sequence

$$\alpha = (x_0, \varphi_1, x_1, \varphi_2, \dots, \varphi_n, x_n),$$

where the $x_k \in X[U_k]$ are cells and, for all k , either

- $\varphi_k = d_A^0 \in \square(U_{k-1}, U_k)$, $A \subseteq U_k$ and $x_{k-1} = \delta_A^0(x_k)$ (up-step), or
- $\varphi_k = d_B^1 \in \square(U_k, U_{k-1})$, $B \subseteq U_{k-1}$, $\delta_B^1(x_{k-1}) = x_k$ (down-step).

We write $x_{k-1} \nearrow^A x_k$ for the up-steps and $x_{k-1} \searrow_B x_k$ for the down-steps in α , generally assuming $A \neq \emptyset \neq B$. We refer to the up- or down-steps (paths $(x_i, \varphi_{i+1}, x_{i+1})$, $0 \leq i < n$) as *steps* in α and write \mathbf{P}_X for the set of all paths on X .

► **Example 5.** Figure 1 (on the right) in the introduction depicts the three paths

$$\begin{aligned} \alpha_1 &= (\delta_{ab}^0(x) \nearrow^a \delta_b^0(x) \searrow_a \delta_{ac}^1(y)), \\ \alpha_2 &= (\delta_{ab}^0(x) \nearrow^{ab} x \searrow_b \delta_b^1(x) \nearrow^c y \searrow_{ac} \delta_{ac}^1(y)), \\ \alpha_3 &= (\delta_{ab}^0(x) \nearrow^b \delta_a^0(x) \searrow_b \delta_{ac}^0(y) \nearrow^{ac} y \searrow_{ac} \delta_{ac}^1(y)). \end{aligned}$$

We equip paths with source and target maps as usual: $\ell(\alpha) = x_0$ and $r(\alpha) = x_n$ for path α as above. For $x, y \in X$, we write $\mathbf{P}_X(x, y) = \{\alpha \in \mathbf{P}_X \mid \ell(\alpha) = x, r(\alpha) = y\}$. Any ipc-map $f : X \rightarrow Y$ induces a map $f : \mathbf{P}_X \rightarrow \mathbf{P}_Y$. For α as above it is $f(\alpha) = (f(x_0), \varphi_1, f(x_1), \varphi_2, \dots, \varphi_n, f(x_n))$. For paths $\alpha = (x_0, \varphi_1, \dots, x_n)$, $\beta = (y_0, \psi_1, \dots, y_m)$ with $r(\alpha) = \ell(\beta)$ the *concatenation* $\alpha * \beta = (x_0, \varphi_1, \dots, x_n, \psi_1, \dots, y_m)$ is defined as expected.

Cell $y \in X$ is *reachable* from cell $x \in X$, denoted $x \preceq y$, if $\mathbf{P}_X(x, y) \neq \emptyset$. This preorder is generated by $\delta_A^0(x) \preceq x \preceq \delta_B^1(x)$ for $x \in X$ and $A, B \subseteq \text{ev}(x)$. We call X *acyclic* if \preceq is a partial order. The reversal on ipc-sets reverses paths and \preceq .

From paths to ipomsets. Next we introduce a map ev that computes ipomsets of paths. The interval ipomset $\text{ev}(\alpha)$ of path $\alpha \in \mathbf{P}_X$ is computed recursively:

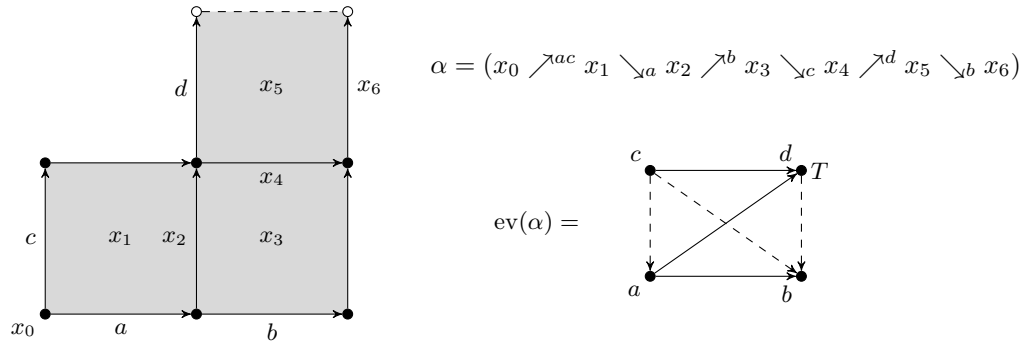
- If $\alpha = (x)$ has length 0, then $\text{ev}(\alpha) = \text{ev}(x) \text{ev}(x)_{\text{ev}(x)}$.
- If $\alpha = (y \nearrow^A x)$, then $\text{ev}(\alpha) = \text{ev}(x) \setminus_A \text{ev}(x)_{\text{ev}(x)}$.
- If $\alpha = (x \searrow_B y)$, then $\text{ev}(\alpha) = \text{ev}(x) \text{ev}(x)_{\text{ev}(x) \setminus B}$.
- If $\alpha = \beta_1 * \dots * \beta_n$ is a concatenation of steps β_i , then $\text{ev}(\alpha) = \text{ev}(\beta_1) * \dots * \text{ev}(\beta_n)$.

Interfaces and gluings of ipomsets are essential for this construction. It would not have worked with regular pomsets.

► **Example 6.** The ipomset of path α_1 in Example 5 is computed recursively as $\text{ev}(\alpha_1) = \text{ev}(\delta_{ab}^0(x) \nearrow^a \delta_b^0(x)) * \text{ev}(\delta_b^0(x) \searrow_a \delta_{ac}^1(y)) = \emptyset a_a * a_a \emptyset = a$. Those of the other two paths are $\text{ev}(\alpha_2) = a \parallel (b \rightarrow c)$ and $\text{ev}(\alpha_3) = b * [a]$.

The following fact is immediate from the definition of ev .

► **Lemma 7.** For $\alpha, \beta \in \mathbf{P}_X$ with $r(\alpha) = \ell(\beta)$ and an ipc-map $f : X \rightarrow Y$, $\text{ev}(\alpha * \beta) = \text{ev}(\alpha) * \text{ev}(\beta)$ and $\text{ev}(f(\alpha)) = \text{ev}(\alpha)$. ◀



■ **Figure 3** Example of a path α in an iHDA and its interval ipomset $\text{ev}(\alpha)$. Solid arrows show the precedence on $\text{ev}(\alpha)$, dashed arrows the event order. The start interface is empty, the target interface contains the single event d .

Path equivalence and subsumption. *Path equivalence* is the congruence \simeq on P_X generated by $(z \nearrow^A y \nearrow^B x) \simeq (z \nearrow^{A \cup B} x)$, $(x \searrow_A y \searrow_B z) \simeq (x \searrow_{A \cup B} z)$ and $\gamma * \alpha * \delta \simeq \gamma * \beta * \delta$ whenever $\alpha \simeq \beta$. Further, *path subsumption* is the transitive relation \sqsubseteq on P_X generated by $(y \searrow_A w \nearrow^B z) \sqsubseteq (y \nearrow^B x \searrow_A z)$, for disjoint $A, B \subseteq \text{ev}(x)$, $\gamma * \alpha * \delta \sqsubseteq \gamma * \beta * \delta$ whenever $\alpha \sqsubseteq \beta$, and $\alpha \sqsubseteq \beta$ whenever $\alpha \simeq \beta$. We say that β *subsumes* α if $\alpha \sqsubseteq \beta$.

This means that β is more concurrent than α . Both relations preserve sources and targets of paths; they translate to ipomsets as follows (the proof uses Lemmas 3 and 7).

▶ **Lemma 8.** *If $\alpha, \beta \in P_X$, then $\alpha \sqsubseteq \beta \Rightarrow \text{ev}(\alpha) \sqsubseteq \text{ev}(\beta)$ and $\alpha \simeq \beta \Rightarrow \text{ev}(\alpha) = \text{ev}(\beta)$.* ◀

▶ **Example 9.** It is easy to check that path α_3 in Example 5 is subsumed by α_2 , and so are the corresponding pomsets in Example 6: $\text{ev}(\alpha_3) = b * \begin{bmatrix} a \\ c \end{bmatrix} \sqsubseteq a \parallel (b \rightarrow c) = \text{ev}(\alpha_2)$.

Regular languages. A path $\alpha \in P_X$ of an (i)HDA X is *accepting* if $\ell(\alpha) \in X_\perp$ and $r(\alpha) \in X^\top$. Let $L(X) = \{\text{ev}(\alpha) \mid \alpha \in P_X \text{ is accepting}\}$ be the set of ipomsets recognised by X .

▶ **Proposition 10.** *$L(X)$ is a language (a down-closed set of interval ipomsets).*

▶ **Proposition 11.** *HDA and iHDA recognise the same languages.*

A language is *regular* if it is recognised by an HDA (or an iHDA).

Prop. 10 extends [6, Thm. 95]; see [7] for a proof. The proof of Prop. 11 in [7] uses resolution and closure.

Lem. 2 implies the following.

▶ **Lemma 12.** *Regular languages have finite width.* ◀

An (i)HDA map $f : X \rightarrow Y$ is a *weak equivalence* if for every accepting path $\beta \in P_Y$ there exists an accepting path $\alpha \in P_X$ with $f(\alpha) = \beta$. The next lemma is shown in [7].

▶ **Lemma 13.** *If $f : X \rightarrow Y$ is an (i)HDA-map, then $L(X) \subseteq L(Y)$. If f is a weak equivalence, then $L(X) = L(Y)$.*

5 Kleene Theorem

In this section we state the Kleene theorem for HDAs and ipomset languages and provide a road-map towards its proof. Technical details are explained in the rest of the paper.

► **Theorem 14** (Kleene theorem). *A language is regular if and only if it is rational.*

Proof. By Prop. 16 and Cor. 29 below. ◀

Regular languages are rational. This follows from a translation to the standard automata-theoretic Kleene theorem. It uses the following property which is proved in Section 6 after introducing cylinders.

► **Proposition 15.** *If L is regular, then so is $L \setminus \text{Id}$.*

► **Proposition 16.** *Regular languages are rational.*

Proof. Suppose $L = (L \cap \text{Id}) \cup (L \setminus \text{Id})$ is regular. First, $L \cap \text{Id}$ is a finite set of identity ipomsets and thus rational. Second, $L \setminus \text{Id}$ is regular by Prop. 15; we show that it is rational. Let X be an HDA that recognises $L \setminus \text{Id}$. Then $X_{\perp} \cap X^{\top} = \emptyset$ (otherwise, X accepts an identity ipomset). Let G be an automaton with alphabet $\mathbb{I}\square$ whose states are cells of X and whose transitions are, for all $x \in X[U]$, $A \subseteq U \in \square$,

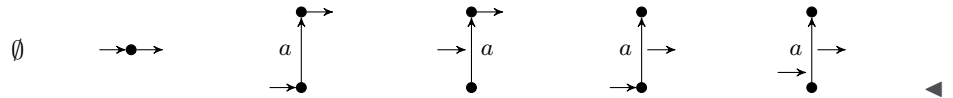
- $\delta_A^0(x) \rightarrow x$ labelled with $(U \setminus A)U$
- $x \rightarrow \delta_A^1(x)$ labelled with $U(U \setminus A)$.

Start and accept states of G are start and accept cells of X . Accepting runs of G are then exactly accepting paths of X . Every such run of G contains at least one transition. Hence, if $U_1 U_2 \cdots U_n \in \mathbb{I}\square^*$ is the word of a run of G , then $U_1 * U_2 * \cdots * U_n$ is the interval ipomset of the corresponding path in X . By the standard Kleene theorem, $L(G)$ is represented by a regular expression $w(U_1, \dots, U_n)$ with operations \cup , $*$ and $(-)^+$ for $U_i \in \mathbb{I}\square$. But each $U_i = e_i^1 \parallel \cdots \parallel e_i^{k(i)}$, a parallel composition of singleton ipomsets. Thus $L(X)$ is represented by $w(e_1^1 \parallel \cdots \parallel e_1^{k(1)}, \dots, e_n^1 \parallel \cdots \parallel e_n^{k(n)})$ and therefore rational. ◀

Rational languages are regular. We need to show, as usual, that the generating languages are regular and that the rational operations preserve regularity.

► **Proposition 17.** *All languages in (2) are regular.*

Proof. The languages in (2) are recognised by the HDAs



► **Proposition 18.** *Unions of regular languages are regular.*

Proof. $L(X \sqcup Y) = L(X) \cup L(Y)$, where $X \sqcup Y$ is coproduct. ◀

In order to show the same for parallel compositions of regular languages, we introduce *tensor products* of HDAs. For HDAs X and Y , $X \otimes Y$ is the HDA defined, for $U, V, W \in \square$, $x \in X[V]$, $y \in Y[W]$, $A, B \subseteq U$, by

$$(X \otimes Y)[U] = \bigcup_{V \parallel W = U} X[V] \times Y[W], \quad \delta_{A,B}(x, y) = (\delta_{A \cap V, B \cap V}(x), \delta_{A \cap W, B \cap W}(y)),$$

and $(X \otimes Y)_{\perp} = X_{\perp} \times Y_{\perp}$, $(X \otimes Y)^{\top} = X^{\top} \times Y^{\top}$.

► **Proposition 19.** $\mathsf{L}(X \otimes Y) = \mathsf{L}(X) \parallel \mathsf{L}(Y)$ for all HDAs X, Y . As a consequence, parallel compositions of regular languages are regular.

A proof for a restricted class of HDAs is in [6]; we show a complete proof in [7].

Analogous proofs for $*$ and $+$ are much harder. They need gluing operations on HDAs and additional machinery.

An ipomset P is *separated* if $P \setminus (S_P \cup T_P) \neq \emptyset$ (some of its elements are not in an interface). A language is *separated* if its elements are separated.

An (i)HDA X is *start simple* if it has exactly one start cell, *accept simple* if it has exactly one accept cell, and *simple* if it is both start and accept simple. A regular language is *simple* if it is recognised by a simple iHDA. Yet this reduces expressivity.

► **Example 20.** The HDA X with a single 0-cell x , a 1-loop labelled a , and $X_\perp = X^\top = \{a\}$ is simple and recognises the language of all ipomsets $[\bullet a \cdots a \bullet]$, but no simple iHDA does.

Next we prove two lemmas about simple and separated languages.

► **Lemma 21.** *Regular languages are unions of simple regular languages.*

Proof. Prop. 11 allows us to work with an iHDA X , say. Denote $X_\perp = \{x_\perp^i\}_{i=1}^m$ and $X^\top = \{x_j^\top\}_{j=1}^n$. For each tuple (i, j) let X_i^j be the iHDA with the same underlying ipc-set as X and $(X_i^j)_\perp = \{x_\perp^i\}$, $(X_i^j)^\top = \{x_j^\top\}$. We have $\mathsf{L}(X) = \bigcup_{i,j} \mathsf{L}(X_i^j)$. ◀

► **Lemma 22.** *For L of finite width with $L \cap \text{Id} = \emptyset$, and n sufficiently large, L^n is separated.*

Proof. For every $Q \in L^n$ there exists $P = P_1 * \dots * P_n$ such that $P_k \in L$ and $Q \sqsubseteq P$. As $\#(P_k) \geq \frac{1}{2}$, additivity of size implies $\#(Q) = \#(P) = \#(P_1) + \dots + \#(P_n) \geq \frac{n}{2}$ and therefore $|S_Q|, |T_Q| \leq \mathsf{w}(Q) \leq \mathsf{w}(P) = \max_k \mathsf{w}(P_k) \leq \mathsf{w}(L)$, since gluing composition does not increase width. Eventually, $|S_Q| + |T_Q| \leq 2\mathsf{w}(L) < n \leq 2\#(Q) = 2|Q| - |S_Q| - |T_Q|$ holds for $n \geq 2\mathsf{w}(L) + 1$ and therefore $|S_Q| + |T_Q| < |Q|$. ◀

Let X, Y be simple HDAs with $X^\top = \{x^\top\}$, $Y_\perp = \{y_\perp\}$, and $\text{ev}(x^\top) = \text{ev}(y_\perp) = U$. The *gluing composition* of X and Y is the HDA

$$X * Y = \text{colim} \left(X \xleftarrow{x^\top} \square^U \xrightarrow{y_\perp} Y \right)$$

with $(X * Y)_\perp = X_\perp$, $(X * Y)^\top = Y^\top$. In other words, we identify the accept cell of X with the start cell of Y , as well as their corresponding faces. In general, $\mathsf{L}(X * Y)$ is a strict superset of $\mathsf{L}(X) * \mathsf{L}(Y)$; but in Sect. 6 we will introduce properties of *start* and *accept proper* which ensure the following.

► **Proposition 23.** *Let X, Y be simple iHDAs. If X is accept simple and accept proper, and Y is start simple and start proper, then $\mathsf{L}(\text{Cl}(X) * \text{Cl}(Y)) = \mathsf{L}(X) * \mathsf{L}(Y)$.*

Proof. This is a special case of Prop. 52 which is shown in [7]. ◀

► **Proposition 24.** *Every simple regular language is recognised by a start simple and start proper iHDA as well as by an accept simple and accept proper iHDA.*

The proof can be found in the next section.

► **Proposition 25.** *Gluing compositions of simple regular languages are regular.*

29:12 A Kleene Theorem for Higher-Dimensional Automata

Proof. Let X, Y be simple iHDAs recognising L and M , respectively. We can assume that X is accept simple and accept proper and Y is start simple and start proper (Prop. 24). Thus, by Prop. 23, $L(\text{Cl}(X) * \text{Cl}(Y)) = L(X) * L(Y) = L * M$. ◀

► **Proposition 26.** *The Kleene plus of a separated regular language is regular.*

The proof is in [7]. Below we show that the additional assumptions above may be removed, finishing the proof of the Kleene theorem.

► **Proposition 27.** *Gluing compositions of regular languages are regular.*

Proof. Suppose L and M are regular. By Lem. 21, $L = \bigcup_i L_i$ and $M = \bigcup_j M_j$ for simple regular languages L_i and M_j . Then $L * M = (\bigcup_i L_i) * (\bigcup_j M_j) = \bigcup_i \bigcup_j L_i * M_j$ is regular by Propositions 18 and 25. ◀

► **Proposition 28.** *The Kleene plus of a regular language is regular.*

Proof. Suppose L is regular. If $L \cap \text{Id} = \emptyset$, then L^n is separated for sufficiently large n by Lem. 22. In this case, $L^+ = \bigcup_{i=1}^n L^i \cup (\bigcup_{i=1}^n L^i) * (L^n)^+$ is regular by Propositions 18, 27 and 26. Otherwise, if $L \cap \text{Id} \neq \emptyset$, then $L^+ = ((L \cap \text{Id}) \cup (L \setminus \text{Id}))^+ = (L \cap \text{Id}) \cup (L \setminus \text{Id})^+$ is regular by Prop. 15 and Prop. 18. ◀

► **Corollary 29.** *Every rational language is regular.*

Proof. By Propositions 17, 18, 19, 27, and 28 ◀

The remaining proofs (Prop. 15, 25, 26). Prop. 15 needs translations between HDAs and iHDAs via resolution and closure, established in the next section.

Next we briefly explain the proof of Prop. 25. Sequential compositions $X * Y$ of standard automata require some care. When accept states of X have outgoing transitions or start states of Y have incoming ones, one cannot simply identify accept states of X with start states of Y . The language of the resulting automaton may contain more words than $L(X) * L(Y)$. To alleviate this, one usually replaces X and Y by equivalent “proper” automata without such transitions. We proceed along similar lines for iHDA. Suppose X, Y are simple iHDA with $X^\top = \{x^\top\}$, $Y_\perp = \{y_\perp\}$ and $\text{ev}(x^\top) = \text{ev}(y_\perp)$ ($\text{ev}(x^\top) \neq \text{ev}(y_\perp)$ implies $L(X) * L(Y) = \emptyset$).

We show that any iHDA may be converted into a *start proper* or *target proper* iHDA that recognises the same language. Start-properness implies that any start cell is \preceq -minimal and that different start cells do not share any faces. Target-properness is defined similarly.

We introduce *cylinders* in Sec 6 to perform this conversion. For ipc-sets X, Y, Z and image-disjoint maps $f : Y \rightarrow X$, $g : Z \rightarrow X$, we construct an ipc-set $C(f, g)$ together with maps $\tilde{f} : Y \rightarrow C(f, g)$, $\tilde{g} : Z \rightarrow C(f, g)$ and $p : C(f, g) \rightarrow X$, see Fig. 4. The image of \tilde{f} is \preceq -minimal, that of \tilde{g} is \preceq -maximal, and $p \circ \tilde{f} = f$ and $p \circ \tilde{g} = g$. This construction is inspired by algebraic topology: \tilde{f} and \tilde{g} are reminiscent of cofibrations and p of a trivial fibration. Here we only show that p has suitable lifting properties. We also use cylinders for Prop. 15.

We may therefore assume that X is accept proper and Y start proper. Gluing iHDAs is intricate due to the missing faces of start and accept cells. So, after rearranging iHDAs, we convert them back into HDAs using closure. Then we show that the gluing of $\text{Cl}(X)$ and $\text{Cl}(Y)$ along their accept and start cells yields an HDA that recognises $L(X) * L(Y)$.

Finally, the proof of Prop. 26 is similar but more sophisticated. Cylinders allow us to separate start cells from each other (same for accept cells), but we are not able to separate start cells from accept cells. Thus, we require the separability assumption.

The tools needed for gluing closures of proper iHDAs are developed in [7].

6 Cylinders

In the final technical section of this paper we describe the cylinder construction mentioned above, as it may be of independent interest. Omitted proofs are shown in [7].

Initial and final inclusions. A sub-ipc-set $Y \subseteq X$ is *initial* if it is down-closed with respect to the reachability preorder \preceq . Equivalently, $\delta_B^1(x) \in Y$ implies $x \in Y$ for all $x \in X[U]$ and $B \subseteq U \setminus T_U$. By reversal, Y is *final* if it is up-closed with respect to \preceq or, equivalently, $\delta_A^0(x) \in Y$ implies $x \in Y$. An *initial (final) inclusion* is an injective ipc-map whose image is an initial (final) sub-ipc-set.

► **Lemma 30.** *If $f : Y \rightarrow X$ is an initial or final inclusion, then so is $\text{Cl}(f) : \text{Cl}(Y) \rightarrow \text{Cl}(X)$.*

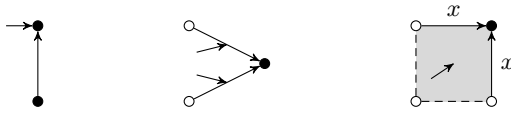
Proper iHDAs. The *start* and *accept maps* of iHDA X are the ipc-maps

$$\iota_{\perp}^X = \prod_{x \in X_{\perp}} \iota_x : \prod_{x \in X_{\perp}} \mathbb{I}\square^{\text{iev}(x)} \rightarrow X, \quad \iota_X^{\top} = \prod_{x \in X^{\top}} \iota_x : \prod_{x \in X^{\top}} \mathbb{I}\square^{\text{iev}(x)} \rightarrow X.$$

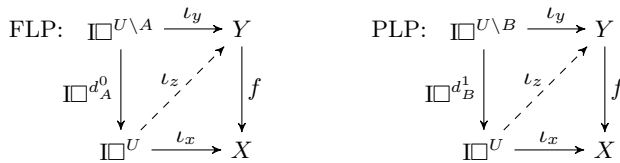
We call an iHDA *start proper* if its start map is an initial inclusion, *accept proper* if its accept map is a final inclusion, and *proper* if it is start proper, accept proper and the images of the start map and the accept map are disjoint.

► **Lemma 31.** *All start cells of start proper iHDAs are \preceq -minimal; all accept cells of accept proper iHDAs are \preceq -maximal.*

The condition of Lem. 31 is not sufficient for properness. The diagrams show examples of iHDAs that are not start proper; edges marked with x are identified.



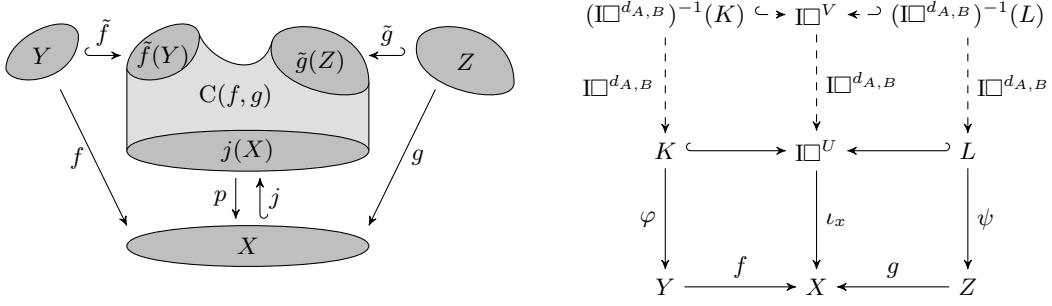
Lifting properties. An ipc-map $f : Y \rightarrow X$ has the *future lifting property (FLP)* if for every up-step $\alpha = (\delta_A^0(x) \nearrow^A x)$ in X and every $y \in Y$ such that $f(y) = \delta_A^0(x)$ there is an up-step $\beta = (y \nearrow^A z)$ in Y such that $f(\beta) = \alpha$. The *past lifting property (PLP)* is defined by reversal. FLP and PLP are equivalent to the lifting property for the following diagrams.



The next lemma is immediate from the definitions.

► **Lemma 32.** *An ipc-map $f : Y \rightarrow X$ has the FLP if and only if, for every $\alpha \in \mathbb{P}_X$ and $y \in f^{-1}(\ell(\alpha))$, there exists a path $\beta \in \mathbb{P}_Y$ such that $\ell(\beta) = y$ and $f(\beta) = \alpha$. An analogous property holds for PLP.* ◀

Let $f : Y \rightarrow X$ be an ipc-map, let $S, T \subseteq X$ (subsets, but not necessarily sub-ipc-sets). Then f has the *total lifting property (TLP)* with respect to S and T if for every path $\alpha \in \mathbb{P}_X$ with $\ell(\alpha) \in S$ and $r(\alpha) \in T$ and every $y \in f^{-1}(\ell(\alpha))$ and $z \in f^{-1}(r(\alpha))$, there exists a path $\beta \in \mathbb{P}_Y(y, z)$ such that $f(\beta) = \alpha$.



■ **Figure 4** The cylinder $C(f, g)$ and a diagram defining its cell.

► **Proposition 33.** *Let $f : Y \rightarrow X$ be an iHDA map such that the functions $Y_{\perp} \rightarrow X_{\perp}$ and $Y^{\top} \rightarrow X^{\top}$ induced by f are surjective. Assume that one of the following holds.*

- (a) *f has the FLP and $Y^{\top} = f^{-1}(X^{\top})$,*
- (b) *f has the PLP and $Y_{\perp} = f^{-1}(X_{\perp})$,*
- (c) *f has the TLP with respect to X_{\perp} and X^{\top} .*

Then f is a weak equivalence.

Cylinders. Let $X, Y, Z \in \mathbb{I}\Box\text{Set}$ and $f : Y \rightarrow X, g : Z \rightarrow X$ ipc-maps, assuming f and g to have disjoint images; this is not used directly in the construction, but crucial in proofs.

The *cylinder* $C(f, g)$ is the ipc-set such that $C(f, g)[U]$ is the set of (x, K, L, φ, ψ) , where $x \in X[U]$, K is an initial and L is a final sub-ipc-set of $\mathbb{I}\Box^U$, $\varphi : K \rightarrow Y$ and $\psi : L \rightarrow Z$ are ipc-maps satisfying $f \circ \varphi = \iota_x|_K$ and $g \circ \psi = \iota_x|_L$. For $d_{A,B} \in \mathbb{I}\Box(V, U)$ and $(x, K, L, \varphi, \psi) \in C(f, g)[U]$, we put

$$\delta_{A,B}(x, K, L, \varphi, \psi) = (\delta_{A,B}(x), (\mathbb{I}\Box^{d_{A,B}})^{-1}(K), (\mathbb{I}\Box^{d_{A,B}})^{-1}(L), \varphi \circ \mathbb{I}\Box^{d_{A,B}}, \psi \circ \mathbb{I}\Box^{d_{A,B}}).$$

Equivalently, $C(f, g)[U]$ is the set of commutative diagrams of solid arrows in Fig. 4 and the face map $\delta_{A,B}$ composes the diagram with the dashed arrows. The following is then clear (recall that $f(Y) \cap g(Z) = \emptyset$).

► **Lemma 34.** *For every $(x, K, L, \varphi, \psi) \in C(f, g)$ we have $K \subseteq (\iota_x)^{-1}(f(Y))$ and $L \subseteq (\iota_x)^{-1}(g(Z))$. Thus $K \cap L = \emptyset$, $x \in f(Y)$ implies $L = \emptyset$, and $x \in g(Z)$ implies $K = \emptyset$.*

$C(f, g)$ is equipped with the ipc-maps shown in Fig. 4. They are defined by $j(x) = (x, \emptyset, \emptyset, \emptyset, \emptyset)$, $p(x, K, L, \varphi, \psi) = x$, $\tilde{f}(y) = (f(y), \mathbb{I}\Box^{\text{iev}(y)}, \emptyset, \iota_y, \emptyset)$, $\tilde{g}(z) = (g(z), \emptyset, \mathbb{I}\Box^{\text{iev}(z)}, \emptyset, \iota_z)$. Below we collect some of their properties.

► **Lemma 35.**

- (a) $p \circ \tilde{f} = f, p \circ \tilde{g} = g, p \circ j = \text{id}_X$.
- (b) \tilde{f} is an initial inclusion and $\tilde{f}(Y) = \{(x, K, L, \varphi, \psi) \in C(f, g) \mid K = \mathbb{I}\Box^{\text{iev}(x)}, L = \emptyset\}$.
- (c) \tilde{g} is a final inclusion and $\tilde{g}(Z) = \{(x, K, L, \varphi, \psi) \in C(f, g) \mid L = \mathbb{I}\Box^{\text{iev}(x)}, K = \emptyset\}$.
- (d) j is an inclusion and $j(X) = \{(x, \emptyset, \emptyset, \emptyset, \emptyset) \in C(f, g)\}$.
- (e) $\tilde{f}(Y), \tilde{g}(Z)$ and $j(X)$ are pairwise disjoint.

► **Proposition 36.** *The projection $p : C(f, g) \rightarrow X$ has the FLP and PLP, and the TLP with respect to $f(Y)$ and $g(Z)$.*

Proof of Prop. 24. We show only the first claim; the second follows by reversal. Let L be recognised by a simple iHDA X with start cell $x_{\perp} \in X[U]$. Let Y be the iHDA with underlying precubical set $C(\iota_{\perp}^X, \emptyset)$ ($\emptyset : \emptyset \rightarrow X$ is the empty map). Let $y_{\perp} = (x_{\perp}, \mathbb{I}\square^U, \emptyset, \text{id}_{\mathbb{I}\square^U}, \emptyset)$ be the only start cell of Y and $Y^{\top} = p^{-1}(X^{\top})$. Since $\iota_{\perp}^Y = \widetilde{\iota_{\perp}^X}$, Y is start proper (Lem. 35(b)). The projection $p : Y \rightarrow X$ has the FLP (Prop. 36); thus $\mathsf{L}(Y) = \mathsf{L}(X) = L$ (Prop. 33(a)). ◀

Proof of Prop. 15. Suppose first that L is simple. Let X be a start simple and start proper iHDA recognising L (Prop. 24) and let Y be the iHDA with same underlying ipc-set and start cells as X and accept cells $Y^{\top} = X^{\top} \setminus X_{\perp}$. By Lem. 31, an accepting path $\alpha \in \mathsf{P}_X$ is accepting in Y iff it has positive length ($\text{ev}(\alpha)$ is not an identity), thus $\mathsf{L}(Y) = \mathsf{L}(X) \setminus \text{ld} = L \setminus \text{ld}$ is regular. If L is not simple, then let $L = \bigcup_i L_i$ be a finite sum of simple languages. Then $L \setminus \text{ld} = (\bigcup_i L_i) \setminus \text{ld} = \bigcup_i (L_i \setminus \text{ld})$ is regular by the first case and Prop. 18. ◀

7 Conclusion

Automata accept languages, but higher-dimensional automata (HDAs) have so far been an exception to this rule. We have proved a Kleene theorem for HDAs, connecting models to behaviours through an equivalence between regular and rational languages.

Showing that regular languages are rational was quite direct, but the converse direction required some effort. One reason is that HDAs may be glued not only at states, but also at higher-dimensional cells. This in turn led us to consider languages of pomsets with interfaces (ipomsets) and to equip HDAs with interfaces (iHDAs), too. After showing that HDAs and iHDAs recognise the same languages, we used topology-inspired constructions to glue (i)HDAs and show that rational operations on languages can be reflected by operations on (i)HDAs.

Kleene theorems build bridges between machines and languages, and there is a vast literature on the subject. In non-interleaving concurrency, one school considers (Mazurkiewicz) trace languages. Zielonka introduces asynchronous automata and shows that languages are regular iff they are recognisable [26]. Droste’s automata with concurrency relations have similar properties [4]. Yet not all rational trace languages (generated from singletons using union, concatenation and Kleene star) are recognisable [3]. Trace languages use a binary notion of independence and already 2-dimensional HDAs may exhibit behaviour that cannot be captured by trace languages [12].

Another school studies Kleene theorems for series-parallel pomset languages and automata models for these, such as branching and pomset automata [17, 19], and for Petri automata [2]. Series-parallel pomsets are incomparable to the interval orders accepted by Petri nets or HDAs, see [8, 25].

HDAs have been developed first of all with a view on operational, topological and geometric aspects, see [10] and the extensive bibliography of [24]. Languages have only been introduced recently [6]. Topological intuition has also guided our work, for example in the cylinder construction. Partial HDAs [9] were introduced for defining open maps and unfoldings on HDAs. HDAs with interfaces are a special case of partial HDAs, and our tools and techniques should be useful for those as well.

Our new formalisation of (i)HDAs as presheaves over a category of labelled ordered sets opens up connections to presheaf automata [22], coalgebra, and open maps [16], which we intend to explore. Finally, our introduction of iHDA morphisms akin to cofibrations and trivial fibrations hints at factorisation systems for HDAs. Weak factorisation systems and model categories have been considered in a bisimulation context, for example in [18], so we wonder about the connection.

References

- 1 Marc Bezem, Thierry Coquand, and Simon Huber. A model of type theory in cubical sets. In Ralph Matthes and Aleksy Schubert, editors, *TYPES*, volume 26 of *Leibniz International Proceedings in Informatics*, pages 107–128. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013. doi:10.4230/LIPIcs.TYPES.2013.107.
- 2 Paul Brunet and Damien Pous. Petri automata. *Logical Methods in Computer Science*, 13(3), 2017. doi:10.23638/LMCS-13(3:33)2017.
- 3 Christian Choffrut and Leucio Guerra. Logical definability of some rational trace languages. *Mathematical Systems Theory*, 28(5):397–420, 1995. doi:10.1007/BF01185864.
- 4 Manfred Droste. A Kleene theorem for recognizable languages over concurrency monoids. In Serge Abiteboul and Eli Shamir, editors, *ICALP*, volume 820 of *Lecture Notes in Computer Science*, pages 388–399. Springer, 1994. doi:10.1007/3-540-58201-0_84.
- 5 Uli Fahrenberg. *Higher-Dimensional Automata from a Topological Viewpoint*. PhD thesis, Aalborg University, Denmark, 2005.
- 6 Uli Fahrenberg, Christian Johansen, Georg Struth, and Krzysztof Ziemiański. Languages of higher-dimensional automata. *Mathematical Structures in Computer Science*, 31(5):575–613, 2021. doi:10.1017/S0960129521000293.
- 7 Uli Fahrenberg, Christian Johansen, Georg Struth, and Krzysztof Ziemiański. A Kleene theorem for higher-dimensional automata. *CoRR*, abs/2202.03791, 2022. arXiv:2202.03791.
- 8 Uli Fahrenberg, Christian Johansen, Georg Struth, and Krzysztof Ziemiański. Posets with interfaces as a model for concurrency. *Information and Computation*, 285(B):104914, 2022. doi:10.1016/j.ic.2022.104914.
- 9 Uli Fahrenberg and Axel Legay. Partial higher-dimensional automata. In Lawrence S. Moss and Pawel Sobocinski, editors, *CALCO*, volume 35 of *Leibniz International Proceedings in Informatics*, pages 101–115. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CALCO.2015.101.
- 10 Lisbeth Fajstrup, Eric Goubault, Emmanuel Haucourt, Samuel Mimram, and Martin Raussen. *Directed Algebraic Topology and Concurrency*. Springer, 2016. doi:10.1007/978-3-319-15398-8.
- 11 Peter C. Fishburn. *Interval Orders and Interval Graphs: A Study of Partially Ordered Sets*. Wiley, 1985.
- 12 Eric Goubault. Labelled cubical sets and asynchronous transition systems: an adjunction. In *CMCIM*, 2002. URL: <http://www.lix.polytechnique.fr/~goubault/papers/cmcim02.ps.gz>.
- 13 Jan Grabowski. On partial languages. *Fundamentae Informatica*, 4(2):427, 1981.
- 14 Marco Grandis. *Directed algebraic topology: models of non-reversible worlds*. New mathematical monographs. Cambridge University Press, 2009.
- 15 Marco Grandis and Luca Mauri. Cubical sets and their site. *Theory and Applications of Categories*, 11(8):185–211, 2003.
- 16 André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation from open maps. *Information and Computation*, 127(2):164–185, 1996.
- 17 Tobias Kappé, Paul Brunet, Bas Luttik, Alexandra Silva, and Fabio Zanasi. On series-parallel pomset languages: Rationality, context-freeness and automata. *Journal of Logic and Algebraic Methods in Programming*, 103:130–153, 2019. doi:10.1016/j.jlamp.2018.12.001.
- 18 Alexander Kurz and Jiří Rosický. Weak factorizations, fractions and homotopies. *Applied Categorical Structures*, 13(2):141–160, 2005.
- 19 Kamal Lodaya and Pascal Weil. Series-parallel languages and the bounded-width property. *Theoretical Computer Science*, 237(1-2):347–380, 2000. doi:10.1016/S0304-3975(00)00031-1.
- 20 Saunders Mac Lane and Ieke Moerdijk. *Sheaves in Geometry and Logic*. Springer, 1992.
- 21 Vaughan R. Pratt. Modeling concurrency with geometry. In *POPL*, pages 311–322, New York City, 1991. ACM Press.

- 22 Pawel Sobocinski. Relational presheaves, change of base and weak simulation. *J. Comput. Syst. Sci.*, 81(5):901–910, 2015. doi:10.1016/j.jcss.2014.12.007.
- 23 Rob J. van Glabbeek. Bisimulations for higher dimensional automata. Email message, June 1991. URL: <http://theory.stanford.edu/~rvlg/hda>.
- 24 Rob J. van Glabbeek. On the expressiveness of higher dimensional automata. *Theoretical Computer Science*, 356(3):265–290, 2006.
- 25 Walter Vogler. *Modular Construction and Partial Order Semantics of Petri Nets*, volume 625 of *Lecture Notes in Computer Science*. Springer, 1992. doi:10.1007/3-540-55767-9.
- 26 Wiesław Zielonka. Notes on finite asynchronous automata. *RAIRO - Theoretical Informatics and Applications*, 21(2):99–135, 1987. doi:10.1051/ita/1987210200991.

A Definitions of HDAs

Precubical sets and HDAs are used in a few different incarnations in the literature, all more or less equivalent. We expose some of these here in order to relate them to the setting in this paper; we make no claim as to completeness.

Precubical sets. according to Grandis [14,15] are presheaves on a small category \square_G which is generated by the following data:

- objects are $\{0, 1\}^n$ for $n \geq 0$;
- elementary coface maps $d_i^\nu : \{0, 1\}^n \rightarrow \{0, 1\}^{n+1}$, for $i = 1, \dots, n+1$ and $\nu = 0, 1$, are given by $d_i^\nu(t_1, \dots, t_n) = (t_1, \dots, t_{i-1}, \nu, t_i, \dots, t_n)$.

Elementary coface maps compose to coface maps $\{0, 1\}^m \rightarrow \{0, 1\}^n$ for $n \geq m$. See also [5].

\square_G -sets, *i.e.*, elements $X \in \mathbf{Set}^{\square_G^{\text{op}}}$, are then graded sets $X = \{X_n\}_{n \geq 0}$ (where $X_n = X[\{0, 1\}^n]$) together with face maps $X_n \rightarrow X_m$ for $n \geq m$. The elementary face maps are denoted $\delta_i^\nu = X[d_i^\nu]$, and they satisfy the *precubical identity*

$$\delta_i^\nu \delta_j^\mu = \delta_{j-1}^\mu \delta_i^\nu \quad (i < j) \tag{3}$$

for $\nu, \mu \in \{0, 1\}$.

The above elementary description of \square_G -sets may be taken as definition, allowing one to avoid any talk about presheaves. For example, van Glabbeek [24] defines a precubical set $Q = (Q, s, t)$ as a family of sets $(Q_n)_{n \geq 0}$ and maps $s_i : Q_n \rightarrow Q_{n-1}$, $1 \leq i \leq n$, such that $\alpha_i \circ \beta_j = \beta_{j-1} \circ \alpha_i$ for all $1 \leq i < j \leq n$ and $\alpha, \beta \in \{s, t\}$; this is evidently equivalent to the above.

The paper [6] introduces another base category, \square_Z , given as follows:

- objects are totally ordered sets (S, \dashrightarrow_S) ;
- morphisms $S \rightarrow T$ are pairs (f, ε) , where $f : S \hookrightarrow T$ is an order preserving injection and $\varepsilon : T \rightarrow \{0, \lrcorner, 1\}$ fulfils $f(S) = \varepsilon^{-1}(\lrcorner)$.

(The element \lrcorner stands for “executing”.) Letting $A = \varepsilon^{-1}(0)$ and $B = \varepsilon^{-1}(1)$, the above notion of morphisms is equivalent to having triples (f, A, B) consisting of $f : S \hookrightarrow T$ (order preserving and injective) and $A, B \subseteq T$ such that $T = A \sqcup f(S) \sqcup B$ (disjoint union). Except for the labelling, this is the same as our definition of \square in Section 2. ([6] also makes a connection to [1].)

Then [6] goes on to show that the full subcategory of \square_Z spanned by objects \emptyset and $\{1, \dots, n\}$ for $n \geq 1$ is skeletal and equivalent to \square_Z . Moreover, this subcategory, \square_Z , is shown to be isomorphic to \square_G , and that the presheaf categories on \square_Z and on \square_G (and thus also on \square_G) are uniquely naturally isomorphic. It is clear that \square_Z is a representative of the quotient of \square_Z under isomorphisms, so except for the labelling, this is again the same as our \square of Section 2.

The advantage of \square_Z , and of our \square , over the skeleton versions is that the precubical identity (3) is automatic and that there is a built-in notion of events, *i.e.*, in a \square_Z -set X , a cell $x \in X[U]$ has events U .

HDAs. are, generally speaking, labelled precubical sets (on the alphabet Σ) with specified start and accept cells. The labelling may be obtained using the labelling object $!\Sigma$ [12]. This is the precubical set with $!\Sigma_n = \Sigma^n$ and $\delta_i^y((a_1, \dots, a_n)) = (a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n)$, and a labelled precubical set is then a precubical map $X \rightarrow !\Sigma$: an object of the slice category of precubical sets over $!\Sigma$.

A labelling $\lambda : X \rightarrow !\Sigma$ induces a function $\lambda_1 : X_1 \rightarrow \Sigma$ with the property that for all $x \in X_2$, $\lambda_1(\delta_1^0(x)) = \lambda_1(\delta_1^1(x))$ and $\lambda_1(\delta_2^0(x)) = \lambda_1(\delta_2^1(x))$. Conversely, any such function extends uniquely to a precubical map $X \rightarrow !\Sigma$ [6, Lem. 14], so that λ_1 may be taken as the definition of labelling instead. This is the approach taken in [24], where an HDA is defined as a precubical set Q equipped with a function $\lambda_1 : Q \rightarrow \Sigma$ such that $\lambda_1(s_i(q)) = \lambda_1(t_i(q))$ for all $q \in Q_2$ and $i = 1, 2$, and subsets of start and accept states $I, F \subseteq Q_0$.

Another observation made in [6] is that regarded as a presheaf, $!\Sigma(S) = \mathbf{Set}(S, \Sigma)$, hence $!\Sigma$ is representable in \mathbf{Set} via the forgetful functor $\square_Z \rightarrow \mathbf{Set}$. This means that the labelling may be integrated into the base category, turning \square_Z into our \square with objects being labelled totally ordered sets. Using \square instead of \square_Z allows us to avoid working in the slice category: everything is labelled from the outset.

To sum up, let X be an HDA in our sense, then the corresponding HDA $(Q, s, t, \lambda_1, I, F)$ in the sense of [24] is given as follows.

- $Q_n = \coprod_{U \in \square, |U|=n} X[U]$.
- If $x \in X[U]$, then $s_i(x) = \delta_u^0(x)$ and $t_i(x) = \delta_u^1(x)$, where $u \in U$ is the i -th smallest element of U in the order \dashrightarrow_U .
- If $x \in X[U] \subseteq Q_1$ with $U = (\{e\}, \emptyset, \lambda(e) = a)$, then $\lambda_1(x) = a$.
- $I = X_\perp, F = X^\top$.

Conversely, let $(Q, s, t, \lambda_1, I, F)$ be an HDA as in [24]. There exist unique labelling functions $\lambda_n : Q_n \rightarrow \Sigma^n$ such that $\lambda_{n-1}(\alpha_i(q)) = \delta_i(\lambda_n(q))$ [6, Lem. 14], where $\alpha \in \{s, t\}$ and δ_i skips the i -th element of a sequence. We construct an HDA X as follows.

- $X[U] = \{q \in Q_n \mid \lambda_n(q) = U\}$ for $U \in \square$ and $|U| = n$.
- $\delta_a^0(q) = s_i(q)$ and $\delta_a^1(q) = t_i(q)$ for $q \in X[U]$ and $a \in U$ the i -th smallest element of U in the order \dashrightarrow_U . The remaining face maps are compositions of these.
- $X_\perp = I, X^\top = F$.

Finally, the only difference between van Glabbeek's HDAs and ours is that we allow start and accept cells that are not necessarily vertices. Our definition of HDAs, based on lo-sets, treats the elements of these sets as labelled events that are regarded either as unstarted, executed, or terminated. In this way, they incorporate events in a direct manner. This makes our definition of HDAs into a model that combines event-based and state-based models of concurrency.




Diamonds for Security: A Non-Interleaving Operational Semantics for the Applied Pi-Calculus

Clément Aubert   

Augusta University, Augusta, GA, USA

Ross Horne  

University of Luxembourg, Luxembourg

Christian Johansen   

NTNU – Norwegian University of Science and Technology, Gjøvik, Norway

Abstract

We introduce a non-interleaving structural operational semantics for the applied π -calculus and prove that it satisfies the properties expected of a labelled asynchronous transition system (LATS). LATS have well-studied relations with other standard non-interleaving models, such as Mazurkiewicz traces or event structures, and are a natural extension of labelled transition systems where the independence of transitions is made explicit. We build on a considerable body of literature on located semantics for process algebras and adopt a static view on locations to identify the parallel processes that perform a transition. By lifting, in this way, work on CCS and π -calculus to the applied π -calculus, we lay down a principled foundation for reusing verification techniques such as partial-order reduction and non-interleaving equivalences in the field of security. The key technical device we develop is the notion of located aliases to refer unambiguously to a specific output originating from a specific process. This light mechanism ensures stability, avoiding disjunctive causality problems that parallel extrusion incurs in similar non-interleaving semantics for the π -calculus.

2012 ACM Subject Classification Theory of computation \rightarrow Program semantics; Theory of computation \rightarrow Concurrency; Theory of computation \rightarrow Process calculi

Keywords and phrases Security, Processes, Structural operational semantics, Asynchronous transition systems, Applied pi-calculus

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.30

Acknowledgements The authors wish to express their gratitude to the reviewers for their recommendations, that helped us improve our presentation.

1 Introduction

The purpose of this paper is to give a principled foundation for methods from concurrency theory that are gaining traction in the verification of security properties. Tools used in the security domain have applied partial-order-based techniques, notably partial-order reduction (POR), to improve their efficiency, thereby increasing the size of the systems that can be analyzed [6, 7, 8, 17, 18, 21, 27, 38]. To date, much of that work is tool-driven and guided by examples. However, without a solid foundation, we cannot be certain that the methods employed are correct, and hence, if an untested example is presented, the tool might produce incorrect results or fail to apply reductions where they might well have been applied.

This paper fulfils what we believe to be an important role: drawing from decades of theory on the topic of non-interleaving semantics for process algebras, we bring essential concurrency concepts to the security verification community. In particular, we aim for a semantics that is operational, rather than denotational, to stay close to the existing semantics employed by the main tools in the security community, so that our work may easily be adopted. As in the non-interleaving tradition, we define a concept of *event* as an enhancement of actions. The



© Clément Aubert, Ross Horne, and Christian Johansen;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Sławomir Lasota, and Anca Muscholl; Article No. 30; pp. 30:1–30:26



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

notion of independence is then defined on events in a straightforward way using the structure of the events only. This enables the implementation of techniques such as partial-order reduction on the fly.

The applied π -calculus [1, 43] has become instrumental in leveraging theoretical process calculi to certify and verify security protocols [4, 10, 14, 31, 33, 36]. Its syntax can describe a variety of interactions between the processes involved in a security protocol, stressing the study of the communication of complex messages. Abstracting away from cryptographic primitives with an equational theory allows tools to focus on problems arising due to the information flow in security protocols, while its flexibility allows a large variety of situations to be modelled. Its inductive structure lends itself to automation of the analysis and verification of protocols used in production thanks to tools such as ProVerif [10], DEEPSEC [17], Akiss [16], Sapic [35], SAT-Equiv [19] and SPEC [46], which have flavours of the applied π -calculus as their input language.

Labelled asynchronous transition systems (LATS) are a non-interleaving model of concurrency, which, because they extend transition systems in a natural way, are suited to being the objects generated by our structural operational semantics for the applied π -calculus. The word *asynchronous* does not refer to the type of communication between processes: it is a nod to the Petri net literature, where independent transitions are said to act asynchronously [41]. LATS were introduced by Bednarczyk [9] and Shields [45], and have been well studied in the concurrency literature [28, 29, 51]. They are generally required to satisfy properties such as *event determinism*, and to provide an independence relation that satisfies what we call *concurrency diamonds*. Developing a LATS is also an important prerequisite for further methods to be developed such as non-interleaving process equivalences.

Endowing the applied π -calculus with a structural operational semantics that defines a labelled asynchronous transition system gives us a concrete path towards applying non-interleaving concurrency techniques to security problems such as the verification of security protocols. This requires us to import and mix ideas from different communities and lines of work, and to carefully design “the right fit” to prove seamlessly the desired properties of a LATS. We believe our proposal to be not only elegant, but also enlightening in the way it sidesteps “traditional” problems stemming from the π -calculus, such as the need to represent some forms of extrusion with disjunctive causality [20, 30]. In particular, in our theory there is never any ambiguity about which outputs an event is causally dependent on, in contrast to the traditional semantics for the π -calculus [11] where, if two outputs concurrently extrude the same name that is then later used, then the semantics does not record from which output the name originates.

Our proposal is also lightweight – our *events* consist simply of a location (a binary string indicating where in the binary tree of parallel and choice components the event originates) and an action label – and provides event determinism almost for free. Avoiding disjunctive causality will allow our model to be related to more standard concurrency models such as safe Petri nets or prime event structures.

Outline. Sect. 2 recalls the abstract concept of a LATS. Sect. 3 introduces a syntax for the applied π -calculus. Sect. 4 explains the design of our non-interleaving structural operational semantics. Sect. 5 presents the main result: that our non-interleaving structural operational semantics is a labelled asynchronous transition system. Sect. 6 situates this work in the literature, particularly in relation to POR for the applied π -calculus. The appendices provide example derivations and outline the key elements of the proof of the main theorem.

2 Our Target Model: Labelled Asynchronous Transition Systems

A labelled asynchronous transition system is a labelled transition system enriched with an independence relation on events I that satisfies what we call *diamond* properties. Standard labelled transition systems, such as those obtained from structural operational semantics of process algebras such as CCS, π -calculus, and applied π -calculus, are labelled with *actions*, indicating, e.g., the input or output action of a process. Labelled *asynchronous* transition systems (LATS), in contrast, are labelled with *events*, which contain more information, sufficient to distinguish between events that are triggered in different ways but share the same action label. We denote states (which usually in process algebras are process terms) by capital letters, events by small letters, and transitions between states labelled with an event as $A \xrightarrow{e} B$. We illustrate the properties of LATS using standard π -calculus notation. The reader familiar with the π -calculus may safely skip the following info box.

Key features of the π -calculus: We assume for now only some familiarity with π -calculus features, some of which we informally recall here. Later, in Sect. 3, we extend this to the applied π -calculus and define formally its semantics:

- Name restriction $\nu x.P$ binds occurrences of variable x in P , indicating that x is a freshly generated name. Importantly, an observer (who may be an attacker) cannot know (e.g., by guessing) which name was generated, without intercepting a communication of this name on a public channel.
 - An output prefix $\bar{x}(y).P$ indicates that the variable y is output on a channel x , before continuing to execute process P . Such outputs may only be observed if the channel is known to the observer, either because x is a free variable or is a fresh name that has previously been output.
 - An input prefix $x(y).P$ indicates that a channel x is used to receive a message, which is always a variable representing a name in the π -calculus. Notably, a fresh name may only be received if it has previously been output. The process then continues to execute P , but with occurrences of variable y in P replaced by the variable received, e.g., as $P\{z/y\}$ if z was received along x .
 - Parallel composition $P \mid Q$ indicates that processes P and Q execute in separate locations, possibly communicating with each other by synchronising an input and output on the same channel name, whether or not the channel is known to an observer. The precise meaning of parallel composition is the main subject of the interleaving/non-interleaving spectrum of process semantics.
 - The match prefix $[x = y]P$ should be read as, “if x and y are the same variable (due to a prior input action for example) then P can execute.”
 - The process 0 represents termination. We follow standard conventions such as omitting the deadlock process 0 when it is preceded by an action (e.g., output or input) prefix.
- A distinguishing feature of the π -calculus is that the scope of a name restriction is mobile in the sense that it can expand when a message containing the variable it binds is output. We assume that action prefixes and name restriction bind stronger than parallel composition.

To begin with, a LATS must satisfy the following property, which ensures that, in any given state, any two (co-initial) transitions labelled with the same event are actually the same transition (up to some minimal congruence relation \equiv , quotienting the set of states).

► **Definition 1** (event determinism). *A LATS satisfies event determinism whenever*

$$\text{If } A_0 \equiv A_1, A_0 \xrightarrow{e} B_0 \text{ and } A_1 \xrightarrow{e} B_1, \text{ then } B_0 \equiv B_1.$$

For example, for a processes $\bar{c}\langle n \rangle \mid \bar{c}\langle n \rangle$ consisting of two parallel threads both sending n on channel c , a LATS must label the two output transitions differently, which is not guaranteed by standard labelled transition systems.

The other two properties a LATS must satisfy are the diamond properties (sometimes called “sideways diamond” [42] and “square property” [37]), which ensure that events considered to be independent with respect to the independence relation I can permute. The first diamond property ensures that two independent events labelling (co-initial) transitions from the same state can be performed in either order without affecting the outcomes.

► **Definition 2** (diamond property 1). *A LATS satisfies diamond property 1 whenever*

$$\text{If } e_0 \ I \ e_1, A \xrightarrow{e_0} B_0, \text{ and } A \xrightarrow{e_1} B_1 \text{ then } \exists C_0, C_1 \text{ s.t. } B_0 \xrightarrow{e_1} C_0, B_1 \xrightarrow{e_0} C_1 \text{ and } C_0 \equiv C_1.$$

The second diamond property concerns independent events labelling (composable) transitions in *consecutive* states, ensuring such transitions may be permuted.

► **Definition 3** (diamond property 2). *A LATS satisfies diamond property 2 whenever*

$$\text{If } e_0 \ I \ e_1, A \xrightarrow{e_0} B_0, \text{ and } B_0 \xrightarrow{e_1} C_0, \text{ then } \exists B_1, C_1 \text{ s.t. } A \xrightarrow{e_1} B_1, B_1 \xrightarrow{e_0} C_1 \text{ and } C_0 \equiv C_1.$$

Observe that all of the above properties only concern co-initial or composable transitions, i.e., transitions from the same state or adjacent states. We use the following example to illustrate why this is significant: $\nu n.(\bar{c}\langle n \rangle \mid \overline{c(x)}.[x = n]\overline{\text{ok}}\langle \text{ok} \rangle)$. For any execution of this process in which the event corresponding to $\overline{\text{ok}}\langle \text{ok} \rangle$ occurs, the two events corresponding to $\bar{c}\langle n \rangle$ and $c(x)$ must have both occurred previously. Since we work with an *early* semantics, strictly speaking we have one event for each possible instantiation of the variable x in the input $c(x)$. *Structural* (a.k.a. *prefixing*) causality ensures that the event corresponding to $\overline{\text{ok}}\langle \text{ok} \rangle$ should not be independent from any of the input events corresponding to $c(x)$, since this part (or *location*) of the process must execute the latter to be able to access the former. *Link* (a.k.a. *name*) causality ensures that the event corresponding to $\bar{c}\langle n \rangle$ should not be independent from the input event corresponding to $c(x)$ when x is instantiated with the name n , which would also enable the guard $x = n$. Taken together, those two causalities ensure that our expectations regarding the concurrency inherent in π -calculi are met.

In contrast to the above observations, although the events corresponding to $\bar{c}\langle n \rangle$ and $\overline{\text{ok}}\langle \text{ok} \rangle$, in the above example, are causally dependent on each other by transitivity, when defining a LATS, we are free to allow them to be defined as “independent”, since these events can never be executed concurrently or consecutively, and hence they will never be considered together in a diamond property. This is a reason why LATS are suited for defining a non-interleaving structural operational semantics, as there is no need to compute global dependencies between events: local calculations are enough to determine whether consecutive events are concurrent. If global dependencies are required, they are established by unfoldings of LATS into event structures and Petri nets [39].

The relatively light requirements on the independence relation, as explained above, justifies why LATS offer an attractive take on structural operational semantics. A structural operational semantics makes transitions easy to compute, and an easy to compute independence relation facilitates partial-order reduction, that drastically reduces the number of states to explore when verifying concurrent processes [3, 18]. An easily calculated independence relation also facilitates the checking of non-interleaving variants of equivalences, e.g., distinguishing $\bar{c}\langle n \rangle \mid \bar{c}\langle n \rangle$ from $\bar{c}\langle n \rangle.\bar{c}\langle n \rangle$ (the *autoconcurrency* problem). Causality in the π -calculus has been explored in related work [11, 20, 22, 34, 40].

PROCESSES: $P, Q, R ::=$		GUARDED PROCESSES: $G, H ::=$	
0	deadlock	$M(x).P$	input prefix
$\nu x.P$	new	$\overline{M}\langle N \rangle.P$	output prefix
$P \mid Q$	parallel	$[M = N]G$	match
G	guarded process	$[M \neq N]G$	mismatch
$!P$	replication	$G + H$	choice
EXTENDED PROCESSES:			
$A, B, C ::=$	$\sigma \mid P$	active process	
	$\nu x.A$	new	

■ **Figure 1** Syntax of processes with guarded choice and of extended processes.

3 A Syntax for the Applied π -Calculus

We detail the syntax for the applied π -calculus we employ in Fig. 1 – which is close to the one used in tools such as e.g., ProVerif [1, 10]. Variables are denoted by lowercase roman letters such as x, y, z (generally reserved for input variables), a, b, c (generally reserved for channel names) m, n , (generally reserved for nonces and keys). All variables are the same syntactic category, but we are careful to distinguish variables from *aliases*, ranging over α, β, γ . Traditionally in the applied π -calculus, aliases are also simply variables, but they play a special role in this theory. We will explain the notion of alias properly in Sect. 4.2.

Variables, aliases and function symbols – discussed below – are used to build messages, denoted by M, N, K . In processes, denoted by P, Q, R , variables, but not aliases, can be bound by input prefix $M(x).P$ or fresh name binders $\nu x.P$, where the latter is used to indicate which variables – in this case, x – are treated as opaque names, such as private keys hidden from an attacker observing the process. We use sequences of names $\nu \vec{x}.P$ to abbreviate multiple name binders defined inductively such that $\nu \epsilon.P = P$ and $\nu x, \vec{y}.P = \nu x.\nu \vec{y}.P$, where ϵ is the empty sequence.

As standard, a substitution σ, θ or ρ is a function with a domain ($\text{dom}(\sigma) = \{\alpha : \alpha \neq \alpha\sigma\}$) and a range ($\text{ran}(\sigma) = \{\alpha\sigma : \alpha \in \text{dom}(\sigma)\}$) that can be applied to messages as suffixes, e.g., $\text{fst}(\alpha)\{\langle m, n \rangle / \alpha\} = \text{fst}(\langle m, n \rangle)$. We write id for the identity substitution, with $\text{dom}(\text{id}) = \text{ran}(\text{id}) = \emptyset$. Substitutions can be composed, notated $\sigma \circ \theta$, and are applied in reverse to function composition, thus $M(\sigma \circ \theta) = (M\sigma)\theta$. When applied to processes, substitutions are capture-avoiding with respect to processes $\nu x.P$ and $a(x).P$ that bind x in P . In this work, *active substitutions*, which map aliases in their finite domain to messages containing no aliases (hence are idempotent, i.e., $\sigma \circ \sigma = \sigma$), play a central role.

The ability to choose any function symbols to construct messages is the real flexibility of the applied π -calculus, allowing many message theories to be encoded, representing cryptographic functions by defining an equational theory E , containing equations such as for the decryption function $\text{dec}(\{M\}_K, K) =_E M$ or first projection of a pair $\text{fst}(\langle M, N \rangle) =_E M$. This clean design allows us to separate problems related to the semantics of processes, as explored in this paper, from problems associated with conducting proofs in the presence of specific choices of message theories. Thus we never fix a specific message theory in this paper, and any we provide is just to make examples more digestible.

Extended processes, ranging over A, B, C , offer a compact way of representing a process along with the messages and names that have been sent. In extended processes, the messages already sent are represented by active substitutions; and the scope of fresh name binders

are enlarged to include the substitution, so that they may bind variables in both the substitutions representing messages that have been output and in the continuation processes. In an extended process $A = \nu \vec{x}.(\sigma \mid P)$, we assume, in this work, a *normal form*, where aliases do not appear in processes. This has the effect that the active substitution σ in the extended process A has already been applied to P .

► **Definition 4** (free variables and aliases). *A variable x (resp. an alias α) is free in a message M if $x \in \text{fv}(M)$ (resp. $\alpha \in \text{fa}(M)$) for*

$$\begin{aligned} \text{fv}(f(M_1, \dots, M_n)) &= \cup_{i=1}^n \text{fv}(M_i) & \text{fv}(x) &= \{x\} & \text{fv}(\alpha) &= \emptyset \\ \text{fa}(f(M_1, \dots, M_n)) &= \cup_{i=1}^n \text{fa}(M_i) & \text{fa}(x) &= \emptyset & \text{fa}(\alpha) &= \{\alpha\}. \end{aligned}$$

The fv function extends in the standard way to (extended) processes, letting $\text{fv}(\nu x.P) = \text{fv}(P) \setminus \{x\}$ and $\text{fv}(M(x).P) = \text{fv}(M) \cup (\text{fv}(P) \setminus \{x\})$, and similarly for $\text{fv}(A)$.

► **Definition 5** (fresh). *We say a variable x is fresh for a message M (resp. process P , extended process A), written $x \# M$ (resp. $x \# P$, $x \# A$) whenever $x \notin \text{fv}(M)$ (resp. $x \notin \text{fv}(P)$, $x \notin \text{fv}(A)$), and similarly for aliases. Freshness extends point-wise to lists of entities, i.e., $x_1, x_2, \dots, x_m \# M_1, M_2, \dots, M_n$, denotes the conjunction of all $x_i \# M_j$ for all $1 \leq i \leq m$ and $1 \leq j \leq n$.*

► **Definition 6** (α -equivalence). *We define α -equivalence (denoted \equiv_α) for variables only (not aliases which are fixed “addresses”) as the least congruence (a reflexive, transitive, and symmetric relation preserved in all contexts) such that, whenever $z \# \nu x.P$, we have $\nu x.P \equiv_\alpha \nu z.(P\{z/x\})$ and $M(x).P \equiv_\alpha M(z).(P\{z/x\})$. Similarly, for extended processes, we have the least congruence such that, whenever $z \# \nu x.A$, we have $\nu x.A \equiv_\alpha \nu z.(A\{z/x\})$.*

► **Definition 7** (capture-avoiding substitutions). *Restriction is such that $\theta \upharpoonright_{\bar{\alpha}}(x) = \theta(x)$ if $x \in \bar{\alpha}$ and x otherwise. Capture-avoiding substitutions are defined for processes such that for any $z \# \text{dom}(\sigma)$, $\text{ran}(\sigma)$, $\nu x.P$, we have $(M(x).P)\sigma \equiv_\alpha M\sigma(z).P\{z/x\}\sigma$ and $(\nu x.P)\sigma \equiv_\alpha \nu z.P\{z/x\}\sigma$. For extended processes, it is defined such that $(\nu x.A)\rho \equiv_\alpha \nu z.(A(\{z/x\} \circ \rho))$ and $(\sigma \mid P)\rho = (\sigma \circ \rho \upharpoonright_{\text{dom}(\sigma)} \mid P\rho)$, for $z \# \text{dom}(\rho)$, $\text{ran}(\rho)$, $\nu x.A$.*

► **Definition 8** (structural congruence). *Our minimal structural congruence (denoted \equiv) is the least equivalence relation on extended processes extending α -equivalence such that whenever $\sigma = \theta$ (i.e., the substitutions denote the same function), $P \equiv_\alpha Q$ and $A \equiv B$, we have:*

$$\sigma \mid P \equiv \theta \mid Q \qquad \nu x.A \equiv \nu x.B \qquad \nu x.\nu z.A \equiv \nu z.\nu x.A$$

Notice that we did not include equations such as $P \mid Q \equiv Q \mid P$, $(P \mid Q) \mid R \equiv P \mid (Q \mid R)$ or $P \mid 0 \equiv P$, for reasons that will become clear in Sect. 4.2. Many similar systems (sometimes called *proved* transition systems) miss a structural congruence altogether [15, 24, 26], or miss the associativity and commutativity of the parallel composition [25, p. 242], since they can alter the label of the transition and complicate tracking the source of an action, yet can be recovered by a suitable observational equivalence (e.g., a non-interleaving bisimilarity).

4 The Design of a Non-interleaving Structural Operational Semantics

The located structural operational semantic rules introduced in Figs. 2 and 3 adapt the recent semantics developed for the applied π -calculus in [31, 32], which can be obtained from the one here by simply ignoring the information in red and any other information beneath the arrow in labelled transitions. To obtain a structural operational semantics for

$$\begin{array}{c}
\frac{M =_E K}{K(x).P \xrightarrow[\square]{MN} \text{id} \mid P\{N/x\}} \text{INP} \qquad \frac{M =_E K}{\overline{K}(N).P \xrightarrow[\square]{\overline{M}(\lambda)} \{N/\lambda\} \mid P} \text{OUT} \\
\\
\frac{P \xrightarrow[u]{\pi} \nu \vec{x}.(\sigma \mid R) \quad \vec{x} \# Q}{P \mid Q \xrightarrow[\text{0u}]{\pi} \nu \vec{x}.(\sigma \mid R \mid Q)} \text{PAR-L} \qquad \frac{Q \xrightarrow[u]{\pi} \nu \vec{x}.(\sigma \mid R) \quad \vec{x} \# P}{P \mid Q \xrightarrow[\text{1u}]{\pi} \nu \vec{x}.(\sigma \mid P \mid R)} \text{PAR-R} \\
\\
\frac{P \xrightarrow[u]{\pi} A \quad x \# \text{fv}(\pi)}{\nu x.P \xrightarrow[u]{\pi} \nu x.A} \text{EXTRUDE} \qquad \frac{A \xrightarrow[u]{\pi} B \quad x \# \text{fv}(\pi)}{\nu x.A \xrightarrow[u]{\pi} \nu x.B} \text{RES} \\
\\
\frac{P \xrightarrow[\text{s[s']}]{\overline{M}\sigma(\lambda)} \nu \vec{x}.(\{N/\lambda\} \mid Q) \quad \vec{x} \# \text{ran}(\sigma) \quad \text{fa}(M) \subseteq \text{dom}(\sigma) \quad \text{s}\lambda \# \text{dom}(\sigma)}{\sigma \mid P \xrightarrow[\text{s[s']}]{\overline{M}(s\lambda)} \nu \vec{x}.(\sigma \circ \{N/\text{s}\lambda\} \mid Q)} \text{ALIAS-OUT} \\
\\
\frac{P \xrightarrow[u]{\pi\sigma} \nu \vec{x}.(\text{id} \mid Q) \quad \vec{x} \# \text{ran}(\sigma) \quad \text{fa}(\pi) \subseteq \text{dom}(\sigma)}{\sigma \mid P \xrightarrow[u]{\pi} \nu \vec{x}.(\sigma \mid Q)} \text{ALIAS-FREE} \\
\\
\frac{G \xrightarrow[t]{\pi} A}{G + H \xrightarrow[\text{[0t]}]{\pi} A} \text{SUM-L} \qquad \frac{H \xrightarrow[t]{\pi} A}{G + H \xrightarrow[\text{[1t]}]{\pi} A} \text{SUM-R} \qquad \frac{P \mid !P \xrightarrow[u]{\pi} A}{!P \xrightarrow[u]{\pi} A} \text{BANG} \\
\\
\frac{P \xrightarrow[u]{\pi} A \quad M =_E N}{[M = N]P \xrightarrow[u]{\pi} A} \text{MAT} \qquad \frac{P \xrightarrow[u]{\pi} A \quad M \neq_E N}{[M \neq N]P \xrightarrow[u]{\pi} A} \text{MISMAT}
\end{array}$$

■ **Figure 2** An *early* located non-interleaving structural operational semantics.

the π -calculus in this style, simply assume that all messages M , N , etc., are variables, and that two variables are equal only if they are the same variable. After briefly introducing our structural operational semantics, we explain our design choices in the subsections that follow.

Action labels range over π . A *free input* action label MN indicates the input of message N on channel M . A *bound output* action label $\overline{M}(\alpha)$ indicates the output of something on channel M where the message we output is assigned the alias α , which can be used to refer to that message in the future by the observer, or a direct communication τ that the observer does not intercept. This is the reason why the substitution is applied to the continuation in the INP rule, but “stored” in the substitution in the OUT rule. The label τ denotes an internal communication, and is used in the synchronisation rules presented in Fig. 3.

The functions for free variables and free aliases extend to labels as follows.

$$\text{fv}(\pi) = \begin{cases} \text{fv}(M) \cup \text{fv}(N) & \text{if } \pi = MN \\ \text{fv}(M) & \text{if } \pi = \overline{M}(\alpha) \\ \emptyset & \text{if } \pi = \tau \end{cases} \qquad \text{fa}(\pi) = \begin{cases} \text{fa}(M) \cup \text{fa}(N) & \text{if } \pi = MN \\ \text{fa}(M) & \text{if } \pi = \overline{M}(\alpha) \\ \emptyset & \text{if } \pi = \tau \end{cases}$$

Readers familiar with labelled transition systems will immediately notice the additional annotation below transitions, that uses prefixes composed of **0s** and **1s**. This indicates the *location* of the parallel sub-process (a.k.a. component or thread) performing the action:

$$\begin{array}{c}
 \frac{P \xrightarrow[\ell_0]{\overline{M}(\lambda)} \nu \vec{y}.(\{N/\lambda\} | P') \quad Q \xrightarrow[\ell_1]{MN} \nu \vec{w}.(\text{id} | Q') \quad \vec{y} \# Q \quad \vec{w} \# P, \vec{y}}{P | Q \xrightarrow[(0\ell_0, 1\ell_1)]{\tau} \nu \vec{y}, \vec{w}.(\text{id} | P' | Q')} \text{CLOSE-L} \\
 \\
 \frac{P \xrightarrow[\ell_0]{MN} \nu \vec{y}.(\text{id} | P') \quad Q \xrightarrow[\ell_1]{\overline{M}(\lambda)} \nu \vec{w}.(\{N/\lambda\} | Q') \quad \vec{w} \# P \quad \vec{y} \# Q, \vec{w}}{P | Q \xrightarrow[(0\ell_0, 1\ell_1)]{\tau} \nu \vec{y}, \vec{w}.(\text{id} | P' | Q')} \text{CLOSE-R}
 \end{array}$$

■ **Figure 3** Rules for communication.

► **Definition 9** (locations). A location ℓ is of the form $s[t]$, where $s \in \{0, 1\}^*$ and $t \in \{0, 1\}^*$. If s or t is empty, we omit it (hence, we write $\epsilon[\epsilon]$ as $[\]$).

The colour coding above is to emphasise everywhere, what we call, the *location prefix* used to indicate the parallel component from which a transition originates, and the “ t part” of a location serves to identify which operand of the sum triggered the transition, as indicated in the SUM-L and SUM-R rules in Fig. 2.

To handle τ -transitions that originate from a synchronisation between an input and an output in two different locations, location labels, used to annotate transitions with events, may be pairs of locations, as employed in Fig. 3.

► **Definition 10** (location labels). A location label u is either a location ℓ or a pair of locations (ℓ_0, ℓ_1) , and we let $c(\ell_0, \ell_1) = (c\ell_0, c\ell_1)$ for $c \in \{0, 1\}$.

Events are pairs of action labels and location labels, as they appear above and below labelled transitions in Fig. 2 and 3. The role of the aliases and the usefulness of their prefix location is justified in Sec. 4.1 and 4.2. The rules SUM-L and SUM-R, and BANG rules have also been carefully designed, as explained in Sec. 4.3, 4.4 and 4.5. We will return, in those sections, to these rules to provide more detailed insight.

The only significant modification that we make to the standard syntax of applied π -calculus is that aliases have more structure than variables. Aliases, ranging over α, β, γ , are variables, extended with a, possibly empty, prefix of **0**s and **1**s representing the location of the process producing them. For convenience, we use λ, λ', \dots to range over variables used as aliases where the prefix is empty, and assume they are of a separate syntactic category from variables used for names and input binders, allowing us to drop some side conditions in rules.

We illustrate throughout the article how some transitions are derived. We sometimes apply the EXTRUDE or RES rules “in batch”, omit the id substitution, and list some hypothesis below the derivation. The derivation presented in Fig. 4 illustrates those conventions, but it also explicitly lists the occurrences of ϵ to help with readability. Indeed, this instance of ALIAS-OUT is trivial, since it turns the process on the left into an extended process with the identity substitution that records that nothing has yet been output.

Letting $P_{\text{ok}} = \nu m, n. (\bar{a}\langle m, n \rangle | m(x).[x = n]\overline{\text{ok}}\langle \text{ok} \rangle)$, we leave to the reader to convince themselves that the following transition is similarly derivable, using the PAR-L rule:

$$\text{id} | P_{\text{ok}} \xrightarrow[\mathbf{0}[\]]{\bar{a}(\mathbf{0}\lambda)} \nu m, n. (\{ \langle m, n \rangle / \mathbf{0}\lambda \} | \mathbf{0} | m(x).[x = n]\overline{\text{ok}}\langle \text{ok} \rangle).$$

$$\begin{array}{c}
\text{OUT} \\
\hline
\bar{c}\langle\langle m, n \rangle\rangle \xrightarrow[\epsilon[\epsilon]]{\bar{c}(\lambda)} \{\langle m, n \rangle / \lambda\} \mid 0 \quad n, m \# \text{fv}(\bar{c}(\lambda)) \\
\hline
\text{EXTRUDE } (\times 2) \\
\hline
\nu m. \nu n. \bar{c}\langle\langle m, n \rangle\rangle \xrightarrow[\epsilon[\epsilon]]{\bar{c}(\lambda)} \nu m. \nu n. (\{\langle m, n \rangle / \lambda\} \mid 0) \quad (\star) \\
\hline
\text{ALIAS-OUT} \\
\hline
\text{id} \mid (\nu m. \nu n. \bar{c}\langle\langle m, n \rangle\rangle) \xrightarrow[\epsilon[\epsilon]]{\bar{c}(\epsilon\lambda)} \nu m. \nu n. (\{\langle m, n \rangle / \lambda\} \mid 0) \\
\\
m, n \# \text{ran}(\text{id}) = \emptyset \quad \text{fa}(c) = \emptyset \subseteq \text{dom}(\text{id}) = \emptyset \quad \epsilon\lambda \# \text{dom}(\text{id}) = \emptyset \quad (\star)
\end{array}$$

■ **Figure 4** First, simple example of derivation.

We discuss the next transition of P_{ok} in Sect. 4.1, and the dependence of the two events in Sect. 5. Another interesting example is given by the τ -transitions of the following process.

$$P_\tau = \nu z. ((\nu x. \bar{a}\langle\langle x, z \rangle\rangle \mid \nu y. \bar{b}\langle\langle y, z \rangle\rangle) \mid (a(x_1).P \mid b(x_2).Q))$$

This process can perform two different (and, as we will discuss in Sect. 5, independent) synchronizations whose location labels are $(00[], 10[])$ and $(01[], 11[])$. The following execution sequences illustrates how our semantics gracefully handles the two (parallel) sources of extrusions of the name z without any additional machinery.

$$\begin{array}{l}
\text{id} \mid P_\tau \xrightarrow[\text{(00[], 10[])}]{\tau} \nu z. \nu x. (\text{id} \mid (0 \mid \nu y. \bar{b}\langle\langle y, z \rangle\rangle) \mid (P\{\langle x, z \rangle / x_1\} \mid b(x_2).Q)) \\
\qquad \qquad \qquad \xrightarrow[\text{(01[], 11[])}]{\tau} \nu z. \nu x. \nu y. (\text{id} \mid (0 \mid 0) \mid (P\{\langle x, z \rangle / x_1\} \mid Q\{\langle y, z \rangle / x_2\}))
\end{array}$$

The full derivation trees for the above transitions are presented in Appendix A, Fig. 6. The derivation trees for the alternative sequence of transitions below are similar.

$$\begin{array}{l}
\text{id} \mid P_\tau \xrightarrow[\text{(01[], 11[])}]{\tau} \nu z. \nu y. (\text{id} \mid (\nu x. \bar{a}\langle\langle x, z \rangle\rangle \mid 0) \mid (a(x_1).P \mid Q\{\langle y, z \rangle / x_2\})) \\
\qquad \qquad \qquad \xrightarrow[\text{(00[], 10[])}]{\tau} \nu z. \nu y. \nu x. (\text{id} \mid (0 \mid 0) \mid (P\{\langle x, z \rangle / x_1\} \mid Q\{\langle y, z \rangle / x_2\}))
\end{array}$$

We reuse this process P_τ in Sect. 4.3 to illustrate the need for equivariance.

4.1 The Modern Applied π -Calculus Avoids Disjunctive Causality

The input and output prefixes $M(x).P$ and $\bar{M}\langle N \rangle.P$, respectively, indicate the channel as a message M , which is the modern approach to the applied π -calculus handling extruded messages [1]. Looking back at $\text{id} \mid P_{\text{ok}} \xrightarrow[\square]{\bar{a}(0\lambda)} \nu m, n. (\{\langle m, n \rangle / 0\lambda\} \mid 0 \mid m(x).[x = n]\text{ok}(\text{ok}))$, note that the active substitution $\{\langle m, n \rangle / 0\lambda\}$ can be used in subsequent events. For example, we may refer to the private name m by using the message $\text{fst}(0\lambda)$, which is then instantiated with the above active substitution, as illustrated by the following derivation of a transition (assuming a message theory featuring equation $\text{fst}\langle\langle m, n \rangle\rangle =_E m$):

Referring to channels that were extruded inside messages was not possible in early versions of the applied π -calculus [2]. This approach to extrusion is important to emphasise since adopting this modern approach significantly simplifies our labelled asynchronous transition system, as we explain next.

The key problem is to define a “stable” semantic in the presence of “link causality”. This means that, if multiple output events extrude the same name, we must record which output was used when that name appears in future events. An established approach to dealing with

$$\begin{array}{c}
 \frac{\text{fst}(\langle m, n \rangle) =_E m}{m(x).[x = n]\overline{\text{ok}}(\text{ok}) \xrightarrow[\square]{\text{fst}(\langle m, n \rangle) \text{snd}(\langle m, n \rangle)} \text{id} \mid [x = n]\overline{\text{ok}}(\text{ok}) \{ \text{snd}(\langle m, n \rangle) / x \}} \text{INP} \quad \epsilon \# 0 \\
 \frac{\quad}{0 \mid m(x).[x = n]\overline{\text{ok}}(\text{ok}) \xrightarrow[\square]{\text{fst}(\langle m, n \rangle) \text{snd}(\langle m, n \rangle)} \text{id} \mid 0 \mid [\text{snd}(\langle m, n \rangle) = n]\overline{\text{ok}}(\text{ok})} \text{PAR-R} \quad (*) \\
 \frac{\quad}{\sigma \mid 0 \mid m(x).[x = n]\overline{\text{ok}}(\text{ok}) \xrightarrow[\square]{\text{fst}(0\lambda) \text{snd}(0\lambda)} \sigma \mid 0 \mid [\text{snd}(\langle m, n \rangle) = n]\overline{\text{ok}}(\text{ok})} \text{ALIAS-FREE} \quad (**) \\
 \hline
 \nu m, n. (\sigma \mid 0 \mid m(x).[x = n]\overline{\text{ok}}(\text{ok})) \xrightarrow[\square]{\text{fst}(0\lambda) \text{snd}(0\lambda)} \nu m, n. (\sigma \mid 0 \mid [\text{snd}(\langle m, n \rangle) = n]\overline{\text{ok}}(\text{ok})) \text{EXT.}
 \end{array}$$

Letting $\sigma = \{ \langle m, n \rangle / 0\lambda \}$, since $\text{fst}(0\lambda) \text{snd}(0\lambda) \sigma = \text{fst}(\langle m, n \rangle) \text{snd}(\langle m, n \rangle)$,

$$\begin{array}{l}
 \epsilon \# \text{ran}(\sigma) = \{m, n\} \quad \text{fa}(\text{fst}(0\lambda) \text{snd}(0\lambda)) = \{0\lambda\} \subseteq \text{dom}(\sigma) = \{0\lambda\} \quad (*) \\
 n, m \# \text{fv}(\text{fst}(0\lambda) \text{snd}(0\lambda)) = \emptyset \quad (**)
 \end{array}$$

this “disjunctive dependency” is to extend the labels of transitions to record explicitly the set of output events each input depends on (by recording the source and target processes of the transition) [30, Def. 2.18]. Another established approach is to represent the disjunctive link causality in an “inclusive way” [20, p. 227], that “ensures that whenever an action with a bound subject is executed, at least one extrusion of that bound name must have been already executed”, but without recording which output was the *real* extruder that influenced another event. These additional mechanisms, used in related work, are used to acknowledge the difference between two extrusion events and recognise them as separate events.

The use of aliases avoids the disjunctive dependency problem entirely. Consider the process $\nu n. (\bar{a}\langle n \rangle \mid (\bar{a}\langle n \rangle \mid n(x).P))$ for example. This process can trigger both the output events $(\bar{a}(0\lambda), 0[\square])$ and $(\bar{a}(10\lambda), 10[\square])$:

$$\begin{aligned}
 \text{id} \mid \nu n. (\bar{a}\langle n \rangle \mid (\bar{a}\langle n \rangle \mid n(x).P)) &\xrightarrow[\square]{\bar{a}(0\lambda)} \nu n. (\{n/0\lambda\} \mid 0 \mid (\bar{a}\langle n \rangle \mid n(x).P)) \\
 &\xrightarrow[\square]{\bar{a}(10\lambda)} \nu n. (\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid n(x).P))
 \end{aligned}$$

and, afterwards, only one of the input events $(0\lambda M, 11[\square])$ or $(10\lambda M, 11[\square])$ (letting $M' = M\{n/0\lambda\} \circ \{n/10\lambda\}$):

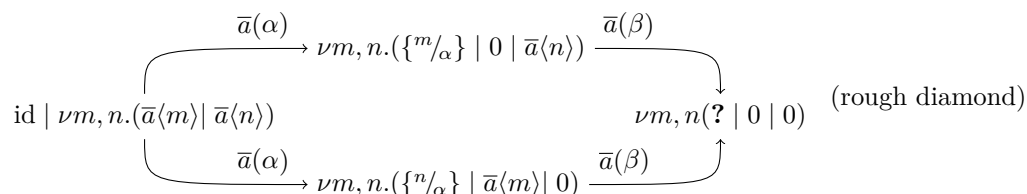
$$\nu n. (\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid n(x).P)) \begin{cases} \xrightarrow[\square]{0\lambda M} \nu n. (\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid P\{M'/x\})) \\ \xrightarrow[\square]{10\lambda M} \nu n. (\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid P\{M'/x\})) \end{cases}$$

It is clear that input $0\lambda M$ is dependent on the output originating from the first thread in location 0 , while the input with alias $10\lambda M$ is dependent on the output in location 10 . Thus there is no need to perform event splitting, since there is no ambiguity about the source of the extrusion used to refer to channel n . The derivations of the transitions producing these events are presented in Appendix A.

4.2 Applied π -Calculus With Located Aliases

Recall that located aliases are variables prefixed with a string indicating the location in which the corresponding output occurred. The idea of prefixing aliases with a string representing a location is a novelty, which is necessary for the development of our LATS, as explained here. The strings themselves are, however, inherited from earlier work on LATS for CCS [39], where such strings are used to annotate labelled transitions and are used to determine whether actions are independent.

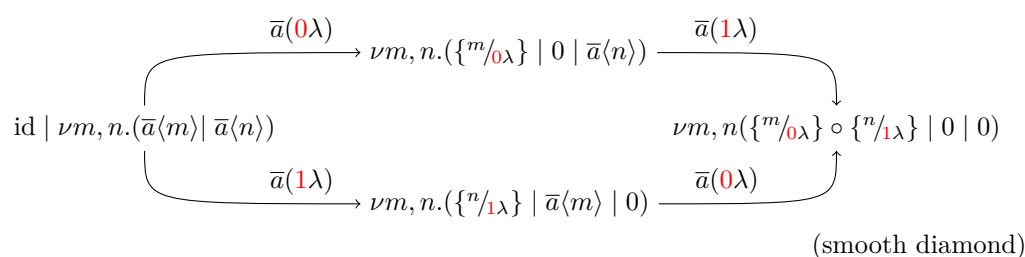
This modification to the syntax of the applied π -calculus serves two purposes: it enables us to define the independence relation only based on information from the events that label the transitions (Def. 12), and provides each location with a separate pool of aliases. Let us illustrate this latter purpose by picturing two execution sequences, with the locations omitted for now:



The challenge stems from needing to satisfy *diamond property 1*, as the two transitions are clearly independent: in the initial process $\nu m, n. (\bar{a}\langle m \rangle \mid \bar{a}\langle n \rangle)$, traditional semantics of the applied π -calculus allow us to use alias α for both initial transitions, whether the up or down transition is triggered. This freedom, unfortunately, violates *diamond property 1*, since taking the top or bottom transitions yields distinct substitutions, represented by $?$. Of course, this difference in aliasing is irrelevant provided α and β are both fresh in every process involved, hence we “localise” the generation of fresh aliases.

To “localise” the generation of fresh aliases, we add a prefix to aliases, similarly to location labels, i.e., a (possibly empty) string of **1**s and **0**s, representing where an alias originates from within the binary tree of parallel locations. Those locations become genuine parts of the aliases, so that, e.g., **10** λ and **11** λ are distinct aliases (and of course **10** λ and **10** λ' are also distinct aliases, so there is an infinite supply of aliases for each location). Since our structural operational semantics does not assume that the parallel operator is commutative or associative, the bracketing of processes composed in parallel remains stable throughout an execution; and locations do not disappear as they would if we had included $P \mid 0 \equiv P$.

Going back to our example (rough diamond), this modification allows us to satisfy *diamond property 1* (ignoring the location label from under the arrows):



4.3 The Need for Equivariance

The order in which threads can be triggered can have an effect on the order of fresh name binders. Therefore, we consider states up to *equivariance*, that is, our structural congruence (Def. 8) extends α -equivalence such that $\nu x. \nu y. A \equiv \nu y. \nu x. A$. To see why we require equivariance, consider the following diamond of output transitions:

$$\begin{array}{c}
 \begin{array}{ccc}
 & \xrightarrow{\bar{a}(0\lambda)} & \nu m.\nu k.(\{\langle m,k \rangle /_{0\lambda}\} \mid 0 \mid \nu n.\bar{a}\langle h(n) \rangle) \\
 \uparrow & & \downarrow \\
 \nu m.\nu k.\bar{a}\langle m,k \rangle \mid \nu n.\bar{a}\langle h(n) \rangle & & \nu m.\nu k.\nu n.(\{\langle m,k \rangle /_{0\lambda}\} \circ \{\langle h(n) \rangle /_{1\lambda}\} \mid 0 \mid 0) \\
 & & \equiv \\
 & & \nu n.\nu m.\nu k.(\{\langle h(n) \rangle /_{1\lambda}\} \circ \{\langle m,k \rangle /_{0\lambda}\} \mid 0 \mid 0) \\
 & & \uparrow \\
 & \xrightarrow{\bar{a}(1\lambda)} & \nu n.(\{\langle h(n) \rangle /_{1\lambda}\} \mid \nu m.\nu k.\bar{a}\langle m,k \rangle \mid 0)
 \end{array} \\
 \bar{a}(0\lambda) & & \bar{a}(1\lambda) \\
 \bar{a}(1\lambda) & & \bar{a}(0\lambda)
 \end{array}$$

(Diamond upto equivariance)

Without taking the quotient, both transitions would reach different states, and any independence relation making the consecutive events $\bar{a}(0\lambda)$ and $\bar{a}(1\lambda)$ independent could not satisfy *diamond property 2*. This is also because the substitutions are the same function, hence the order in which composition is applied does not distinguish the extended processes.

Equivariance is also essential for identifying states that are the same after independent synchronisations. For example, the two execution sequences of $\text{id} \mid P_\tau$ discussed earlier would not reach equivalent states without equivariance. More explicitly, after two τ transitions $\text{id} \mid P_\tau$ can either reach the state $\nu z.\nu y.\nu x.(\text{id} \mid (0 \mid 0) \mid (P\{\langle x,z \rangle /_{x_1}\} \mid Q\{\langle y,z \rangle /_{x_2}\}))$ or the state $\nu z.\nu x.\nu y.(\text{id} \mid (0 \mid 0) \mid (P\{\langle x,z \rangle /_{x_1}\} \mid Q\{\langle y,z \rangle /_{x_2}\}))$ depending on which synchronisation is applied first. Observe how these extended processes only differ in that the binders for x and y are swapped, and hence are the same state up to equivariance.

4.4 Distinguishing Events in Conflict

The *event determinism* property of LATS requires more care in defining the semantics for the choice operator. Since event determinism is more fine-grained than the concept of “action determinism” in works on POR [5, Definition 4.1], our work is useful there too.

Our mechanism giving a located semantics to choice is similar to *proved transtions* [12] in the sense that our locations $s[t]$ contain information not only about the parallel structure (given by s), but also about the structure of choices (given by t). For instance, in transition

$$\text{id} \mid (((P_1 + a(x).P) + P_2) \mid (P_3 + \bar{a}\langle n \rangle)) \mid P_4 \xrightarrow[\mathbf{0}(0[01],1[1])]{\tau} \text{id} \mid (P\{^n/x\} \mid 0) \mid P_4,$$

(derived in Fig. 5) the first *choice label* [01] indicates that $a(x).P$ was responsible for the input action in $(P_1 + a(x).P) + P_2$. Our choice labels diverge, however, from [39] where each location contains the source and target processes involved in that transition. We found that this established approach for CCS does not appear to work for applied π -calculus extended processes, such as $\nu m,n.(\{^m/\lambda_0\} \mid \bar{c}\langle m,n \rangle + \bar{c}\langle n,m \rangle)$, for which transitions $\xrightarrow[\overline{[\bar{c}\langle m,n \rangle]}[0]]{\bar{c}(\lambda_1)}$ and $\xrightarrow[\overline{[\bar{c}\langle n,m \rangle]}[0]]{\bar{c}(\lambda_1)}$ could be made by either branch of the choice upto equivariance, unless we add information that would break diamond properties.

Besides providing a more concise notation, our location labels unambiguously indicate which output in the non-deterministic choice was triggered in the following two co-initial transitions:

$$\nu m,n.(\{^m/\lambda_0\} \mid \bar{c}\langle m,n \rangle + \bar{c}\langle n,m \rangle) \left\{ \begin{array}{l} \xrightarrow[\mathbf{1}]{\bar{c}(\lambda_1)} \nu m,n.(\{^m/\lambda_0\} \circ \{\langle n,m \rangle /_{\lambda_1}\} \mid 0) \\ \xrightarrow[\mathbf{0}]{\bar{c}(\lambda_1)} \nu m,n.(\{^m/\lambda_0\} \circ \{\langle m,n \rangle /_{\lambda_1}\} \mid 0) \end{array} \right.$$

$$\begin{array}{c}
\frac{}{a(x).P \xrightarrow{a^n} \text{id} \mid P\{^n/x\}} \text{INP} \\
\frac{}{P_1 + a(x).P \xrightarrow{a^n} \text{id} \mid P\{^n/x\}} \text{SUM-R} \quad \frac{}{\bar{a}\langle n \rangle \xrightarrow{\bar{a}(\lambda)} \{^n/\lambda\} \mid 0} \text{OUT} \\
\frac{}{(P_1 + a(x).P) + P_2 \xrightarrow{a^n} \text{id} \mid P\{^n/x\}} \text{SUM-L} \quad \frac{}{P_3 + \bar{a}\langle n \rangle \xrightarrow{\bar{a}(\lambda)} \{^n/\lambda\} \mid 0} \text{SUM-R} \quad (*) \\
\frac{}{((P_1 + a(x).P) + P_2) \mid (P_3 + \bar{a}\langle n \rangle) \xrightarrow{\tau} \text{id} \mid P\{^n/x\} \mid 0} \text{CLOSE-R} \quad P_4 \# \emptyset \\
\frac{}{((P_1 + a(x).P) + P_2) \mid (P_3 + \bar{a}\langle n \rangle) \mid P_4 \xrightarrow{\tau} \text{id} \mid (P\{^n/x\} \mid 0) \mid P_4} \text{PAR-L} \quad (**) \\
\frac{}{\text{id} \mid ((P_1 + a(x).P) + P_2) \mid (P_3 + \bar{a}\langle n \rangle) \mid P_4 \xrightarrow{\tau} \text{id} \mid (P\{^n/x\} \mid 0) \mid P_4} \text{ALIAS-FREE} \\
\epsilon \# (P_1 + a(x).P) + P_2 \quad \epsilon \# P_3 + \bar{a}\langle n \rangle, \emptyset \quad (*) \\
\epsilon \# \text{ran}(\text{id}) = \emptyset \quad \text{fa}(\tau) = \emptyset \subseteq \text{dom}(\text{id}) = \emptyset \quad (**)
\end{array}$$

■ **Figure 5** Derivation example involving sum and synchronisation.

Moreover, identifying the events would violate *event determinism*, as the resulting extended processes are different. Notice that we do not record name binders in the events, since, unlike CSS, names move around in a way that would violate diamonds. Instead, name binders are handled by mechanisms used inside the proofs of diamond properties.

4.5 Addressing the Location of Replicated Processes

As usual, the replication operator $!$ is used to represent an unbounded number of sessions. Recording more structure than the labels when defining events prevents the rule BANG from creating image-finiteness problems that the same rule creates for an action-based LTS. Indeed, since we have *event determinism*, the image of any extended process and event is a singleton upto structural congruence. This design decision gives to every replicated process an infinite pool of explicit location names, and allows each of these locations to be triggered in any order. This is important to satisfy *diamond property 2*.

To explain this, consider the following example that use the process $\bar{a}\langle y \rangle.\bar{b}\langle z \rangle.Q = P_b$, which can perform the following transitions:

$$\text{id} \mid !P_b \xrightarrow{\bar{a}(0\lambda)} \{y/0\lambda\} \mid \bar{b}\langle z \rangle.Q \mid !P_b \xrightarrow{\bar{b}(0\lambda')} \{y/0\lambda\} \circ \{z/0\lambda'\} \mid Q \mid !P_b$$

Our definition of independence relation (Def. 11) – and, we believe, *any* reasonable location-based definition of independence – would ensure that the two events $(\bar{a}(0\lambda), 0\llbracket \rrbracket)$ and $(\bar{b}(0\lambda'), 0\llbracket \rrbracket)$ are not independent – they are treated as coming from *the same* location, even if that location “did not exist” when P_b started its execution. Hence *diamond property 2* cannot apply and these events cannot permute, as expected. This mechanism echoes e.g., the dependency relation that can be developed to accommodate replication for CCS [23].

In contrast, consider the following transitions, also originating from the same process $!P_b$:

$$\begin{array}{c}
\bar{a}(0\lambda) \xrightarrow{0\llbracket \rrbracket} \{y/0\lambda\} \mid \bar{b}\langle z \rangle.Q \mid !P_b \xrightarrow{\bar{a}(10\lambda)} \{y/10\lambda\} \mid \bar{b}\langle z \rangle.Q \mid !P_b \\
\bar{a}(10\lambda) \xrightarrow{10\llbracket \rrbracket} \{y/10\lambda\} \mid P_b \mid (\bar{b}\langle z \rangle.Q \mid !P_b) \xrightarrow{\bar{a}(0\lambda)} \{y/0\lambda\} \mid \bar{b}\langle z \rangle.Q \mid !P_b
\end{array}$$

(Diamond with a bang)

The events on the left-hand side of the diamond above are expected to be independent. Fortunately, by triggering BANG twice, we can permute these transitions as required by *diamond property 2*. That is, the right side of the above diamond exists.

An alternative solution could be to use explicit names to label locations, and partially ordering those labels to reflect the hierarchy of locations, and then minting fresh names for each transition of a replication [13, 44]. Using opaque names as locations, however, would have forced us to record them in the syntax of processes, that would have become e.g., of the form $\ell_1 :: \bar{b}\langle z \rangle.Q \mid (\ell_2 :: \bar{b}\langle z \rangle.Q \mid !P)$ [12]. The intent is however the same.

5 The Independence Relation and the Main Result

In this section, we define what it means for two events to be independent. To do this, firstly, we define structural independence, which ensures that two events occur in different locations by checking that it is not the case that one location prefix is a prefix (as a string) of the other event's location prefix.

► **Definition 11** (structural independence). *Define \mathcal{Loc} on location labels such that $\mathcal{Loc}(\ell) = \{\ell\}$ and $\mathcal{Loc}(\ell_0, \ell_1) = \{\ell_0, \ell_1\}$. The structural independence relation I_ℓ on location labels is the least relation defined by $u_0 I_\ell u_1$ whenever for all locations $\ell_0 \in \mathcal{Loc}(u_0)$ and $\ell_1 \in \mathcal{Loc}(u_1)$, there exist a string $s \in \{0, 1\}^*$ and locations ℓ'_0, ℓ'_1 , such that either: $\ell_0 = s0\ell'_0$ and $\ell_1 = s1\ell'_1$; or $\ell_0 = s1\ell'_0$ and $\ell_1 = s0\ell'_1$.*

For example, consider the locations of the four output events in $\bar{a}\langle a \rangle \mid \bar{b}\langle b \rangle.(\bar{c}\langle c \rangle \mid \bar{d}\langle d \rangle)$. The output on channel a (location prefix 0) is structurally independent from all other outputs. The output on channel b (location prefix 1) is not structurally independent with respect to the outputs on channels c or d ; which will have location prefixes 10 and 11 respectively, both with 1 as a common prefix. However, the outputs on channel c and d are independent, since neither 10 nor 11 are prefixes of each other.

Independence on events in addition detects whether an output influences another action. That is, in addition to structural independence, we have link independence.

► **Definition 12** (independence of events). *Events $e = (\pi, u)$ are pairs of labels π and location labels u . The independence relation I on events is the least symmetric relation such that $(\pi_0, u_0) I (\pi_1, u_1)$ whenever $u_0 I_\ell u_1$ and if $\pi_0 = \overline{M}(\alpha)$, then $\alpha \# \pi_1$.*

Remember that P_{ok} from Sect. 4 and 4.1 can trigger the event $(\bar{a}(0\lambda), 0)$ followed by $(\text{fst}(0\lambda) \text{snd}(0\lambda), 1)$. Even if the locations are independent (as $0 I_\ell 1$), the two events are not independent, since 0λ is not fresh in $(\text{fst}(0\lambda) \text{snd}(0\lambda))$.

► **Theorem 13.** *The structural operational semantics in Fig. 2 and 3 generates a labelled asynchronous transition system with respect to the independence relation I from Def. 12, i.e., it respects Def. 1 – 3, where events are the pairs of action and location labels (π, u) , as they appear on the labels of transitions, and states are extended processes modulo the structural congruence from Def. 8. [see proof in Appendix B]*

Link independence is only required to permute an output event followed by an independent event, when establishing *diamond property 2*. The main intricacy compared to CCS is to ensure that the name restrictions occurring along any common prefix of two independent transitions are handled correctly – this is how the parallel extrusion problem in related theories manifests itself in this theory. We handle this problem entirely within the proof, via variables accumulated in a function that picks out the component of a process corresponding to a location prefix, rather than within the semantics as in related work [30, 50].

6 Related Work

The earliest papers on partial-order reduction for security are not working with the applied π -calculus, but rely on constructing execution DAGs (or Mazurkiewicz traces) recording all input-output dependencies; as a result, the protocols considered in [18, 21, 27, 38] are essentially threads of inputs and outputs, disregarding channels.

Our paper is closer to more recent approaches to POR for the applied π -calculus [6, 7, 8]. However, one limitation of these works is that they require processes to be of a particular form. In that related line of work, structural independence is based on the channel of an input and output action, thus considering events structurally independent only when they employ distinct channels. Thus, their scope is restricted to processes in which every location is a single thread of sequential actions (e.g., cannot spawn parallel threads in a location) and **if-then-else** branches that employ a unique channel. While many finite protocol problems can be formulated with a fixed number of threads, each employing separate channels, this is still a significant restriction. In contrast, we base the structural independence on the components' addresses, which allows us to consider the “full” applied π -calculus.

For modelling infinite protocols (or protocols where the same, or multiple, actors can engage in multiple parallel sessions) one normally uses replication, and thus parallel extrusion appears naturally. Some of the above related works [7] approximate replication by creating fresh channels manufactured every time a parallel component is created. However, since these works do not have mechanisms for dealing with disjunctive causality (which in these settings is required because of the use of a different mechanism for extrusion, similar to the standard π -calculus), channel extrusion is only supported for processes where parallel extrusion of channels never occurs. In contrast, our approach, where channels are extruded like any other message and aliases resolve the aforementioned disjunctive causality problem, supports all forms of processes including replication and parallel extrusion.

A further difference compared to the above work concerns τ transitions. In the related work we are discussing, there is not enough information to determine whether two τ transitions are independent, and hence such parts of a protocol's behaviour cannot be considered for POR optimisation. In our semantics, we resolve this problem using our structural independence relation based on the recorded locations responsible for the input and output actions involved in a τ transition. Our solution simply lifts classic work on CCS [39] to this security setting, while taking care to handle parallel extrusion correctly (recall the P_τ example from Sect. 4).

All the above work does not follow the non-interleaving tradition in the sense that they do not feature *diamond property 1*. It is possible in the semantics of [6, 8, 17] that two transitions that we deem independent and are enabled from some process will disable each other, since their executions change the substitution on which the other depends. This is the key problem our located aliases address. The only diamond property that related approaches obtain (and for a limited subset of processes, as explained above) is the “reordering of sequential independent transitions”, which is our *diamond property 2*. However, this property alone can be achieved without located aliases since, anyway, two sequential outputs would be named differently by the regular constraints of the applied π -calculus, which simply ensure that an extruded alias is globally fresh.

Event determinism can be achieved in works such as [7, 8] only for choice of the type **if-then-else**. By borrowing from proved transitions [12], our events record the precise branch of a general non-deterministic choice from which they originate, thereby achieving event-determinism for all types of processes. This problem has been acknowledged [6], and an

alternative approach to POR using *sleep sets* has been proposed. This approach essentially defines independence in terms of events satisfying our *diamond property 1*, but since the semantics they employ allows two concurrently enabled outputs on the same channel to use the same label, such events would incorrectly be considered not independent.

We see a good opportunity in adopting (concepts from) our semantics in the settings and tools of the above papers. This possibility has been one of our goals all along, and guided our decisions to consider a syntax very close to the standard applied π -calculus and to define a structural operational semantics that just extends previous semantics for applied π [32]. Even the choice of LATS was guided by our wish to stay within the realm of transition systems, yet to go over from interleaving to non-interleaving semantics. Hence, we expect it to be possible to upgrade tools from the above mentioned papers, so that they may fully support POR for all processes.

7 Conclusion

The work we build on incorporates elements of the modern applied π -calculus [1] – aliases for extrusion of channels as messages – into a structural operational semantics [31, 32]. Our semantics in Fig. 2 and 3 transforms this established semantics into a non-interleaving structural operational semantics by recording on transitions also the location from which an event originates as well as the location from where an alias representing an output originates. The former is a standard device, coming from non-interleaving semantics for CCS [12, 39] and π -calculus [30], while the latter “located alias” is the key technical innovation required to ensure that our structural operational semantics defines a LATS (Theorem 13). In this way we obtain a genuine operational semantics, with a remarkably simple independence relation (Def. 12) for such a powerful calculus. Moreover, this paper can also be seen as proposing LATS as the semantic objects for applied π -calculi, instead of transition systems, so as to bridge other non-interleaving models to which LATS have been related in the literature.

Because LATS have been shown [28] to be exactly the higher-dimensional automata of dimension 2, we can reuse the definitions for higher-dimensional automata of the classical concurrency bisimulations (i.e., of ST-, history preserving-, and hereditary history preserving-bisimulations) for LATS [47]. Moreover, through relations of LATS with configuration structures [49] and event structures [51] we can reuse also other concurrency bisimulations [48]. In related work, partial-order semantics have been employed in tools for optimising verification of equivalence properties [17]. Making precise the connection between the equivalences employed in such tools and the above non-interleaving equivalences for LATS is future work.

References

- 1 Martín Abadi, Bruno Blanchet, and Cédric Fournet. The applied pi calculus: Mobile values, new names, and secure communication. *J. ACM*, 65(1):1:1–1:41, 2018. doi:10.1145/3127586.
- 2 Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL’01)*, pages 104–115, 2001. doi:10.1145/360204.360213.
- 3 Reynald Affeldt and Naoki Kobayashi. Partial order reduction for verification of spatial properties of pi-calculus processes. *Electron. Notes Theor. Comput. Sci.*, 128(2):151–168, 2004. Proceedings of the 11th International Workshop on Expressiveness in Concurrency (EXPRESS 2004). doi:10.1016/j.entcs.2004.11.034.
- 4 Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing unlinkability and anonymity using the applied pi calculus. In *23rd IEEE Computer Security Foundations Symposium (CSF’10)*, pages 107–121, 2010. doi:10.1109/CSF.2010.15.

- 5 David Baelde. *Contributions à la Vérification des Protocoles Cryptographiques*. Habilitation à diriger des recherches, Université Paris-Saclay, February 2021. URL: http://www.lsv.fr/~baelde/hdr/habilitation_baelde.pdf.
- 6 David Baelde, Stéphanie Delaune, and Lucca Hirschi. POR for security protocol equivalences – beyond action-determinism. In Javier López, Jianying Zhou, and Miguel Soriano, editors, *Computer Security – 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I*, volume 11098 of *LNCS*, pages 385–405. Springer, 2018. doi:10.1007/978-3-319-99073-6_19.
- 7 David Baelde, Stéphanie Delaune, and Lucca Hirschi. Partial order reduction for security protocols. In Luca Aceto and David de Frutos-Escrig, editors, *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1-4, 2015*, volume 42 of *LIPICs*, pages 497–510. Schloss Dagstuhl, 2015. doi:10.4230/LIPICs.CONCUR.2015.497.
- 8 David Baelde, Stéphanie Delaune, and Lucca Hirschi. A reduced semantics for deciding trace equivalence. *Log. Meth. Comput. Sci.*, 13(2), 2017. doi:10.23638/LMCS-13(2:8)2017.
- 9 Marek A. Bednarczyk. *Categories of asynchronous systems*. PhD thesis, University of Sussex, Brighton, UK, 1988.
- 10 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. doi:10.1561/33000000004.
- 11 Michele Boreale and Davide Sangiorgi. A fully abstract semantics for causality in the π -calculus. *Acta Inform.*, 35(5):353–400, 1998.
- 12 Gérard Boudol and Ilaria Castellani. A non-interleaving semantics for CCS based on proved transitions. *Fund. Inform.*, 11:433–452, 1988.
- 13 Gérard Boudol, Ilaria Castellani, Matthew Hennessy, and Astrid Kiehn. Observing localities. *Theor. Comput. Sci.*, 114(1):31–61, 1993. doi:10.1016/0304-3975(93)90152-J.
- 14 Sergiu Bursuc, Christian Johansen, and Shiwei Xu. Automated verification of dynamic root of trust protocols. In Matteo Maffei and Mark Ryan, editors, *International Conference on Principles of Security and Trust*, volume 10204 of *LNCS*, pages 95–116. Springer, 2017. doi:10.1007/978-3-662-54455-6_5.
- 15 Georgia Carabetta, Pierpaolo Degano, and Fabio Gadducci. CCS semantics via proved transition systems and rewriting logic. In Claude Kirchner and Hélène Kirchner, editors, *1998 International Workshop on Rewriting Logic and its Applications, WRLA 1998, Abbaye des Prémontrés at Pont-à-Mousson, France, September 1998*, volume 15 of *Electron. Notes Theor. Comput. Sci.*, pages 369–387. Elsevier, 1998. doi:10.1016/S1571-0661(05)80023-4.
- 16 Rohit Chadha, Vincent Cheval, Ștefan Ciobăcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. *ACM Trans. Comput. Log.*, 17(4):23:1–23:32, September 2016. doi:10.1145/2926715.
- 17 Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. Exploiting symmetries when proving equivalence properties for security protocols. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 905–922. ACM, 2019. doi:10.1145/3319535.3354260.
- 18 Edmund M. Clarke, Somesh Jha, and Wilfredo R. Marrero. Efficient verification of security protocols using partial-order reductions. *Int. J. Softw. Tools Technol. Transf.*, 4(2):173–188, 2003. doi:10.1007/s10009-002-0103-4.
- 19 Véronique Cortier, Antoine Dallon, and Stéphanie Delaune. SAT-Equiv: An efficient tool for equivalence properties. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 481–494, August 2017. doi:10.1109/CSF.2017.15.
- 20 Silvia Crafa, Daniele Varacca, and Nobuko Yoshida. Event structure semantics of parallel extrusion in the pi-calculus. In Lars Birkedal, editor, *Foundations of Software Science and Computational Structures – 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 – April 1, 2012. Proceedings*, volume 7213 of *LNCS*, pages 225–239. Springer, 2012. doi:10.1007/978-3-642-28729-9_15.

- 21 Cas J. F. Cremers and Sjouke Mauw. Checking secrecy by means of partial order reduction. In Daniel Amyot and Alan W. Williams, editors, *System Analysis and Modeling, 4th International SDL and MSC Workshop, SAM 2004, Ottawa, Canada, June 1-4, 2004, Revised Selected Papers*, volume 3319 of *LNCS*, pages 171–188. Springer, 2004. doi:10.1007/978-3-540-31810-1_12.
- 22 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 *LNCS*, pages 223–240. Springer, 2015. doi:10.1007/978-3-319-25150-9_14.
- 23 Pierpaolo Degano, Fabio Gadducci, and Corrado Priami. Causality and replication in concurrent processes. In Manfred Broy and Alexandre V. Zamulin, editors, *Perspectives of Systems Informatics, 5th International Andrei Ershov Memorial Conference, PSI 2003, Akademgorodok, Novosibirsk, Russia, July 9-12, 2003, Revised Papers*, volume 2890 of *LNCS*, pages 307–318. Springer, 2003. doi:10.1007/978-3-540-39866-0_30.
- 24 Pierpaolo Degano and Corrado Priami. Proved trees. In Werner Kuich, editor, *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings*, volume 623 of *LNCS*, pages 629–640. Springer, 1992. doi:10.1007/3-540-55719-9_110.
- 25 Pierpaolo Degano and Corrado Priami. Non-interleaving semantics for mobile processes. *Theor. Comput. Sci.*, 216(1-2):237–270, 1999. doi:10.1016/S0304-3975(99)80003-6.
- 26 Pierpaolo Degano and Corrado Priami. Enhanced operational semantics. *ACM Comput. Surv.*, 33(2):135–176, 2001. doi:10.1145/384192.384194.
- 27 Wan Fokkink, Mohammad Torabi Dashti, and Anton Wijs. Partial order reduction for branching security protocols. In *2010 10th International Conference on Application of Concurrency to System Design*, pages 191–200. IEEE, 2010. doi:10.1109/ACSD.2010.19.
- 28 Éric Goubault. Labelled cubical sets and asynchronous transition systems: an adjunction. In *Preliminary Proceedings CMCIM'02*, 2002. URL: <http://www.lix.polytechnique.fr/~goubault/papers/cmcm02.ps.gz>.
- 29 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/>.
- 30 Thomas Troels Hildebrandt, Christian Johansen, and Håkon Normann. A stable non-interleaving early operational semantics for the pi-calculus. *J. Log. Algebr. Methods Program.*, 104:227–253, 2019. doi:10.1016/j.jlamp.2019.02.006.
- 31 Ross Horne and Sjouke Mauw. Discovering epassport vulnerabilities using bisimilarity. *Log. Meth. Comput. Sci.*, 17(2):24, 2021. doi:10.23638/LMCS-17(2:24)2021.
- 32 Ross Horne, Sjouke Mauw, and Semen Yurkov. Compositional analysis of protocol equivalence in the applied π -calculus using quasi-open bisimilarity. In Antonio Cerone and Peter Csaba Ölveczky, editors, *Theoretical Aspects of Computing – ICTAC 2021 – 18th International Colloquium, Virtual Event, Nur-Sultan, Kazakhstan, September 8-10, 2021, Proceedings*, volume 12819 of *LNCS*, pages 235–255. Springer, 2021. doi:10.1007/978-3-030-85315-0_14.
- 33 Ross Horne, Sjouke Mauw, and Semen Yurkov. Unlinkability of an improved key agreement protocol for emv 2nd gen payments. In Stefano Calzavara, editor, *IEEE Computer Security Foundations Symposium 2022 (CSF2022), August, 2022, Haifa, Israel, 2022*. to appear. URL: <https://satoss.uni.lu/members/ross/pdf/emv.pdf>.
- 34 Lalita Jategaonkar Jagadeesan and Radha Jagadeesan. Causality and true concurrency: A data-flow analysis of the pi-calculus. In *International Conference on Algebraic Methodology and Software Technology*, pages 277–291. Springer, 1995.
- 35 Steve Kremer and Robert Künnemann. Automated analysis of security protocols with global state. *JCS*, 24(5):583–616, 2016. doi:10.3233/JCS-160556.
- 36 Steve Kremer and Mark Ryan. Analysis of an electronic voting protocol in the applied pi calculus. In Mooly Sagiv, editor, *Programming Languages and Systems: 14th European Symposium on Programming (ESOP'05 at ETAPS'05)*, volume 3444 of *LNCS*, pages 186–200. Springer-Verlag, 2005. doi:10.1007/978-3-540-31987-0_14.

- 37 Ivan Lanese, Iain C. C. Phillips, and Irek Ulidowski. An axiomatic approach to reversible computation. 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 *LNCS*, pages 442–461. Springer, 2020. doi:10.1007/978-3-030-45231-5_23.
- 38 Sebastian Mödersheim, Luca Viganò, and David Basin. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *J. Comput. Secur.*, 18(4):575–618, 2010. doi:10.3233/JCS-2009-0351.
- 39 Madhavan Mukund and Mogens Nielsen. CCS, Location and Asynchronous Transition Systems. In R. K. Shyamasundar, editor, *Foundations of Software Technology and Theoretical Computer Science, 12th Conference, New Delhi, India, December 18-20, 1992, Proceedings*, volume 652 of *LNCS*, pages 328–341. Springer, 1992. doi:10.1007/3-540-56287-7_116.
- 40 Kirstin Peters, Jens-Wolfhard Schicke-Uffmann, Ursula Goltz, and Uwe Nestmann. Synchrony versus causality in distributed systems. *Math. Struct. Comput. Sci.*, 26(8):1459–1498, 2016. doi:10.1017/S0960129514000644.
- 41 Carl Adam Petri. Fundamentals of a theory of asynchronous information flow. In *Information Processing, Proceedings of the 2nd IFIP Congress 1962, Munich, Germany, August 27 – September 1, 1962*, pages 386–390. North-Holland, 1962.
- 42 Iain Phillips and Irek Ulidowski. Reversibility and models for concurrency. *Electron. Notes Theor. Comput. Sci.*, 192(1):93–108, 2007. doi:10.1016/j.entcs.2007.08.018.
- 43 Mark Dermot Ryan and Ben Smyth. Applied pi calculus. In Véronique Cortier and Steve Kremer, editors, *Formal Models and Techniques for Analyzing Security Protocols*, volume 5 of *Cryptology and Information Security Series*, pages 112–142. IOS Press, 2011. doi:10.3233/978-1-60750-714-7-112.
- 44 Davide Sangiorgi. A theory of bisimulation for the pi-calculus. *Acta Inform.*, 33(1):69–97, 1996. doi:10.1007/s002360050036.
- 45 Michael W. Shields. Concurrent machines. *Comput. J.*, 28(5):449–465, 1985. doi:10.1093/comjnl/28.5.449.
- 46 Alwen Tiu, Nam Nguyen, and Ross Horne. Spec: An equivalence checker for security protocols. In Atsushi Igarashi, editor, *14th Asian Symposium on Programming Languages and Systems (APLAS’16)*, volume 10017 of *LNCS*, pages 87–95. Springer, 2016. doi:10.1007/978-3-319-47958-3_5.
- 47 Rob van Glabbeek. On the Expressiveness of Higher Dimensional Automata. *Theor. Comput. Sci.*, 356(3):265–290, 2006.
- 48 Rob van Glabbeek and Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Inform.*, 37(4/5):229–327, 2001. doi:10.1007/s002360000041.
- 49 Rob van Glabbeek and Gordon D. Plotkin. Configuration structures, event structures and petri nets. *Theor. Comput. Sci.*, 410(41):4111–4159, 2009. doi:10.1016/j.tcs.2009.06.014.
- 50 Daniele Varacca and Nobuko Yoshida. Typed event structures and the linear pi-calculus. *Theor. Comput. Sci.*, 411(19):1949–1973, 2010. doi:10.1016/j.tcs.2010.01.024.
- 51 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 Derivations for selected example transitions

We present the derivations of the transitions used to illustrate stability in Sect. 4.1. We are explicit, in the input transitions, about the source of the extruded name.

$$\begin{array}{c}
 \frac{}{\bar{a}(n) \xrightarrow[\perp]{\bar{a}(\lambda)} \{n/\lambda\} \mid 0} \text{OUT} \quad \epsilon \# \bar{a}(n) \mid n(x).P \\
 \frac{}{\bar{a}(n) \mid (\bar{a}(n) \mid n(x).P) \xrightarrow[\perp]{\bar{a}(\lambda)} \{n/\lambda\} \mid 0 \mid (\bar{a}(n) \mid n(x).P)} \text{PAR-L} \quad n \# \text{fv}(\bar{a}(\lambda)) \\
 \frac{}{\nu n.(\bar{a}(n) \mid (\bar{a}(n) \mid n(x).P)) \xrightarrow[\perp]{\bar{a}(\lambda)} \nu n.(\{n/\lambda\} \mid 0 \mid (\bar{a}(n) \mid n(x).P))} \text{EXTRUDE} \quad (\star) \\
 \frac{}{\text{id} \mid \nu n.(\bar{a}(n) \mid (\bar{a}(n) \mid n(x).P)) \xrightarrow[\perp]{\bar{a}(0\lambda)} \nu n.(\{n/0\lambda\} \mid 0 \mid (\bar{a}(n) \mid n(x).P))} \text{ALIAS-OUT} \\
 n \# \text{ran}(\text{id}) = \emptyset \quad \text{fa}(a) = \emptyset \subseteq \text{dom}(\text{id}) = \emptyset \quad 0\lambda \# \text{dom}(\text{id}) = \emptyset \quad (\star)
 \end{array}$$

$$\begin{array}{c}
 \frac{}{\bar{a}(n) \xrightarrow[\perp]{\bar{a}(\lambda)} \{n/\lambda\} \mid 0} \text{OUT} \quad \epsilon \# n(x).P \\
 \frac{}{\bar{a}(n) \mid n(x).P \xrightarrow[\perp]{\bar{a}(\lambda)} \{n/\lambda\} \mid 0 \mid n(x).P} \text{PAR-L} \quad \epsilon \# 0 \\
 \frac{}{0 \mid (\bar{a}(n) \mid n(x).P) \xrightarrow[\perp]{\bar{a}(\lambda)} \{n/\lambda\} \mid 0 \mid (0 \mid n(x).P)} \text{PAR-R} \quad (\star) \\
 \frac{}{\{n/0\lambda\} \mid 0 \mid (\bar{a}(n) \mid n(x).P) \xrightarrow[\perp]{\bar{a}(10\lambda)} \{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (\bar{a}(n) \mid n(x).P)} \text{ALIAS-OUT} \quad n \# \text{fv}(\bar{a}(10\lambda)) \\
 \frac{}{\nu n.(\{n/0\lambda\} \mid 0 \mid (\bar{a}(n) \mid n(x).P)) \xrightarrow[\perp]{\bar{a}(10\lambda)} \nu n.(\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid n(x).P))} \text{EXTRUDE} \\
 \epsilon \# \text{ran}(\{n/0\lambda\}) = n \quad \text{fa}(a) = \emptyset \subseteq \text{dom}(\{n/0\lambda\}) = 0\lambda \quad 10\lambda \# \text{dom}(\{n/0\lambda\}) = 0\lambda \quad (\star)
 \end{array}$$

$$\begin{array}{c}
 \frac{}{n(x).P \xrightarrow[\perp]{nM'} \text{id} \mid P\{M'/x\}} \text{INP} \quad \epsilon \# 0 \\
 \frac{}{0 \mid n(x).P \xrightarrow[\perp]{nM'} \text{id} \mid 0 \mid P\{M'/x\}} \text{PAR-R} \quad \epsilon \# 0 \\
 \frac{}{0 \mid (0 \mid n(x).P) \xrightarrow[\perp]{nM'} \text{id} \mid 0 \mid (0 \mid P\{M'/x\})} \text{PAR-R} \quad (\star) \\
 \frac{}{\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid n(x).P) \xrightarrow[\perp]{0\lambda M} \{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid P\{M'/x\})} \text{ALIAS-FREE} \quad n \# \text{fv}(0\lambda M) \\
 \frac{}{\nu n.(\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid n(x).P)) \xrightarrow[\perp]{0\lambda M} \nu n.(\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid P\{M'/x\}))} \text{RES}
 \end{array}$$

$$\text{Letting } \sigma = \{n/0\lambda\} \circ \{n/10\lambda\} \text{ and } (0\lambda M)\sigma = nM', \\
 \epsilon \# \text{ran}(\sigma) = \{n\} \quad \text{fa}(0\lambda M) = \{0\lambda\} \subseteq \text{dom}(\sigma) = \{0\lambda, 10\lambda\} \quad (\star)$$

$$\begin{array}{c}
 \frac{}{n(x).P \xrightarrow[\perp]{nM'} \text{id} \mid P\{M'/x\}} \text{INP} \quad \epsilon \# 0 \\
 \frac{}{0 \mid n(x).P \xrightarrow[\perp]{nM'} \text{id} \mid 0 \mid P\{M'/x\}} \text{PAR-R} \quad \epsilon \# 0 \\
 \frac{}{0 \mid (0 \mid n(x).P) \xrightarrow[\perp]{nM'} \text{id} \mid 0 \mid (0 \mid P\{M'/x\})} \text{PAR-R} \quad (\star) \\
 \frac{}{\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid n(x).P) \xrightarrow[\perp]{10\lambda M} \{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid P\{M'/x\})} \text{ALIAS-FREE} \quad n \# \text{fv}(10\lambda M) \\
 \frac{}{\nu n.(\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid n(x).P)) \xrightarrow[\perp]{10\lambda M} \nu n.(\{n/0\lambda\} \circ \{n/10\lambda\} \mid 0 \mid (0 \mid P\{M'/x\}))} \text{RES}
 \end{array}$$

$$\text{Letting } \sigma = \{n/0\lambda\} \circ \{n/10\lambda\} \text{ and } (10\lambda M)\sigma = nM', \\
 \epsilon \# \text{ran}(\sigma) = \{n\} \quad \text{fa}(10\lambda M) = \{10\lambda\} \subseteq \text{dom}(\sigma) = \{0\lambda, 10\lambda\} \quad (\star)$$

$$\begin{array}{c}
\frac{\text{OUT}}{\bar{a}\langle(x, z)\rangle \xrightarrow{\bar{a}(\lambda)} \{\langle(x, z)/\lambda\rangle \mid 0\}} \quad x \# \text{fv}(\bar{a}(\lambda)) \\
\frac{\text{EXTRUDE}}{\nu x.\bar{a}\langle(x, z)\rangle \xrightarrow{\bar{a}(\lambda)} \nu x.(\{\langle(x, z)/\lambda\rangle \mid 0\})} \quad x \# \nu y.\bar{b}\langle(y, z)\rangle \\
\frac{\text{PAR-L}}{\nu x.\bar{a}\langle(x, z)\rangle \mid \nu y.\bar{b}\langle(y, z)\rangle \xrightarrow{\bar{a}(\lambda)} \nu x.(\{\langle(x, z)/\lambda\rangle \mid 0\} \mid \nu y.\bar{b}\langle(y, z)\rangle)} \quad \frac{\text{PAR-L}}{a(x_1).P \xrightarrow{a(x_2)} \text{id} \mid P\{\langle(x, z)/x_1\rangle\}} \quad \text{INP} \\
\frac{\text{PAR-L}}{\nu x.\bar{a}\langle(x, z)\rangle \mid \nu y.\bar{b}\langle(y, z)\rangle \xrightarrow{\bar{a}(\lambda)} \nu x.(\{\langle(x, z)/\lambda\rangle \mid 0\} \mid \nu y.\bar{b}\langle(y, z)\rangle)} \quad \frac{\text{PAR-L}}{a(x_1).P \mid b(x_2).Q \xrightarrow{a(x_2)} \text{id} \mid P\{\langle(x, z)/x_1\rangle\} \mid b(x_2).Q} \quad \text{PAR-L} \\
\frac{\text{CLOSE-L}}{(\nu x.\bar{a}\langle(x, z)\rangle \mid \nu y.\bar{b}\langle(y, z)\rangle) \mid (a(x_1).P \mid b(x_2).Q) \xrightarrow{\tau} \nu x.(\text{id} \mid 0 \mid \nu y.\bar{b}\langle(y, z)\rangle) \mid (P\{\langle(x, z)/x_1\rangle\} \mid b(x_2).Q))} \quad \text{CLOSE-L} \\
\frac{\text{EXT.}}{\nu z.((\nu x.\bar{a}\langle(x, z)\rangle \mid \nu y.\bar{b}\langle(y, z)\rangle) \mid (a(x_1).P \mid b(x_2).Q)) \xrightarrow{\tau} \nu z.\nu x.(\text{id} \mid 0 \mid \nu y.\bar{b}\langle(y, z)\rangle) \mid (P\{\langle(x, z)/x_1\rangle\} \mid b(x_2).Q))} \quad \text{EXT.} \\
\frac{\text{ALIAS-FREE}}{\text{id} \mid \nu z.((\nu x.\bar{a}\langle(x, z)\rangle \mid \nu y.\bar{b}\langle(y, z)\rangle) \mid (a(x_1).P \mid b(x_2).Q)) \xrightarrow{\tau} \nu z.\nu x.(\text{id} \mid 0 \mid \nu y.\bar{b}\langle(y, z)\rangle) \mid (P\{\langle(x, z)/x_1\rangle\} \mid b(x_2).Q))} \quad \text{ALIAS-FREE} \\
\frac{\text{PAR-R}}{x \# a(x_1).P \mid b(x_2).Q \quad \epsilon \# \nu x.\bar{a}\langle(x, z)\rangle \mid \nu y.\bar{b}\langle(y, z)\rangle, x} \quad (*) \\
\frac{\text{PAR-R}}{z, x \# \text{ran}(\text{id}) = \emptyset \quad \text{fa}(\tau) = \emptyset \subseteq \text{dom}(\text{id}) = \emptyset} \quad (**) \\
\frac{\text{OUT}}{\bar{b}\langle(y, z)\rangle \xrightarrow{\bar{b}(\lambda)} \{\langle(y, z)/\lambda\rangle \mid 0\}} \quad y \# \text{fv}(\bar{b}(\lambda)) \\
\frac{\text{EXTRUDE}}{\nu y.\bar{b}\langle(y, z)\rangle \xrightarrow{\bar{b}(\lambda)} \nu y.(\{\langle(y, z)/\lambda\rangle \mid 0\})} \quad y \# 0 \\
\frac{\text{PAR-R}}{0 \mid \nu y.\bar{b}\langle(y, z)\rangle \xrightarrow{\bar{b}(\lambda)} \nu y.(\{\langle(y, z)/\lambda\rangle \mid 0\} \mid 0)} \quad \frac{\text{PAR-R}}{b(x_2).Q \xrightarrow{b(y, z)} \text{id} \mid Q\{\langle(y, z)/x_2\rangle\}} \quad \text{INP} \\
\frac{\text{PAR-R}}{0 \mid \nu y.\bar{b}\langle(y, z)\rangle \xrightarrow{\bar{b}(\lambda)} \nu y.(\{\langle(y, z)/\lambda\rangle \mid 0\} \mid 0)} \quad \frac{\text{PAR-R}}{P\{\langle(x, z)/x_1\rangle\} \mid b(x_2).Q \xrightarrow{b(y, z)} \text{id} \mid P\{\langle(x, z)/x_1\rangle\} \mid Q\{\langle(y, z)/x_2\rangle\}} \quad \text{PAR-R} \\
\frac{\text{CLOSE-L}}{(0 \mid \nu y.\bar{b}\langle(y, z)\rangle) \mid (P\{\langle(x, z)/x_1\rangle\} \mid b(x_2).Q)) \xrightarrow{\tau} \nu y.(\text{id} \mid 0 \mid 0) \mid (P\{\langle(x, z)/x_1\rangle\} \mid Q\{\langle(y, z)/x_2\rangle\})} \quad \text{CLOSE-L} \\
\frac{\text{ALIAS-FREE}}{\text{id} \mid 0 \mid \nu y.\bar{b}\langle(y, z)\rangle \mid (P\{\langle(x, z)/x_1\rangle\} \mid b(x_2).Q)) \xrightarrow{\tau} \nu y.(\text{id} \mid 0 \mid 0) \mid (P\{\langle(x, z)/x_1\rangle\} \mid Q\{\langle(y, z)/x_2\rangle\})} \quad \text{ALIAS-FREE} \\
\frac{\text{RES.}}{\nu z.\nu x.(\text{id} \mid 0 \mid \nu y.\bar{b}\langle(y, z)\rangle) \mid (P\{\langle(x, z)/x_1\rangle\} \mid b(x_2).Q)) \xrightarrow{\tau} \nu z.\nu x.\nu y.(\text{id} \mid 0 \mid 0) \mid (P\{\langle(x, z)/x_1\rangle\} \mid Q\{\langle(y, z)/x_2\rangle\})} \quad \text{RES.} \\
\frac{\text{PAR-R}}{y \# P\{\langle(x, z)/x_1\rangle\} \mid b(x_2).Q \quad \epsilon \# (0 \mid \nu y.\bar{b}\langle(y, z)\rangle), y} \quad (*) \\
\frac{\text{PAR-R}}{y \# \text{ran}(\text{id}) = \emptyset \quad \text{fa}(\tau) = \emptyset \subseteq \text{dom}(\text{id}) = \emptyset} \quad (**)
\end{array}$$

Figure 6 Derivation of the first execution sequence for P_τ .

B Proof outline: event determinism, decomposition, & composition

We prove each condition *event determinism*, *diamond property 1* and *diamond property 2* separately. For event determinism, we first establish a slightly stronger lemma for processes only (Lemma 14), before covering extended processes. The two diamond properties are more involved. For both, we make use of a function (Def. 16) that, for each location prefix, “localises” in a process the corresponding “component”. This is used to pick out components of a process that are active or inactive in a transition. The function also calculates the variables that are to be extruded along the path to the component to be picked out. The decomposition lemmas (Lemmas 18 and 20) use the above function, to pull apart the derivation tree of transitions to get to the exact part of the tree that concerns the component(s) independent from the component(s) of another transition. In both diamond properties, we have two independent transitions, and hence by decomposition, we can identify the common parts of the derivation of both transitions, and the parts of the derivation where the transitions differ. We then appeal to composition lemmas (Lemmas 22 and 23), that construct transitions where we swap the beginning and end of derivations, thereby completing the missing face of each diamond, modulo some permutations of name binders (enabled by equivariance, Sect. 4.3) and substitutions.

We first establish event determinism for processes only, below. The permutation on variables is used to cope with the EXTRUDE rules that possibly applies α -equivalence to rename bound variables.

► **Lemma 14** (determinism for processes). *Assuming ρ is a permutation (bijective operator) on variables such that $\text{dom}(\rho) \# \pi$ we have, if $P \equiv_{\alpha} Q\rho$ and $P \xrightarrow{\pi}_u B$ and $Q \xrightarrow{\pi'}_u C$ then:*

- *if $\pi = \overline{M}(\lambda)$, then $\pi' = \overline{K}(\lambda')$ and $M =_E K$ and, furthermore, given that we have $C = \nu \vec{y}. (\{N/\lambda'\} \mid Q')$ we have also that $B \equiv_{\alpha} (\nu \vec{y}. (\{N/\lambda\} \mid Q'))\rho$;¹*
- *if $\pi = \pi'$, we have $B \equiv_{\alpha} C\rho$.*

Then we extend the above lemma to extended processes, from which event determinism follows by taking the permutation to be the identity.

► **Lemma 15** (determinism for extend processes). *Assuming ρ is a permutation on variables such that $\text{dom}(\rho) \# \pi$, we have, if $A_0 \equiv A_1\rho$ and $A_0 \xrightarrow{\pi}_u B_0$ and $A_1 \xrightarrow{\pi}_u B_1$ then $B_0 \equiv B_1\rho$.*

The proofs of the diamond properties rely on the following function selecting the component of a process corresponding to a location prefix.

► **Definition 16** (components). *Writing **Proc** the set of processes and **Vars** the set of variables, we define inductively a partial function $\text{Comp} : \{0, 1\}^* \rightarrow (\mathbf{Vars}^* \times \mathbf{Proc}) \rightarrow (\mathbf{Vars}^* \times \mathbf{Proc})$ such that $\text{Comp}(\epsilon)(\vec{y}, P) = (\vec{y}, P)$ and if $\mathbf{s} \neq \epsilon$ then it is defined as follows:*

$$\begin{aligned} \text{Comp}(0\mathbf{s})(\vec{y}, P_0 \mid P_1) &= \text{Comp}(\mathbf{s})(\vec{y}, P_0) & \text{Comp}(\mathbf{s})(\vec{y}, \nu x.P) &= \text{Comp}(\mathbf{s})(\vec{y}x, P) \\ \text{Comp}(1\mathbf{s})(\vec{y}, P_0 \mid P_1) &= \text{Comp}(\mathbf{s})(\vec{y}, P_1) & \text{Comp}(\mathbf{s})(\vec{y}, !P) &= \text{Comp}(\mathbf{s})(\vec{y}, P \mid !P) \end{aligned}$$

¹ This means outputs may only differ in the choice of alias (λ v.s. λ') or in that an equivalent recipe (M v.s. K) may be used, and, furthermore, we are free to rename the alias. This clause is a trick used to determine whether the CLOSE-L or CLOSE-R rule applied in an interaction, by looking at the output action and without being required to record additional information in the interaction event.

The decomposition lemmas below select the part of a derivation tree that concerns a single component. We first establish decomposition for simple prefixes consisting of 0 or 1 . In what follows, we write $\neg : \{0, 1\} \rightarrow \{0, 1\}$ for Boolean negation.

- **Lemma 17** (decomposing prefixed transitions). *For all $c \in \{0, 1\}$, if $P \xrightarrow[cu]{\pi} \nu \vec{z}.(\theta \mid Q)$, then:*
- $Comp(c)(\epsilon, P) = (\vec{y}, P')$ and $\vec{z} = \vec{y}, \vec{x}$ and $P' \xrightarrow[u]{\pi} \nu \vec{x}.(\theta \mid Q')$ and $Comp(c)(\epsilon, Q) = (\epsilon, Q')$.
 - In addition, $Comp(\neg c)(\epsilon, P) = (\vec{y}, R)$ and $Comp(\neg c)(\epsilon, Q) = (\epsilon, R)$ and $\vec{x} \# R$.²

We then appeal to the fact that $Comp(s)(Comp(s')(h, P)) = Comp(s's)(h, P)$ and make use of Lemma 17 repeatedly to generalise decomposition to an arbitrary string.

- **Lemma 18** (decomposing process transitions). *For all $s \in \{0, 1\}^*$ and $P \xrightarrow[su]{\pi} \nu \vec{z}.(\theta \mid Q)$ then we have the following:*
- $Comp(s)(\epsilon, P) = (\vec{y}, P')$ and $\vec{z} = \vec{y}, \vec{x}$ and $P' \xrightarrow[u]{\pi} \nu \vec{x}.(\theta \mid Q')$ and $Comp(s)(\epsilon, Q) = (\epsilon, Q')$;
 - for any $s', s'' \in \{0, 1\}^*$, and $c \in \{0, 1\}$, if $s = s'cs''$ then $Comp(s'-c)(\epsilon, P) = (\vec{w}, R)$ and $Comp(s'-c)(\epsilon, Q) = (\epsilon, R)$ and $Comp(s'c)(\epsilon, P) = (\vec{w}, S)$ and $Comp(s'')(\epsilon, S) = (\vec{v}, P')$, and also $\vec{v}, \vec{x} \# R$.

A richer decomposition lemma is needed for τ -transitions, so that we can identify the two components – for the input and output transition involved in the communication – that are both structurally independent of another transition.

- **Lemma 19** (interaction decomposition). *If $P \xrightarrow[(0\ell_0, 1\ell_1)]{\tau} \nu \vec{z}.(\theta \mid Q)$ then we have $Comp(0)(\epsilon, P) = (\vec{y}, P_0)$ and $Comp(1)(\epsilon, P) = (\vec{y}, P_1)$ and $Comp(0)(\epsilon, Q) = (\epsilon, Q_0)$ and $Comp(1)(\epsilon, Q) = (\epsilon, Q_1)$ and $\vec{z} = \vec{y}, \vec{x}_0, \vec{x}_1$ and $\vec{x}_0 \# Q_1$ and $\vec{x}_1 \# Q_0$ and $\vec{x}_0 \# \vec{x}_1$ and one of the following hold:*

- $P_0 \xrightarrow[\ell_0]{\overline{M}(\lambda)} \nu \vec{x}_0.(\{N/\lambda\} \mid Q_0)$ and $P_1 \xrightarrow[\ell_1]{MN} \nu \vec{x}_1.(\text{id} \mid Q_1)$.
- $P_0 \xrightarrow[\ell_0]{MN} \nu \vec{x}_0.(\text{id} \mid Q_0)$ and $P_1 \xrightarrow[\ell_1]{\overline{M}(\lambda)} \nu \vec{x}_1.(\{N/\lambda\} \mid Q_1)$.

Again appealing to that fact that $Comp(s)(Comp(s')(h, P)) = Comp(s's)(h, P)$ we can generalise decomposition of interactions to an arbitrary prefix string.

- **Lemma 20** (full decomposition of interactions). *For all $s, s_0, s_1 \in \{0, 1\}^*$ such that we have $P \xrightarrow[s(0s_0\ell_0, 1s_1\ell_1)]{\tau} \nu \vec{z}.(\theta \mid Q)$, the following hold:*
- we have $Comp(s0s_0)(\epsilon, P) = (\vec{y}\vec{z}_0, P_0)$ and $Comp(s1s_1)(\epsilon, P) = (\vec{y}\vec{z}_1, P_1)$ and, also we have $Comp(s0s_0)(\epsilon, Q) = (\epsilon, Q_0)$ and $Comp(s1s_1)(\epsilon, Q) = (\epsilon, Q_1)$ and $\vec{z} = \vec{y}, \vec{z}_0, \vec{x}_0, \vec{z}_1, \vec{x}_1$ and $\vec{z}_0, \vec{x}_0 \# Q_1$ and $\vec{z}_1, \vec{x}_1 \# Q_0$ and $\vec{z}_0, \vec{x}_0 \# \vec{z}_1, \vec{x}_1$ and one of the following hold:
 - $P_0 \xrightarrow[\ell_0]{\overline{M}(\lambda)} \nu \vec{x}_0.(\{N/\lambda\} \mid Q_0)$ and $P_1 \xrightarrow[\ell_1]{MN} \nu \vec{x}_1.(\text{id} \mid Q_1)$,
 - $P_0 \xrightarrow[\ell_0]{MN} \nu \vec{x}_0.(\text{id} \mid Q_0)$ and $P_1 \xrightarrow[\ell_1]{\overline{M}(\lambda)} \nu \vec{x}_1.(\{N/\lambda\} \mid Q_1)$;
 - for any $s', s'' \in \{0, 1\}^*$, and $c \in \{0, 1\}$ and $s' \neq s$, we have the following: if $s0s_0 = s'cs''$ or $s1s_1 = s'cs''$ then $Comp(s'-c)(\epsilon, P) = (\vec{w}, R)$ and $Comp(s'-c)(\epsilon, Q) = (\epsilon, R)$ and also $Comp(s'c)(\epsilon, P) = (\vec{w}, S)$ and $Comp(s'')(\epsilon, S) = (\vec{v}, P')$, and we have $\vec{v}, \vec{x}_0, \vec{x}_1 \# R$.

² This states that the locations independent from P' are unchanged by the transition stemming from P' , except that the common name binders will be extruded. Generalisations of this statement carry through to the other decomposition lemmas.

Composition lemmas are required to remember the part of the derivation tree picked out by the decomposition lemmas, when constructing a new transition. As for decomposition, we first establish composition for a single-character prefix: **0** or **1**.

► **Lemma 21** (composing in one-step). *For any $s \in \{0, 1\}$, if $\text{Comp}(s)(P) = (\vec{y}, P')$ and we have $P' \xrightarrow[\pi_u]{\pi} \nu \vec{z}.(\sigma \mid Q')$ and, furthermore, $\text{Comp}(\neg s)(P) = (\vec{y}, R)$ and $\vec{z} \# R$, then, for some Q , we have $P \xrightarrow[\pi_{su}]{\pi} \nu \vec{y}, \vec{z}.(\sigma \mid Q)$ and $\text{Comp}(s)(\epsilon, Q) = (\epsilon, Q')$.*

As for decomposition, we extend composition to any prefix.

► **Lemma 22** (composing transitions). *Assume $s \in \{0, 1\}^*$, and $\text{Comp}(s)(P) = (\vec{y}, P')$ and we have $P' \xrightarrow[\pi_u]{\pi} \nu \vec{z}.(\sigma \mid Q')$, and, furthermore, for any $s', s'' \in \{0, 1\}^*$, and $c \in \{0, 1\}$ such that $s = s'cs''$, we have $\text{Comp}(s'-c)(\epsilon, P) = (\vec{w}, R)$ and $\text{Comp}(s'c)(\epsilon, P) = (\vec{w}, S)$ and $\text{Comp}(s'')(c, S) = (\vec{v}, P')$, and also $\vec{v}, \vec{x} \# R$. Given these assumptions, we have that, for some Q , we have $P \xrightarrow[\pi_{su}]{\pi} \nu \vec{y}, \vec{z}.(\sigma \mid Q)$ and $\text{Comp}(s)(\epsilon, Q) = (\epsilon, Q')$.*

Interactions can also be composed.

► **Lemma 23** (composing interactions). *Assume $s, s_0, s_1 \in \{0, 1\}$, are such that we have $\text{Comp}(s0)(\epsilon, P) = (\vec{y}, Q_0)$ and $\text{Comp}(s1)(\epsilon, P) = (\vec{y}, Q_1)$ and $\text{Comp}(s_0)(\epsilon, Q_0) = (\vec{x}_0, P_0)$ and $\text{Comp}(s_1)(\epsilon, Q_1) = (\vec{x}_1, P_1)$ and either of the following hold:*

- $P_0 \xrightarrow[\ell_0]{\overline{M}(\lambda)} \nu \vec{z}.(\{N/\lambda\} \mid P'_0)$ and $P_1 \xrightarrow[\ell_1]{MN} \nu \vec{w}.(\text{id} \mid P'_1)$,
- $P_0 \xrightarrow[\ell_0]{MN} \nu \vec{z}.(\text{id} \mid P'_0)$ and $P_1 \xrightarrow[\ell_1]{\overline{M}(\lambda)} \nu \vec{w}.(\{N/\lambda\} \mid P'_1)$;

and also assume we have, for any $s', s'' \in \{0, 1\}^*$, and $c \in \{0, 1\}$, such that $s' \neq s$ and either $s0s_0 = s'cs''$ or $s1s_1 = s'cs''$, we have that $\text{Comp}(s'-c)(\epsilon, P) = (\vec{w}, R)$ and also $\text{Comp}(s'c)(\epsilon, P) = (\vec{w}, S)$ and $\text{Comp}(s'')(c, S) = (\vec{v}, P')$, and $\vec{v}, \vec{z}, \vec{w} \# R$. Under those assumptions, for some P' , we have the transition $P \xrightarrow[\pi_{su}]{\pi} \nu \vec{y}, \vec{x}_0, \vec{z}, \vec{x}_1, \vec{w}.(\sigma \mid P')$ and also we have $\text{Comp}(s0s_0)(\epsilon, P) = (\epsilon, P'_0)$ and $\text{Comp}(s1s_1)(\epsilon, P) = (\epsilon, P'_1)$.

Using the decomposition and composition lemmas, we can now construct the transitions required to complete the two diamond properties.

► **Lemma 24** (diamond property 1). *If $(\pi_0, u_0) I (\pi_1, u_1)$, $A \xrightarrow[\pi_0]{\pi_0} B_0$ and $A \xrightarrow[\pi_1]{\pi_1} B_1$, then $\exists C_0, C_1$ s.t. $B_0 \xrightarrow[\pi_1]{\pi_1} C_0$ and $B_1 \xrightarrow[\pi_0]{\pi_0} C_1$ and $C_0 \equiv C_1$.*

In both diamond properties, there are several cases depending on what combination of τ and output labelled events we are considering to be independent. Below we present the top-level breakdown of the case analysis, which applies to both diamond properties. We also provide the details of one of the most interesting cases, where two independent output transitions occur.

► **Lemma 25** (diamond property 2). *If $(\pi_0, u_0) I (\pi_1, u_1)$, $A \xrightarrow[\pi_0]{\pi_0} B_0$ and $B_0 \xrightarrow[\pi_1]{\pi_1} C_0$, then $\exists B_1, C_1$ s.t. $A \xrightarrow[\pi_1]{\pi_1} B_1$ and $B_1 \xrightarrow[\pi_0]{\pi_0} C_1$ and $C_0 \equiv C_1$.*

Proof. Assume we have $(\pi_0, u_0) I (\pi_1, u_1)$, and the two transitions $A \xrightarrow[\pi_0]{\pi_0} B_0$ and $B_0 \xrightarrow[\pi_1]{\pi_1} C_0$.

For two transitions from the same state we have $(\pi_0, u_0) I (\pi_1, u_1)$ iff for all $\ell_0 \in \text{Loc}(u_0)$ and for all $\ell_1 \in \text{Loc}(u_1)$, we have $\ell_0 I_\ell \ell_1$ (i.e., there is no structural causality), and, furthermore, if $\pi_0 = \overline{M}_0(\alpha_0)$, then $\alpha_0 \# \pi_1$ (i.e., there is no link causality). The structural independence ensures that, without loss of generality, we have one of the following.

- Both location labels have a largest common prefix $s \in \{0, 1\}^*$ and hence are of the form $s0u'_0 = u_0$ and $s1u'_1 = u_1$.
- At least one transition is labelled with a τ action (without loss of generality let $\pi_0 = \tau$), and so $u_0 = s0(\ell_0, 1\ell_1)$ and there is a string $s_1 \in \{0, 1\}^*$, characters $c, d \in \{0, 1\}$ and prefix location ℓ' such that $s_0ds_1cu'_1 = u_1$ and $s_1\text{-}c\ell' = \ell_d$. That is, one transition is an interaction, and the other transition is entirely located within one of the locations from which either interacting input or the interacting output emanated.
- Both are τ transitions with a common prefix $s \in \{0, 1\}^*$ such that $u_0 = s(0\ell_0^0, 1\ell_0^1)$ and $u_1 = s(0\ell_1^0, 1\ell_1^1)$ and there are strings $s_1 \in \{0, 1\}^*$ such that $\ell_0^0 = s_00\ell_0^0$ and $\ell_1^0 = s_01\ell_1^0$ and $\ell_0^1 = s_10\ell_0^1$ and $\ell_1^1 = s_11\ell_1^1$. That is, we have two interactions, where the interaction occurs in the same location (even though the inputs and outputs involved are independent).

We break down further the first case above, where the two independent transitions have a common prefix. Due to differences between ALIAS-OUT or and ALIAS-FREE, we should consider separately the cases where the first transition is an output transition or a free transition. We consider only the most interesting case, where we appeal to the absence of link causality, which is only relevant when $\pi_0 = \overline{M_0}(\alpha_0)$ is an output transition. That case itself breaks down into two cases, where π_1 is either another output transition or a free transition. We provide only the case when π_1 is an output transition such that $\pi_1 = \overline{M_1}(\alpha_1)$ below.

Without loss of generality (0 and 1 can be reversed without changing the proof), assume there exists s such that $u_0 = \ell_0 = s0s_0[t_0]$ and $u_1 = \ell_1 = s1s_1[t_1]$.

Thus, by the RES rule, repeatedly, we have $A = \nu\vec{x}.(\sigma | P)$ and $\alpha_0 = s0s_0\lambda_0$ and $\vec{x} \# M_0$ and $B_0 = \nu\vec{x}, \vec{y}_0.(\sigma \circ \{N_0/s_0s_0\lambda_0\} | Q_0)$, and, also by the ALIAS-OUT rule we have the following.

$$\frac{P \xrightarrow[\text{s0s}_0[t_0]]{\overline{M_0}\sigma(\lambda_0)} \nu\vec{y}_0.(\{N_0/\lambda_0\} | Q_0) \quad \vec{y} \# \text{ran}(\sigma) \quad \text{fa}(M_0) \subseteq \text{dom}(\sigma) \quad \text{s0s}_0\lambda_0 \# \text{dom}(\sigma)}{\frac{\sigma | P \xrightarrow[\text{s0s}_0[t_0]]{\overline{M_0}(s_0s_0\lambda_0)} \nu\vec{y}_0.(\sigma \circ \{N_0/s_0s_0\lambda_0\} | Q_0)}{\nu\vec{x}.(\sigma | P) \xrightarrow[\text{s0s}_0[t_0]]{\overline{M_0}(s_0s_0\lambda_0)} \nu\vec{x}, \vec{y}_0.(\sigma \circ \{N_0/s_0s_0\lambda_0\} | Q_0)}}$$

Now, since we have $P \xrightarrow[\text{s0s}_0[t_0]]{\overline{M_0}\sigma(\lambda_0)} \nu\vec{y}_0.(\{N_0/\lambda_0\} | Q_0)$, by Lemma 18, we have the following:

- $P_0 \xrightarrow[\text{s}_0[t_0]]{\overline{M_0}\sigma(\lambda_0)} \nu\vec{z}_0.(\{N_0/\lambda_0\} | Q'_0)$ and $\text{Comp}(s_0)(\epsilon, P) = (\vec{y}, P_0)$ and $\vec{y}_0 = \vec{y}, \vec{z}_0$ and $\text{Comp}(s_0)(\epsilon, Q_0) = (\vec{y}, Q'_0)$.
- for any $s', s'' \in \{0, 1\}^*$, and $c \in \{0, 1\}$, if $s_0 = s'cs''$ then $\text{Comp}(s'\text{-}c)(\epsilon, P) = (\vec{w}, R)$ and $\text{Comp}(s'\text{-}c)(\epsilon, Q_0) = (\epsilon, R)$ and $\text{Comp}(s'c)(\epsilon, P) = (\vec{w}, S)$ and $\text{Comp}(s'')(\epsilon, S) = (\vec{v}, P_0)$, and also $\vec{v}, \vec{x} \# R$.

Now since $B_0 = \nu\vec{x}, \vec{y}_0.(\sigma \circ \{N_0/s_0s_0\lambda_0\} | Q_0)$ and $B_0 \xrightarrow[\ell_1]{\pi_1} C$, by the RES rule repeatedly and the ALIAS-OUT rule, we have the following, where $\vec{x}, \vec{y}_0 \# M_1$ and $\theta_0 = \sigma \circ \{N_0/s_0s_0\lambda_0\}$ and $C_0 = \nu\vec{x}, \vec{y}_0, \vec{z}_1.(\theta_0 \circ \{N_1/s_1s_1\lambda_1\} | R_0)$.

$$\frac{Q_0 \xrightarrow[\text{s1s}_1[t_1]]{\overline{M_1}\sigma(\lambda_1)} \nu\vec{z}_1.(\{N_1/\lambda_1\} | R_0) \quad \vec{z}_1 \# \text{ran}(\theta_0) \quad \text{fa}(M_1) \subseteq \text{dom}(\theta_0) \quad \text{s1s}_1\lambda_1 \# \text{dom}(\theta_0)}{\frac{\theta_0 | Q_0 \xrightarrow[\text{s1s}_1[t_1]]{\overline{M_1}(s_1s_1\lambda_1)} \nu\vec{z}_1.(\theta_0 \circ \{N_1/s_1s_1\lambda_1\} | R_0)}{\nu\vec{x}, \vec{y}_0.(\theta_0 | Q_0) \xrightarrow[\text{s1s}_1[t_1]]{\overline{M_1}(s_1s_1\lambda_1)} \nu\vec{x}, \vec{y}_0, \vec{z}_1.(\theta_0 \circ \{N_1/s_1s_1\lambda_1\} | R_0)}}$$

Since $Q_0 \xrightarrow[\mathbf{s1s1}[t_1]]{\overline{M_1\sigma}(\lambda_1)} \nu \bar{z}_1.(\{M_1/\lambda_1\} \mid R_0)$, by Lemma 18, we have the following:

- $Comp(\mathbf{s1})(\epsilon, Q_0) = (\mathbf{s1}, Q'_0)$ and $Q'_0 \xrightarrow[\mathbf{s1}[t_1]]{\overline{M_1\sigma}(\lambda_1)} \nu \bar{z}_1.(\{M_1/\lambda_1\} \mid R'_0)$ and $Comp(\mathbf{s1})(\epsilon, R_0) = (\epsilon, R'_0)$.
- for any $s', s'' \in \{0, 1\}^*$, and $c \in \{0, 1\}$, if $\mathbf{s1} = s'cs''$ then $Comp(s'\neg c)(\epsilon, Q_0) = (\bar{w}, R)$ and $Comp(s'\neg c)(\epsilon, R_0) = (\epsilon, R)$ and $Comp(s'c)(\epsilon, Q_0) = (\bar{w}, S)$ and $Comp(s'')(\epsilon, S) = (\bar{v}, Q'_0)$, and also $\bar{v}, \bar{x} \# R$.

From the above we have that $Comp(\mathbf{s1})(\epsilon, P) = (\bar{y}, Q'_0)$ and $Comp(\mathbf{s1})(\epsilon, Q_0) = (\epsilon, Q'_0)$. We also have that, for any $s', s'' \in \{0, 1\}^*$, and $c \in \{0, 1\}$, if $s = s'cs''$ then $Comp(s'\neg c)(\epsilon, P_0) = (\bar{w}, R)$ and $Comp(s'\neg c)(\epsilon, Q_0) = (\epsilon, R)$ and $Comp(s'\neg c)(\epsilon, R_0) = (\epsilon, R)$ and $Comp(s'c)(\epsilon, Q_0) = (\bar{w}, S)$ and $Comp(s'')(\epsilon, S) = (\bar{v}, Q'_0)$, and also $\bar{v}, \bar{x} \# R$.

We now appeal to the absence of *link causality*, so we know that $\mathbf{s0s0}\lambda_0 \# M_1$, and hence $M_1\theta_0 = M_1\sigma$, and so $Q'_0 \xrightarrow[\mathbf{s1}[t_1]]{\overline{M_1\sigma}(\lambda_1)} \nu \bar{z}_1.(\{N_1/\lambda_1\} \mid R'_0)$. Therefore by Lemma 22, we have $P \xrightarrow[\mathbf{s1s1}[t_1]]{\overline{M_1\sigma}(\lambda_1)} \nu \bar{y}, \bar{z}_1.(\{N_1/\lambda_1\} \mid Q_1)$ and $Comp(\mathbf{s1})(\epsilon, Q_1) = (\epsilon, R'_0)$. Since we know $\text{fa}(M_1) \subseteq \text{dom}(\theta_0)$ and $\mathbf{s0s0}\lambda_0 \# M_1$ we know that $\text{fa}(M_1) \subseteq \text{dom}(\sigma)$. Since $\mathbf{s1s1}\lambda_1 \# \text{dom}(\theta_0)$ we have $\mathbf{s1s1}\lambda_1 \# \text{dom}(\sigma)$. Thus, by ALIAS-OUT and RES repeatedly, we have the following.

$$\frac{P \xrightarrow[\mathbf{s1s1}[t_1]]{\overline{M_1\sigma}(\lambda_1)} \nu \bar{y}, \bar{z}_1.(\{N_1/\lambda_1\} \mid Q_1) \quad \text{fa}(M_1) \subseteq \text{dom}(\sigma) \quad \bar{y}, \bar{z}_1 \# \text{ran}(\sigma) \quad \mathbf{s1s1}\lambda_1 \# \text{dom}(\sigma)}{\sigma \mid P \xrightarrow[\mathbf{s1s1}[t_1]]{\overline{M_1\sigma}(\mathbf{s1s1}\lambda_1)} \nu \bar{y}, \bar{z}_1.(\sigma \circ \{N_1/\mathbf{s1s1}\lambda_1\} \mid Q_1)}$$

$$\frac{\sigma \mid P \xrightarrow[\mathbf{s1s1}[t_1]]{\overline{M_1\sigma}(\mathbf{s1s1}\lambda_1)} \nu \bar{y}, \bar{z}_1.(\sigma \circ \{N_1/\mathbf{s1s1}\lambda_1\} \mid Q_1)}{\nu \bar{x}.(\sigma \mid P) \xrightarrow[\mathbf{s1s1}[t_1]]{\overline{M_1\sigma}(\mathbf{s1s1}\lambda_1)} \nu \bar{x}, \bar{y}, \bar{z}_1.(\sigma \circ \{N_1/\mathbf{s1s1}\lambda_1\} \mid Q_1)}$$

Recall that $A = \nu \bar{x}.(\sigma \mid P)$, hence we have the first of our desired transitions.

It remains to show that $\nu \bar{x}, \bar{y}, \bar{z}_1.(\theta_1 \mid Q_1) \xrightarrow[u_0]{\pi_0} C_1$, where $\theta_1 = \sigma \circ \{N_1/\mathbf{s1s1}\lambda_1\}$, and also $C_0 \equiv C_1$. Since $\text{fa}(M_0) \subseteq \text{dom}(\sigma) \subseteq \text{dom}(\theta_1)$, we have $M_0\sigma = M_0\theta_1$, thus $P_0 \xrightarrow[\mathbf{s0}[t_0]]{\overline{M_0\theta_1}(\lambda_0)} \nu \bar{z}_0.(\{N_0/\lambda_0\} \mid Q'_1)$. Therefore, since we know (via Lemma 18) that $Comp(\mathbf{s0})(\epsilon, Q_1) = (\epsilon, P_0)$, by Lemma 22, $Q_1 \xrightarrow[\mathbf{s0}[t_0]]{\overline{M_0\theta_1}(\lambda_0)} \nu \bar{z}_0.(\{N_0/\lambda_0\} \mid R_1)$ and $Comp(\mathbf{s0})(\epsilon, R_1) = (\epsilon, Q'_0)$.

By the RES rule repeatedly and the ALIAS-OUT rule, we have that $\bar{x}, \bar{y}, \bar{z}_1 \# M_0$ and we have the following transition, and so $C_1 = \nu \bar{x}, \bar{y}, \bar{z}_1, \bar{z}_0.(\theta_1 \circ \{N_0/\mathbf{s0s0}\lambda_0\} \mid R_1)$.

$$\frac{Q_1 \xrightarrow[\mathbf{s0s0}[t_0]]{\overline{M_0}(\lambda_0)} \nu \bar{z}_0.(\{N_0/\mathbf{s0s0}\lambda_0\} \mid R_1) \quad \text{fa}(M_0) \subseteq \text{dom}(\theta_1) \quad \bar{z}_0 \# \text{ran}(\theta_1) \quad \mathbf{s0s0}\lambda_0 \# \text{dom}(\theta_1)}{\theta_1 \mid Q_1 \xrightarrow[\mathbf{s0s0}[t_0]]{\overline{M_0}(\mathbf{s0s0}\lambda_0)} \nu \bar{z}_0.(\sigma \circ \{N_1/\mathbf{s1s1}\lambda_1\} \circ \{N_0/\mathbf{s0s0}\lambda_0\} \mid R_1)}$$

$$\frac{\theta_1 \mid Q_1 \xrightarrow[\mathbf{s0s0}[t_0]]{\overline{M_0}(\mathbf{s0s0}\lambda_0)} \nu \bar{z}_0.(\sigma \circ \{N_1/\mathbf{s1s1}\lambda_1\} \circ \{N_0/\mathbf{s0s0}\lambda_0\} \mid R_1)}{\nu \bar{x}, \bar{y}, \bar{z}_1.(\theta_1 \mid Q_1) \xrightarrow[\mathbf{s0s0}[t_0]]{\overline{M_0}(\mathbf{s0s0}\lambda_0)} \nu \bar{x}, \bar{y}, \bar{z}_1, \bar{z}_0.(\sigma \circ \{N_1/\mathbf{s1s1}\lambda_1\} \circ \{N_0/\mathbf{s0s0}\lambda_0\} \mid R_1)}$$

Now since for all $s', s'' \in \{0, 1\}$ and $c, d \in \{0, 1\}$ such that $s'cs'' = sd$ we have $Comp(s'\neg c)(\epsilon, R_0) = Comp(s'\neg c)(\epsilon, R_1)$ (via Lemma 18 and the above), clearly it is the case that $R_0 = R_1$. Furthermore, we have the following, as required.

$$C_0 = \nu \bar{x}, \bar{y}, \bar{z}_0, \bar{z}_1.(\sigma \circ \{N_0/\mathbf{s0s0}\lambda_0\} \circ \{N_1/\mathbf{s1s1}\lambda_1\} \mid R_0)$$

$$\equiv \nu \bar{x}, \bar{y}, \bar{z}_1, \bar{z}_0.(\sigma \circ \{N_1/\mathbf{s1s1}\lambda_1\} \circ \{N_0/\mathbf{s0s0}\lambda_0\} \mid R_1) = C_1$$

Other cases follow a similar pattern of applying to decomposition and composition. ◀

Weak Progressive Forward Simulation Is Necessary and Sufficient for Strong Observational Refinement

Brijesh Dongol  

University of Surrey, UK

Gerhard Schellhorn 

Universität Augsburg, Germany

Heike Wehrheim  

Universität Oldenburg, Germany

Abstract

Hyperproperties are correctness conditions for labelled transition systems that are more expressive than traditional trace properties, with particular relevance to security. Recently, Attiya and Enea studied a notion of strong observational refinement that preserves all hyperproperties. They analyse the correspondence between forward simulation and strong observational refinement in a setting with only finite traces. We study this correspondence in a setting with both finite and infinite traces. In particular, we show that forward simulation does not preserve hyperliveness properties in this setting. We extend the forward simulation proof obligation with a (weak) progress condition, and prove that this *weak progressive forward simulation* is equivalent to strong observational refinement.

2012 ACM Subject Classification Theory of computation → Semantics and reasoning; Theory of computation → Concurrency; Security and privacy → Formal methods and theory of security

Keywords and phrases Strong Observational Refinement, Hyperproperties, Forward Simulation, Weak Progressiveness

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.31

Funding *Brijesh Dongol*: EPSRC grants EP/V038915/1 and EP/R032556/1, VeTSS and ARC Discovery Grant DP190102142.

Heike Wehrheim: DFG Grant WE2290/12-1.

Acknowledgements We thank John Derrick, Simon Doherty, Constantin Enea and our anonymous CONCUR reviewers for their comments on earlier versions of this paper.

1 Introduction

Linearizability [19] has become a standard safety condition for concurrent objects that access shared state. Golab, Higham and Woelfel [13] however showed that linearizability does *not* preserve probability distributions in randomised algorithms. They therefore proposed a notion called *strong linearizability*, which unlike linearizability, must use the same linearization order for *every* prefix of a linearizable history. Strong linearizability allows consideration of concurrent objects in the presence of *adversaries* and can – amongst others – be used to show the preservation of security properties. Here, the adversary is modelled by an *adversarial scheduler*, which plays the role of a *strong adversary* [1].

Our security properties of interest are *hyperproperties* [5], which are properties over *sets of sets of traces* (analogous to trace properties, which are over sets of traces). Hyperproperties allow characterisation, for instance, of information flow properties such as non-interference and observational determinism. Like trace properties, which can be characterised by a conjunction of a safety and a liveness property, every hyperproperty can be characterised as the conjunction of a hypersafety and hyperliveness property. For instance, as observed



© Brijesh Dongol, Gerhard Schellhorn, and Heike Wehrheim;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 31; pp. 31:1–31:23

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

by Clarkson and Schneider [5], *observational determinism* [33] is a hypersafety property, *possibilistic information flow* [23] is a hyperliveness property, and *Goguen and Meseguer's noninterference* property [12] is a conjunction of a hypersafety and hyperliveness property.

Attiya and Enea [2] revisited preservation of hyperproperties in the context of concurrent objects and proposed a generalisation of strong linearizability called *strong observational refinement*. They showed that strong observational refinement preserves *all* hyperproperties, when replacing an abstract library specification, A , by a concrete library implementation, C , in a client program, P . Here, C strongly observationally refines A iff the executions of *any* client program P using C as scheduled by *some* scheduler cannot be observationally distinguished from those of P using A under another scheduler¹.

A second claim in [2] is that forward simulation [22] is *equivalent* to strong observational refinement, i.e., it is both necessary and sufficient. The claim is motivated with examples using (hyper)safety properties, however it raises questions for (hyper)liveness. It turns out, that the study of strong observational refinement and forward simulation by Attiya and Enea is in the restricted setting of finite traces², though this restriction is unclear in their paper [2]. Thus, all hyperproperties considered by Attiya and Enea are hypersafety properties, which leaves out a large class of hyperproperties. We described the problem, namely that forward simulation does *not* preserve hyperliveness properties in our recent brief announcement [8]. There, we also proposed a new condition called *progressive forward simulation* that strengthens forward simulation so that it preserves all hyperproperties through refinement (i.e., progressive forward simulation is a *sufficient* condition).

Our point of departure for this paper is the question in the other direction: “Is progressive forward simulation necessary for strong observational refinement?” The answer, it turns out, is no! As we shall see in §4.1, it is possible for a concrete object to be a strong observational refinement of some abstract object, yet for there to be no progressive forward simulation between them.

Contributions

In this paper, we present a relaxation of progressive forward simulation that is both necessary and sufficient. Our main contribution therefore is a new result that closes the gap between strong observational refinement and a corresponding proof technique between concurrent objects. In particular, we provide, for the first time, a stepwise technique that coincides with a notion of refinement that preserves all client-object hyperproperties.

Overview

In §2 we present our main example to demonstrate the inadequacy of forward simulation for hyperliveness properties. §3 presents the formal background and recaps the key definitions and prior results. §4 motivates and defines weak progressive forward simulation, which we prove to be both sufficient (§5) and necessary (§6) for strong observational refinement.

2 Motivating Example

We start by giving an example of an abstract atomic object A and a non-atomic implementation C such that there *is* a forward simulation from C to A , but hyperliveness properties are not preserved for all schedules.

¹ Both of these schedulers are additionally required to be *admissible* and *deterministic* (see §3.2).

² Private communication


```

int* current_val initially 0

int fetch_and_inc():
F1. do
F2.   n = LL(&current_val)
F3.   while (!SC(&current_val, n + 1))
F4.   return n

```

■ **Figure 1** A fetch-and-inc implementation with a nonterminating schedule when LL and SC are implemented using the algorithm of [20].

As the atomic abstract object A we choose a *fetch-and-inc* object with just one operation, `fetch_and_inc()`, which increments the value of a shared integer variable and returns the value of that variable before the increment. Let P be a program with two threads t_1 and t_2 , each of which executes one `fetch_and_inc` operation and assigns the return value to a local variable of the thread. Clearly, for any scheduler S , the variable assignment of both threads will eventually occur. This “eventually” property can be expressed as a hyperproperty.

Now, consider the *fetch-and-inc* implementation presented in Figure 1. This implementation uses the *load-linked/store-conditional* (LL/SC) instruction pair. The `LL(ptr)` operation loads the value at the location pointed to by the pointer `ptr`. The `SC(ptr,v)` conditionally stores the value v at the location pointed to by `ptr` if the location has not been modified by another `SC` since the executing thread’s most recent `LL(ptr)` operation. If the update actually occurs, `SC` returns `true`, otherwise the location is not modified and `SC` returns `false`. In the first case, we say that the `SC` *succeeds*. Otherwise, we say that it *fails*.

Critically, we stipulate that the LL and SC operations are implemented using the algorithm of [20]. This algorithm has the following property. If thread t_1 executes an LL operation, and then thread t_2 executes an LL operation *before* t_1 has executed its subsequent SC operation, then that SC is guaranteed to fail. This happens even though there is no intervening modification of the location.

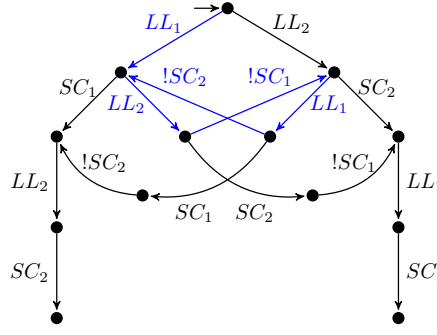
Now, let C be a *labelled transition systems* (LTS) representing a multithreaded version of this `fetch_and_inc` implementation, using the specified LL/SC algorithm³. Figure 2 gives a sketch of this LTS, detailing just the most important actions. Consider furthermore the program P (above) running against the object O_1 . A scheduler can continually alternate the LL at line F2 of t_1 and that of t_2 (with some executions of F3 in between), such that neither `fetch_and_inc` operation ever completes (see the blue arrows in the LTS). Therefore, unlike when using the A object, the variable assignments of P will never occur, so the C system does not satisfy the hyperproperty for all schedulers.

There is, however, a forward simulation (see Definition 3.2) from C to A . Therefore, standard forward simulation is insufficient to show that all hyperproperties are preserved.

3 Background

Notation. Let ξ and ξ' be sequences. The *empty sequence* is denoted ε and the *length* of ξ denoted $\#\xi$. We write $\xi \sqsubseteq \xi'$ (similarly, $\xi \sqsubset \xi'$) iff ξ is a *prefix* (similarly, *proper prefix*) of ξ' . Assuming $m < n \leq \#\xi$, we write $\xi^{<n}$ for the prefix of ξ of length n and $\xi[m]$ for the element of ξ at index m . Thus, $\xi^{<0} = \varepsilon$, and if $n > 0$, $\xi^{<n} = \xi[0] \cdot \xi[1] \cdot \xi[2] \cdots \xi[n-1]$. If ξ

³ There are several ways to represent a multithreaded program or object as an LTS, e.g., [21, 28].



■ **Figure 2** Sketch of labelled transition system for the example in Fig. 1 for threads t_1 and t_2 calling `fetch_and_inc` once (LL_i action for execution of F2 by thread t_i , SC_i for F3 when the SC returns true, $!SC_i$ when it returns false, $i \in \{1, 2\}$).

is finite, we let $last(\xi)$ be the last element of ξ , i.e., $last(\xi) = \xi[\#\xi - 1]$. For a set S , let $\xi|_S$ be the sequence ξ restricted to elements in S . We lift this to sets of sequences and define $T|_S = \{\xi|_S \mid \xi \in T\}$. Let $x^\omega = x \cdot x \cdot x \cdot \dots$ be the infinite sequence comprising the element x .

3.1 LTSs, refinement and forward simulation

We describe (concurrent) systems by *labelled transition systems* (LTSs). An LTS $L = (Q, Q^{ini}, \Sigma, \delta)$ consists of a (possibly infinite) set of *states* Q , an alphabet Σ of *actions*, initial states $Q^{ini} \subseteq Q$ and a transition relation $\delta \subseteq Q \times \Sigma \times Q$. We say that an action a is enabled in state q iff there exists a state q' such that $(q, a, q') \in \delta$. Labelled transition systems give rise to (finite or infinite) *runs* which are alternating sequences $q_0 \cdot a_1 \cdot q_1 \cdot a_2 \cdot \dots$ of states and actions with $(q_i, a_{i+1}, q_{i+1}) \in \delta$. We also write $q_0 \xrightarrow{a_1 \dots a_n} q_n$ if there is a finite run $q_0 \cdot a_1 \cdot q_1 \cdot \dots \cdot a_n \cdot q_n$. In particular, $q \xrightarrow{\varepsilon} q$. A run is an *execution* of an LTS L if $q_0 \in Q^{ini}$.

A *trace* is the sequence of actions of an execution and the set of traces of an LTS L is denoted $T(L)$, which may be partitioned into *finite* traces, denoted Σ^* , and *infinite* traces, denoted Σ^ω . We use $\sigma \in \Sigma^*$ and $\pi \in \Sigma^\omega$ when referring to finite and infinite traces, respectively, and $\rho \in T(L)$ to refer to a trace that may be finite or infinite. Note that for any L , $T(L)$ is prefix closed.

An LTS is *step-deterministic* if it has a single initial state (i.e., $Q^{ini} = \{q^{ini}\}$), and for every state q and action a , if $q \xrightarrow{a} q'$ and $q \xrightarrow{a} q''$ then $q' = q''$. Step-determinism implies that each trace corresponds to a unique run; if the trace is finite then there is at most one state q' such that $q^{ini} \xrightarrow{\sigma} q'$. In this case, we let $state(\sigma)$ denote q' .⁴

Like [2], we use step-deterministic LTSs to describe *objects* and the *programs* that use these objects. The terms “object” and “program” are taken from [2], the object describing a library (e.g. of a data structure) and the program using this library by calling operations of it. To define interfaces between objects and programs, we partition the actions of an LTS into *internal* and *external* actions. Objects offer operations to their environment which can be invoked (by programs) using an external invocation action from a set I with a corresponding external response action from a set R . For this paper, the exact form of invocation and response actions is unimportant. Besides invocations and responses, an object may have further internal actions used to implement the operations.

⁴ Note that a step-deterministic LTS differs from the notion of a deterministic automaton of Lynch and Vaandrager [22]. Attiya and Enea simply refer to step-deterministic LTSs as deterministic LTSs [2].

A program *uses* objects by invoking their operations and waiting for the corresponding responses. Thus, if P is an LTS corresponding to a program its actions can be partitioned as follows: $\Sigma_P = I \dot{\cup} R \dot{\cup} \Gamma_P$, where $\dot{\cup}$ is a disjoint union and Γ_P is the set of program actions. The composition of a program with an object is formally defined as the *product* of two LTSs.

► **Definition 3.1** (Program-object composition). *Suppose that $P = (Q_P, Q_P^{ini}, \Sigma_P, \delta_P)$ and $O = (Q_O, Q_O^{ini}, \Sigma_O, \delta_O)$ are LTSs. The product of P with O , denoted $P \times O$, is the LTS $(Q, Q^{ini}, \Sigma, \delta)$ with*

- $Q = Q_P \times Q_O$, $Q^{ini} = Q_P^{ini} \times Q_O^{ini}$, $\Sigma = \Sigma_P \cup \Sigma_O$, and
- $\delta = \bigcup_{a \in \Sigma_P \cap \Sigma_O} \{(q_P, q_O) \xrightarrow{a} (q'_P, q'_O) \mid q_P \xrightarrow{a} q'_P \wedge q_O \xrightarrow{a} q'_O\} \cup$
 $\bigcup_{a \in \Sigma_P \setminus \Sigma_O} \{(q_P, q_O) \xrightarrow{a} (q'_P, q_O) \mid q_P \xrightarrow{a} q'_P\} \cup$
 $\bigcup_{a \in \Sigma_O \setminus \Sigma_P} \{(q_P, q_O) \xrightarrow{a} (q_P, q'_O) \mid q_O \xrightarrow{a} q'_O\}$

Note that $\Sigma_P \cap \Sigma_O$ in our case will typically be $I \cup R$.

An object can either be an abstract (often sequential) specification (denoted L_A or simply A) or a concrete implementation (denoted L_C or C). A *history* of an LTS is a sequence $\rho|_\Delta$, where ρ is a trace of the LTS and $\Delta \subseteq \Sigma$ is the set of external actions. We formally relate the behaviours of A and C by comparing their histories. We say C is a Δ -refinement of A iff $T(C)|_\Delta \subseteq T(A)|_\Delta$. One can establish Δ -refinement between C and A by proving forward simulation between the systems.⁵

► **Definition 3.2** (Forward simulation). *Let C and A be two LTSs with sets of actions Σ_C and Σ_A , respectively, and let $\Delta \subseteq \Sigma_C \cap \Sigma_A$. A relation $F \subseteq Q_C \times Q_A$ is a Δ -forward simulation from C to A iff both of the following hold:*

Initialisation. $(q_C^{ini}, q_A^{ini}) \in F$,

Simulation step. For all $(q_C, q_A) \in F$, if $q_C \xrightarrow{a} q'_C$ then there exist $\sigma \in \Sigma_A^*$ and $q'_A \in Q_A$ such that $a|_\Delta = \sigma|_\Delta$, $q_A \xrightarrow{\sigma} q'_A$ and $(q'_C, q'_A) \in F$.

Note that σ in the above definition may be ε in which case the condition $a|_\Delta = \sigma|_\Delta$ reduces to $a|_\Delta = \varepsilon$. In this case, the proof obligation for the simulation step forms a *triangular diagram*. For instance, in Figure 4, the step executing τ_3 forms such as diagram.

► **Lemma 3.3** (Lynch [21]). *If there is a Δ -forward simulation from C to A , then $T(C)|_\Delta \subseteq T(A)|_\Delta$.*

3.2 Strong Observational Refinement

Attiya and Enea [2] have proposed the notion of *strong observational refinement*, which is a strengthening of refinement (and generalisation of strong linearizability [13]) that preserves all hyperproperties. Strong observational refinement is defined in terms of an adversary and is modelled by a scheduler that is assumed to have full control over a step-deterministic LTS's execution.

Formally, a *scheduler* for an LTS is a function $S : \Sigma^* \rightarrow 2^\Sigma$ that determines the next action to be executed based on the sequence of actions that have been executed thus far. A trace ρ is *consistent* with a scheduler S if $\rho[n] \in S(\rho^{<n})$ for all $n < \#\rho$. We write $T(L, S)$ for the set of traces of L that are consistent with S . A scheduler is *admitted* by an LTS L if for all finite traces σ of L consistent with S , the scheduler satisfies

⁵ It is well known that forward simulation is sound for proving refinement, but completeness requires both forward and backward simulation [7, 22].

1. $S(\sigma)$ is non-empty and
2. all actions in $S(\sigma)$ are enabled in $state(\sigma)$.

The scheduled traces in $T(L, S)$ can alternatively be viewed as the traces of a product $L \times LTS(S)$, where $LTS(S)$ is an LTS generated from S with set of states Σ^* ; initial state ε ; and transitions $\sigma \xrightarrow{a} \sigma \cdot a$, where $a \in S(\sigma)$.

In addition to being admissible, the schedulers we consider (for the combination of program with object, $P \times O$) must be *deterministic*: they must deterministically choose one of the enabled actions of the object. A scheduler S for $P \times O$ is *deterministic* if either (i) $S(\sigma) \subseteq \Gamma_P$ (i.e., it can choose several program actions, excluding invocations and responses of object operations) or (ii) $|S(\sigma)| = 1$ (i.e., if S chooses an action of O , including invocations and responses, then it chooses exactly one).

Now we are ready to define strong observational refinement.

► **Definition 3.4** (Strong observational refinement). *An object C strongly observationally refines an object A , written $C \leq_s A$, iff for every program P and every deterministic scheduler S_C admitted by $P \times C$, there exists a deterministic scheduler S_A admitted by $P \times A$ such that $T(P \times C, S_C)|_{\Gamma_P} = T(P \times A, S_A)|_{\Gamma_P}$.*

Note that unlike Attiya and Enea [2], this definition of strong observational refinement considers infinite traces, which is necessary for preservation of all hyperproperties. In this setting, as discussed in §2, forward simulation is no longer necessary and sufficient for establishing strong observational refinement (contrasting the results of Attiya and Enea [2]).

4 A necessary and sufficient condition

We now motivate and develop the notion of *weak progressive forward simulation*, providing a proof method for strong observational refinement. We first recap progressive forward simulation [8] and show that it is *not* a necessary condition (§4.1). Our new relaxed definition is given in §4.2.

4.1 Progressive forward simulation is too strong

In [8], we developed a condition called *progressive forward simulation* that enhances forward simulation with a well-founded order that rules out infinite stuttering. It is guaranteed by an implementation, e.g., when the underlying implementation is lock-free [6, 18]. First we provide the formal definition of progressive forward simulation.

► **Definition 4.1** (Progressive Forward Simulation [8]). *Let C and A be two deterministic LTSs and $\Delta \subseteq \Sigma_C \cup \Sigma_A$. A relation $F \subseteq Q_C \times Q_A$ together with a well-founded order $\gg \subseteq Q_C \times Q_C$ is called a progressive Δ -forward simulation from C to A iff*

Initialisation. $(q_C^{ini}, q_A^{ini}) \in F$,

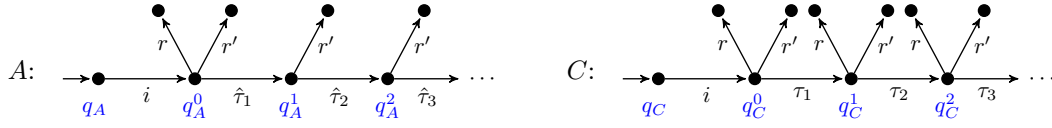
Step. For all $(q_C, q_A) \in F$, if $q_C \xrightarrow{a} q'_C$ then there exist $\sigma \in \Sigma_A^*$ and $q'_A \in Q_A$ such that

Simulation. $a|_{\Delta} = \sigma|_{\Delta}$, $q_A \xrightarrow{\sigma} q'_A$ and $(q'_C, q'_A) \in F$, and

Progressiveness. if $\sigma = \varepsilon$ then $q_C \gg q'_C$.

In [8], we have additionally shown that progressive forward simulation is *sufficient* for strong observational refinement.

► **Theorem 4.2** (Sufficiency [8]). *If there exists a progressive forward simulation between C and A , then $C \leq_s A$.*



■ **Figure 3** Two objects A and C such that $C \leq_s A$, but there does not exist a progressive forward simulation between C and A .

The main motivation for this paper has been the pursuit of a proof in the other direction, i.e., that progressive forward simulation is also *necessary* for strong observational refinement. However, it turns out that strong observational refinement does *not* imply the existence of a progressive forward simulation.

To see this consider the labelled transition systems depicted in Figure 3. We assume an abstract object A and concrete implementation C with a single operation, external actions $i, r, r' \in I \cup R$, abstract internal actions $\hat{\tau}_k \in \Sigma_A \setminus (I \cup R)$, and concrete internal actions $\tau_k \in \Sigma_C \setminus (I \cup R)$. The objects A and C differ in that C continually allows both r and r' after c , whereas A only allows both r and r' immediately after i ; it stops offering r after $\hat{\tau}_1$.

It is straightforward to show that $C \leq_s A$:

1. if C generates a run $q_C \cdot i \cdot q'_C \cdot r$ or $q_C \cdot i \cdot q'_C \cdot r'$ without executing any internal actions after i , a corresponding run can clearly be generated by A ;
2. if C generates a run $q_C \cdot i \cdot q_C^0 \cdot \tau_1 \cdots \tau_n \cdot q_C^n \cdot r$ or $q_C \cdot i \cdot q_C^0 \cdot \tau_1 \cdots \tau_n \cdot q_C^n \cdot r'$ executing the internal actions τ_1, \dots, τ_n , then a corresponding run can be executed in A by not executing any of the internal actions of A ;
3. if C generates a (diverging) run $q_C \cdot i \cdot q_C^0 \cdot \tau_1 \cdot q_C^1 \cdot \tau_2 \cdots$ that never responds, a corresponding run $q_A \cdot i \cdot q_A^0 \cdot \hat{\tau}_1 \cdot q_A^1 \cdot \hat{\tau}_2 \cdots$ can be generated in A .

Thus, for any program P and scheduler S_C , there exists a scheduler S_A such that $T(P \times C, S_C)|_{\Gamma_P} = T(P \times A, S_A)|_{\Gamma_P}$.

Now we show that there does not exist a progressive forward simulation. First, there exists a forward simulation, F , that allows each τ_k to behave as the corresponding $\hat{\tau}_k$ and as a stuttering step, i.e., $(q_C^j, q_A^0) \in F$ for each j . However, there is no well-founded ordering over the states since stuttering is unbounded, thus progressiveness cannot be guaranteed. The problem is that progressiveness enforces **2** above, but does not account for the possibility of **3**.

The main result of this work is a weaker form of progressive forward simulation that we prove to *coincide* with strong observational refinement. *Weak progressiveness* does not necessitate a well-founded order when the concrete implementation executes an infinite number of consecutive internal actions provided that the abstraction can also execute an infinite number of consecutive internal actions. This relaxation accounts for the scenario highlighted by **3** above.

4.2 Weak progressive forward simulation

In our LTSs, distinguishing between internal and external actions allows us to define *divergent* states. State q is Δ -divergent (written $q \xrightarrow{\infty}_{\Delta}$) if there exists an infinite run $q \cdot a_1 \cdot q_1 \cdot a_2 \cdots$ such that $a_i \in \Sigma \setminus \Delta$ for all $i \geq 1$.

We therefore obtain the following definition of weak progressive forward simulation, which relaxes the progressiveness condition from Definition 4.1.

► **Definition 4.3** (Weak Progressive Forward Simulation). *Let C and A be two deterministic LTSs and $\Delta \subseteq \Sigma_C \cup \Sigma_A$. A relation $F \subseteq Q_C \times Q_A$ together with a well-founded order $\gg \subseteq Q_C \times Q_C$ is called a weak progressive Δ -forward simulation from C to A iff*

Initialisation. $(q_C^{ini}, q_A^{ini}) \in F$,

Step. For all $(q_C, q_A) \in F$, if $q_C \xrightarrow{a}_C q'_C$ then there exist $\sigma \in \Sigma_A^*$ and $q'_A \in Q_A$ such that

Simulation. $a|_\Delta = \sigma|_\Delta$, $q_A \xrightarrow{\sigma}_A q'_A$ and $(q'_C, q'_A) \in F$, and

Weak progressiveness. if $\sigma = \varepsilon$ then either $q_C \gg q'_C$ or $q_A \xrightarrow{\infty}_{\Delta}$.

Note that we do not require that any triangular diagram with $q_C \xrightarrow{a}_C q'_C$, $(q_C, q_A) \in F$, and $(q'_C, q_A) \in F$ with no diverging trace from q_A to have $q_C \gg q'_C$. We only require this if there is no other $(q'_C, q'_A) \in F$ with $q_A \xrightarrow{\sigma}_A q'_A$ and $\sigma \neq \varepsilon$.

Also note that for concrete LTSs without any divergence, all three notions (forward simulation, strong observational refinement and weak progressive forward simulation) coincide (because without divergence $s \gg s'$ iff $s \xrightarrow{\tau} s'$ for some internal action τ is a well-founded order). For example, progress conditions such as *lock-freedom* [18] would be sufficient to ensure absence of divergence in the concurrent object (see also [11, 14, 17]).

This definition weakens the progressiveness condition: Either the concrete state must decrease in the well-founded order, or q_C corresponds to a q_A that diverges. A standard forward simulation would allow one to relate all concrete states of the diverging run to q_A , “hiding” the divergence. Weak progressiveness ensures that divergence on the concrete level is not possible without a corresponding diverging run from q_A . Our earlier definition of a *progressive forward simulation* always required the well-founded ordering to decrease for a stuttering concrete transition. With this change in place, we can now show equality of strong observational refinement and this form of forward simulation.

► **Theorem 4.4.** $C \leq_s A$ iff there exists a weak progressive $(I \cup R)$ -forward simulation from C to A .

The rest of the paper is now devoted to proving this theorem. We prove sufficiency in §5 and necessity in §6.

5 Weak Progressive Forward Simulation implies Strong Observational Refinement

We start with the sufficiency of weak progressive forward simulation for strong observational refinement. This proof is an adaptation of the proof for progressive forward simulation [8, 9], so we relegate the details to the appendix.

► **Theorem 5.1.** *If there exists a weak progressive $I \cup R$ -forward simulation from C to A , then $C \leq_S A$.*

Given two LTLs C and A for which a weak progressive forward simulation (F, \gg) exists and given an arbitrary program P together with a scheduler S_C for traces over $P \times C$, our proof has to construct a scheduler S_A such that $T(P \times C, S_C)|_{\Gamma_P} = T(P \times A, S_A)|_{\Gamma_P}$. The construction is in two steps: First a function f is constructed that maps traces $\rho_C \in T(P \times C, S_C)$ to traces $f(\rho_C) \in T(P \times A)$, such that the executed program actions in Σ_P are the same for both traces.

The construction is shown in Fig. 4. Steps of C are mapped to fixed steps of A using a mapping m (a formal definition is in the appendix), such that the forward simulation is preserved. Program steps in Γ_P are mapped by identity, such that the program states of both traces are always equal. For finite traces $\rho_C \in T(P \times C, S_C)$ this results in a finite trace with the same program steps.

$$\begin{array}{ccccccc}
& & m(q_C^{ini}, a_1, q_A^{ini}) & & m(q_C^{ini}, i_2, q_A^{ini}) & = \epsilon & m(q_C^2, \tau_3, q_A^1) \\
& & = a_1 & & = \alpha_1 i_2 \alpha_2 & & = \alpha & m(q_C^2, \tau_4, q_A^1) \\
f(\pi_C) : & (q_P^{ini}, q_A^{ini}) & \longrightarrow & (q_P^1, q_A^{ini}) & \longrightarrow & (q_P^2, q_A^1) & \longrightarrow & (q_P^2, q_A^3) \longrightarrow \dots \\
& \Big\downarrow F & & \Big\downarrow F & & \Big\downarrow F & & \Big\downarrow F \\
\pi_C : & (q_P^{ini}, q_C^{ini}) & \longrightarrow & (q_P^1, q_C^{ini}) & \longrightarrow & (q_P^2, q_C^1) & \longrightarrow & (q_P^2, q_C^2) \longrightarrow \dots \\
& & a_1 \in \Gamma_P & & i_2 \in I & & \tau_3 & & \tau_4
\end{array}$$

■ **Figure 4** Constructing $f(\pi_C) \in T(P \times A)$ from $\pi_C \in T(P \times C, S_C)$ with $\tau_3, \tau_4 \in \Sigma_C \setminus (I \cup R)$, $\alpha, \alpha_1, \alpha_2 \in (\Sigma_A \setminus (I \cup R))^*$ and $q_C^1 \gg q_C^2$.

Infinite traces $\pi_C \in T(P \times C, S_C)$ are mapped either to infinite traces directly, or by exploiting that the forward simulation is weakly progressive: if the abstract trace is finite, ending with (q_P, q_A) while the concrete trace ends with an infinite sequence of stuttering steps, then a diverging run of A from q_A is guaranteed to exist. This run (where all A -states are combined with q_P) can then be attached at the end to give an infinite trace which is defined to be $f(\pi_C)$. Again, π_C and $f(\pi_C)$ will have the same program steps.

A formal definition of f will be given in the appendix. Given f , an abstract scheduler S_A can be defined that schedules exactly all the steps of $f(\rho_C)$. For this definition to be well-defined it is crucial that the traces in the image of f form a tree-shaped structure, where branching points are at program actions only. We have the following theorem.

► **Theorem 5.2.** $T(P \times A, S_A) = \{\sigma_A \mid \exists \pi_C \in T(P \times C, S_C). \sigma_A \sqsubseteq f(\pi_C)\}$.

Theorem 5.1 then is a simple consequence, since both π_C and $f(\pi_C)$ have the same program actions, and each $\pi_A \in T(P \times A, S_A)$ is some $f(\pi_C)$ as stated by Theorem 5.2.

6 Strong Observational Refinement implies Weak Progressive Forward Simulation

We now prove the necessity theorem for weak progressive forward simulation.

► **Theorem 6.1.** *If $C \leq_s A$, then there exists a weak progressive $(I \cup R)$ -forward simulation from C to A .*

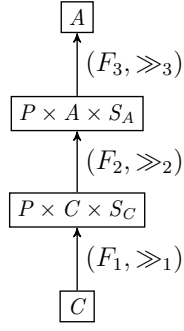
Given that C is a strong observational refinement of A , we must show that a weak progressive forward simulation exists between A and C . Since we are tasked with finding a forward simulation, we must also instantiate a client program and concrete scheduler that act as witnesses to the forward simulation. Our proof proceeds in stages (see Figure 5) for a client program P that invokes the operations of the object in question and concrete scheduler S_C . This method is similar to that of Attiya and Enea [2], but the underlying formal mechanisms have been completely reworked.

Client Program and Concrete Scheduler

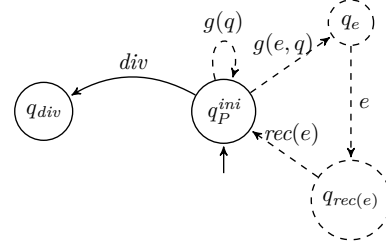
The program for concrete object $C = (Q_C, Q_C^{ini}, \Sigma_C, \delta_C)$ that we use is given by the LTS $P = (Q_P, Q_P^{ini}, \Sigma_P, \delta_P)$, where

- $Q_P = \{q_P^{ini}, q_{div}\} \cup \{q_e, q_{rec(e)} \mid e \in I \cup R\}$,
- $Q_P^{ini} = \{q_P^{ini}\}$,
- $\Gamma_P = \{div\} \cup \{g(q), g(e, q), rec(e) \mid q \in Q_C \wedge e \in I \cup R\}$,

31:10 Weak Progressive Forward Simulation Is Necessary and Sufficient



■ **Figure 5** Proof overview.



■ **Figure 6** Representation of client P used as a witness to the forward simulation, where $q \in Q_C$ and $e \in I \cup R$.

- $\Sigma_P = \Gamma_P \cup I \cup R$,
- $\delta_P = \{q_P^{ini} \xrightarrow{div} q_{div}\} \cup \bigcup_{e \in I \cup R, q \in Q_C} \{q_P^{ini} \xrightarrow{g(e, q)} q_e, q_e \xrightarrow{e} q_{rec(e)}, q_{rec(e)} \xrightarrow{rec(e)} q_P^{ini}, q_P^{ini} \xrightarrow{g(q)} q_P^{ini}\}$

This program is depicted in Figure 6, where dashed states and transitions are used to denote *families* of states and transitions. We refer to $g(q)$ and $g(e, q)$ as *guess actions* and $rec(e)$ as *record actions*. These are used to make the (internal) choices made by the client program, concrete object and scheduler visible in the traces of $T(P \times C \times S_C)|_{\Gamma_P}$. Additionally, we use an external action div that is enabled whenever the underlying object can diverge, i.e., div is enabled in q_C iff $q_C \xrightarrow{\infty}_{\setminus(I \cup R)}$. The program P can only synchronise with div when it is in state q_P^{ini} . Once executed, the program transitions to state q_{div} and from this point, it is only possible to schedule *internal actions* of C in $P \times C$.

We now define a particular admissible scheduler. The concrete scheduler S_C must schedule the actions of $P \times C$, i.e., define the next action for a given trace $\sigma \in (\Sigma_P \cup \Sigma_C)^*$.

$$S_C(\sigma) = \begin{cases} \{\tau\} & \text{if } div \in \sigma, \tau \in \Sigma_C \setminus (I \cup R), \text{ and} \\ & \exists q_C. state(\sigma|_{\Sigma_C}) \xrightarrow{\tau_C} q_C \wedge q_C \xrightarrow{\infty}_{\setminus(I \cup R)} \\ \{\tau\} & \text{else if } last(\sigma) = g(q_C) \text{ and} \\ & \tau \in \Sigma_C \setminus (I \cup R) \wedge state(\sigma|_{\Sigma_C}) \xrightarrow{\tau} q_C \\ \{e\} & \text{else if } last(\sigma) = g(e, q_C) \\ \{rec(e)\} & \text{else if } last(\sigma) = e \\ \{div \mid state(\sigma|_{\Sigma_C}) \xrightarrow{\infty}_{\setminus(I \cup R)}\} \cup & \text{otherwise} \\ \{g(q_C) \mid \exists \tau \in \Sigma_C \setminus (I \cup R). state(\sigma|_{\Sigma_C}) \xrightarrow{\tau} q_C\} \\ \cup \{g(e, q_C) \mid \exists e \in I \cup R. state(\sigma|_{\Sigma_C}) \xrightarrow{e} q_C\} \end{cases}$$

Note that $state(\sigma|_{\Sigma_C})$ is calculated wrt the LTS C as opposed to the composition $P \times C$. Furthermore, the scheduler decides on the next action of $P \times C$ based on the last action in the given trace σ . The first two cases determine the next action of C depending on whether the program has executed div . In the first two cases, there may be a choice of τ , the scheduler chooses one such that the resulting set is a singleton. Clearly the third case results in a singleton set since the action $g(e, q_C)$ fixes the only external action e allowed by the scheduler. Therefore, the scheduler S_C defined above is deterministic.

The last two cases describe the scheduler's behaviour wrt to a program action. As per Figure 6, these are the record and guess actions as well as div . In the fourth case, action $rec(e)$ must be scheduled if the last action executed in σ is e . In the final case, the scheduler may choose to diverge (if the program diverges), perform a guess action, $g(q_C)$ (corresponding

to an internal transition of C) or a guess action, $g(e, q_C)$ (corresponding to an external transition of C). Note that for both $g(q_C)$ and $g(e, q_C)$, state q_C is the post state of the given action after executing from $state(\sigma)$.

By design, we therefore have the following proposition for $P \times C$.

► **Proposition 6.2.** *Let $\sigma \in T(P \times C, S_C)$ for the scheduler S_C . If $div \in \sigma$ holds then $state(\sigma|_{\Sigma_C}) \xrightarrow{\infty}_{\setminus(I \cup R)}$ and $(\exists q'_C. state(\sigma|_{\Sigma_C}) \xrightarrow{S_C(\sigma)} q'_C \wedge q'_C \xrightarrow{\infty}_{\setminus(I \cup R)})$.*

Simulation (F_1, \ggg_1)

Our first step is the construction of a weak progressive forward simulation (F_1, \ggg_1) between C and $P \times C \times S_C$ (see Figure 5) with $F_1 \subseteq Q_C \times (Q_P \times Q_C \times \Sigma_{P \times C}^*)$. We define F_1 such that $(q_C, (q_P, q_C, \sigma)) \in F_1$ iff

- $q_C \xrightarrow{q_C^{ini} \sigma|_{\Sigma_C}} q_C$,
- $div \notin \sigma$ and $last(\sigma) \notin \{g(q), g(e, q), e \mid q \in Q_C \wedge e \in I \cup R\}$ (so that σ is either empty or ends with $rec(e)$ or an internal action; thus the next step is a guessing step),
- $q_P = q_P^{ini}$, and
- $state(\sigma) = q_C$.

► **Lemma 6.3.** *(F_1, \emptyset) is a weak progressive forward simulation from C to $P \times C \times S_C$.*

Simulation (F_2, \ggg_2)

Our next step is to show that a weak progressive forward simulation (F_2, \ggg_2) from $P \times C \times S_C$ to $P \times A \times S_A$ exists, when S_A is any scheduler with $T(P \times C, S_C)|_{\Gamma_P} = T(P \times A, S_A)|_{\Gamma_P}$ that exists due to the assumption $C \leq_s A$. The proof follows from a general completeness result for refinements for Δ -deterministic systems.

► **Lemma 6.4.** *If C Δ -refines A and A is Δ -deterministic, then there exists a Δ -forward simulation.*

A Δ -deterministic LTS is one, where every history $h \in \Delta^*$ has a unique state q that can be reached with shortest executions that have history h . (A formal definition of Δ -deterministic LTSs is given in Definition B.1.) Such a shortest execution is empty, if $h = \epsilon$, and otherwise has the last element of h as the action of its last step (note, that this definition of Δ -deterministic is weaker than the ones given in [2] and [22]). Clearly, $P \times A \times S_A$ is Γ_P -deterministic, so the theorem applies. The proof of 6.4 shown in the appendix constructs a forward simulation F_2 that relates all states of $P \times C \times S_C$ that are reached with history h to the unique minimally reachable state of $P \times A \times S_A$ with the same history. F_2 is weak progressive, since S_C never schedules more than one internal action in a row. Our definition of the program P guarantees that F_2 also preserves the actions $e \in I \cup R$, since each of these is followed by the corresponding $rec(e)$ action, that is already preserved.

► **Lemma 6.5.** *There exists a weak progressive Σ_P -forward simulation (F_2, \ggg_2) between $P \times C \times S_C$ and $P \times A \times S_A$.*

Simulation (F_3, \ggg_3)

We now define the weak progressive $(I \cup R)$ -forward simulation F_3 between $P \times A \times S_A$ and A . The states of $P \times A \times S_A$ includes those of A . Thus we keep the two LTSs synchronised, i.e., the forward simulation is over pairs of the form $((q_P, q_A, \sigma), q_A)$, where $(q_P, q_A, \sigma) \in Q_{P \times A \times S_A}$. For $(q_P, q_A, \sigma) \in Q_{P \times A \times S_A}$, let $F_3 = \{((q_P, q_A, \sigma), q_A) \mid (q_P, q_A, \sigma) \in Q_{P \times A \times S_A}\}$.

31:12 Weak Progressive Forward Simulation Is Necessary and Sufficient

The well-founded ordering that we use is the relation \gg_3 , where:

1. $(q_P^{ini}, _, _) \gg_3 (q_e, _, _)$
2. $(q_{rec(e)}, _, _) \gg_3 (q_P^{ini}, _, _)$
3. $(q_P^{ini}, _, _) \gg_3 (q_{div}, _, _)$

► **Lemma 6.6.** (F_3, \gg_3) is a weak progressive forward $(I \cup R)$ -simulation between $P \times A \times S_A$ and A .

It is trivial to prove that F_3 is a forward simulation. We therefore focus on a proof of weak progressiveness, which provides further insight into our choice of P and the inclusion of the div action in our model.

Note that from $(q_{div}, _, _)$, the only possible transition is an \xrightarrow{a} step, where $a \in \Sigma_A \setminus (I \cup R)$, which is non-stuttering. Similarly, from $(q_e, _, _)$, the only possible transition is \xrightarrow{e} , where $e \in I \cup R$. Thus, when we reach a state that is minimal wrt \gg_3 , no more stuttering is possible. Any transition from state $(q_{rec(e)}, _, _)$ is guaranteed to reduce w.r.t. \gg_3 , as are transitions corresponding to $g(e, q)$ and div from $(q_P^{ini}, _, _)$.

This leaves us with transitions corresponding to $g(q)$ from (q_P^{ini}, q_A, σ) , which may stutter infinitely often. We can show that such stuttering only exists if A contains a diverging run from q_A , i.e., div is enabled in $(q_P^{ini}, q_A, \sigma) \in Q_{P \times A \times S_A}$.

Suppose there exists an infinite run

$$(q_P^{ini}, q_A, \sigma) \xrightarrow{g(q_1)} (q_P^{ini}, q_A, \sigma \cdot g(q_1)) \xrightarrow{g(q_2)} (q_P^{ini}, q_A, \sigma \cdot g(q_1) \cdot g(q_2)) \xrightarrow{g(q_3)} \dots$$

By construction, $P \times A \times S_A$ is an “abstraction” of $P \times C \times S_C$ such that $T(P \times A, S_A)|_\Gamma = T(P \times C, S_C)|_\Gamma$, thus, $(\sigma|_\Gamma) \cdot g(q_1) \cdot g(q_2) \cdot g(q_3) \cdot \dots \in T(P \times C, S_C)|_\Gamma$. Thus, there exists a q_C such that $last(\sigma) = g(q_C)$ and

$$(q_P^{ini}, q_C, \sigma) \xrightarrow{g(q_1) \cdot \tau_1} (q_P^{ini}, q_1, \sigma \cdot g(q_1) \cdot \tau_1) \xrightarrow{g(q_2) \cdot \tau_2} (q_P^{ini}, q_2, \sigma \cdot g(q_1) \cdot \tau_1 \cdot g(q_2) \cdot \tau_2) \xrightarrow{g(q_3) \cdot \tau_3} \dots$$

where $\tau_k \in \Sigma_C \setminus (I \cup R)$ for all k . Note that the definition of S_C enforces a $\xrightarrow{g(q_k) \cdot \tau_k}$ transition for each $\xrightarrow{g(q_k)}$ transition in $P \times A \times S_A$. This execution, when restricted to the actions of C corresponds to a diverging run of C :

$$q_C \xrightarrow{\tau_1} q_1 \xrightarrow{\tau_2} q_2 \xrightarrow{\tau_3} \dots$$

Since this is an infinite run of internal actions, by definition, the action div must be offered by $P \times C \times S_C$, and enabled in (q_P^{ini}, q_C, σ) . Moreover, since $T(P \times A, S_A)|_\Gamma = T(P \times C, S_C)|_\Gamma$, div must also be possible in $P \times A \times S_A$. In particular, div must be enabled in (q_P^{ini}, q_A, σ) . Now, $P \times A \times S_A$ contains a run with final state σ . Therefore, $P \times A \times S_A$ also contains a run

$$(q_P^{ini}, q_A, \sigma) \xrightarrow{div} (q_{div}, q_A, \sigma) \xrightarrow{\tau'_1} (q_{div}, q'_1, \sigma) \xrightarrow{\tau'_2} (q_{div}, q'_2, \sigma) \xrightarrow{\tau'_3} \dots$$

where $\tau'_k \in \Sigma_A \setminus (I \cup R)$ for all k since $P \times A \times S_A$ can no longer schedule any further external actions after executing div , i.e., must schedule an internal action. Thus, we must have a diverging run in A as well.

Combined simulation. Finally, to derive at a weak progressive simulation from C to A , we show that the relation of weak progressive forward simulation is transitive.

► **Theorem 6.7.** Let (F_1, \gg_1) be a weak progressive Δ -forward simulation from C to B , and (F_2, \gg_2) one from B to A . Then there exists a weak progressive Δ -forward simulation (F, \gg) from C to A .

The proof of this theorem uses $F = F_1 \circ F_2$ and \gg as defined by $q_C \gg q'_C$ if $q_C \xrightarrow{a} q'_C$ for an internal action $a \notin \Delta$ such that one of the following two conditions holds:

- (1) $\exists q_B. (q_C, q_B) \in F_1 \wedge (q'_C, q_B) \in F_1 \wedge q_C \gg_1 q'_C \wedge \neg q_B \xrightarrow{\infty} \setminus \Delta,$
- (2) $\exists q_B, \alpha, q'_B, q_A. (q_C, q_B) \in F_1 \wedge (q'_C, q'_B) \in F_1 \wedge q_B \xrightarrow{\alpha} q'_B \wedge q_B \gg_2 q'_B$
 $\wedge (q_B, q_A) \in F_2 \wedge (q'_B, q_A) \in F_2 \wedge \neg q_A \xrightarrow{\infty} \setminus \Delta$

where α is a finite sequence of internal actions.

Case (1) requires a triangular diagram for the lower simulation from B to C with a non-diverging state q_B , where \gg_1 decreases. Case (2) requires an arbitrary commuting diagram for the lower simulation, and a triangular diagram for the upper simulation from A to B , where the abstract state q_A does not have a diverging run, and \gg_2 decreases.

7 Progressive and weak progressive examples

We now present two example programs to demonstrate the implications of progressive and weak progressive forward simulation on program design. The first satisfies progressive forward simulation (and hence weak progressive forward simulation) w.r.t. its abstract specification, while the second satisfies weak progressive simulation only.

7.1 FAI with Lock-free LL/SC

Consider the FAI implementation from Figure 1, but where the LL/SC is assumed to be lock-free. We refer to this implementation as FAI-LF. Unlike the example in §2, we assume that an LL operation executed by one thread does not interfere with an LL in another thread. If two concurrent threads have loaded the same LL value, then only one SC will succeed. Forward simulation for FAI-LF holds for the same reason as FAI. We now define a global well-founded order over states using the technique described in [6], which implies a weak progressive forward simulation. This in turn guarantees that *all* hyperproperties (including hyperliveness) of the abstract specification are preserved by FAI-LF.

The well-founded order is straightforward to define: it is a lexicographic ordering that captures how “close” a thread is to successfully executing a successful SC operation. The base of the well-founded ordering guarantees that some thread will successfully execute its SC operation. The generic lexicographic scheme is the following, where b, b' are booleans and pc, pc' are program counter values:

$$b \gg_B b' \hat{=} b \wedge \neg b'$$

$$(b, pc) \gg_L (b', pc') \hat{=} b \gg_B b' \vee (b' = b \wedge (b \wedge pc \gg_{\checkmark} pc') \wedge (\neg b \wedge pc \gg_{\times} pc'))$$

where \gg_B orders *true* before *false* and \gg_L is a lexicographic order with a different orderings on pc depending on whether or not b holds. We instantiate this generic scheme over states as follows, where $current_val, n_t$ and pc_t are the variables of the algorithm in Figure 1 for a thread t . In particular, $current_val$ corresponds to the shared variable `current_val`, n_t corresponds to the local variable `n` of thread t , and pc_t is the program counter for thread t taking values from the set `{F1, F2, F3, F4, idle}`.

$$q \gg q' \hat{=} \exists t. (q(current_val) = q(n_t), q(pc_t)) \gg_L (q'(current_val) = q'(n_t), q'(pc_t))$$

All that remains is the instantiation of \gg_{\checkmark} and \gg_{\times} . The order \gg_{\checkmark} is empty, since $q(current_val) = q(n_t)$ implies that $q(pc_t) = \text{F3}$, and execution of thread t corresponds to a successful SC operation. We define $\text{F2} \gg_{\times} \text{F3}$, representing a retry since this order allows t to make progress towards making $q(current_val) = q(n_t)$ true.

7.2 FAI with backoff

We now consider a load-balancing FAI specification, which we refer to as FAI-LB. In this example, we weaken the specification to either perform the FAI or perform an operation $backoff_t$ for a thread t that causes the FAI executed by thread t to be delayed. Abstractly, $backoff_t$ is equivalent to a skip action. Note that although the resulting specification is non-deterministic, the LTS is still step-deterministic since the execution of each action from any state results in exactly one next state.

For FAI-LB, we can prove weak progressive forward simulation even for the obstruction-free FAI implementation from §2. Informally, the interference caused by an LL by thread t on another thread's SC can be mapped to a $backoff_t$ operation executed by thread t . Thus, although the obstruction-free implementation in §2 has a divergent execution, this execution can be matched by the specification FAI-LB.

8 Related work

The study of refinement, and in particular linearizability [19], in the context of adversaries was initiated by the work of Golab, Higham and Woelfel [13] who observed that replacing atomic objects by linearizable implementations in randomized algorithms [1] does not guarantee the expected substitutability result of linearizability. Instead, the probability distribution of results may differ when using a linearizable implementation instead of an abstract atomic object. The difference is due to the abilities of adversaries scheduling process steps depending on the current system state. To alleviate this problem, Golab, Higham and Woelfel suggested *strong linearizability*, requiring a “prefix preservation” property in addition to the conditions of linearizability.

Following this proposal, Attiya and Enea studied the preservation of *hyperproperties* by linearizability. They proposed the definition of *strong observational refinement* and showed it to (a) preserve all hyperproperties, and (b) to coincide with strong linearizability for atomic abstract specifications. They also proved strong observational refinement to be equivalent to forward simulation. In a brief announcement [8], Derrick et al. gave a counter example to this proof, and provided an alternative result for one direction of the equivalence, proposing progressive forward simulation and proving it to imply strong observational refinement. Our work in this paper closes the missing gap of the relationship between progressive forward simulation and strong observational refinement by proving strong observational refinement to imply (yet another) version of forward simulation, weak progressive forward simulation. In addition, we strengthen the result of Derrick et al. [8] and show that weak progressive forward simulation also implies strong observational refinement, thereby arriving at an equivalence once again.

The relationship between (the standard definition of) linearizability and forward and backward simulation has already been investigated before, with Schellhorn, Wehrheim and Derrick [27,28] showing linearizability proofs in general to require *both* forward and backward simulations, and Bouajjani et al. [3] studying under what circumstances and how forward simulation alone can be employed. The relationship between observational refinement, safety (linearizability) and progress in the context of atomic objects has been studied in prior works [11,14]. The use of well-founded orderings to enforce progress for a forward simulation has already been used in the context of ASM refinement [25,26] and non-atomic refinement [10]. Another form of simulations employing well-founded orders are the normed (forward and backward) simulations of Griffioen and Vaandrager [15,16]. They require every matching internal (τ) step to decrease a norm defined on a well-founded set. It has been

shown that normed forward simulations do not agree with ordinary forward simulations, even on divergence-free LTSs. As weak progressive forward simulation does coincide with forward simulation on divergence-free LTSs, we thus get inequality of normed forward simulation and weak progressive forward simulation.

The study of notions of refinement and equivalence taking internal actions into account has been actively pursued in the field of process algebras, with weak bisimulation [24] for CCS and failures-divergences refinement [4] for CSP being the two most prominent examples. Failures-divergences refinement explicitly considers divergences (i.e., infinite sequences of internal actions) during the comparison. For bisimulation, there are also extensions for divergence, e.g. [31, 32]. A comparison of various such semantic equivalences and preorders for systems with internal actions has been given by van Glabbeek [30]. Finally, the game-theoretic characterisation of bisimulation [29] is in spirit similar to the idea of adversaries in strong observational refinement which try to bring the concrete object into an execution that can or cannot be mimicked by the abstract object.

9 Conclusion

In this paper, we have proposed a new type of forward simulation which is both necessary and sufficient for strong observational refinement, thereby closing an existing gap. The importance of strong observational refinement lays in the fact that it preserves safety and liveness hyperproperties which are themselves of fundamental significance for the area of security. As future work, we plan to look at concrete case studies, and to this end will develop a formalization of weak progressive forward simulation within a theorem prover. We furthermore plan to re-investigate the third contribution of Attiya and Enea [2], namely the fact that strong linearizability coincides with strong observational refinement for atomic abstract objects. Since the proof of this property assumes equality of forward simulation and strong observational refinement, this—in the light of our result—also requires a fresh investigation.

References

- 1 J. Aspnes. Randomized protocols for asynchronous consensus. *Distributed Comput.*, 16(2-3):165–175, 2003. doi:10.1007/s00446-002-0081-5.
- 2 H. Attiya and C. Enea. Putting strong linearizability in context: Preserving hyperproperties in programs that use concurrent objects. In J. Suomela, editor, *DISC*, volume 146 of *LIPICs*, pages 2:1–2:17. Schloss Dagstuhl, 2019. doi:10.4230/LIPICs.DISC.2019.2.
- 3 A. Bouajjani, M. Emmi, C. Enea, and S. O. Mutluergil. Proving linearizability using forward simulations. In R. Majumdar and V. Kuncak, editors, *CAV*, volume 10427 of *Lecture Notes in Computer Science*, pages 542–563. Springer, 2017. doi:10.1007/978-3-319-63390-9_28.
- 4 S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, 1984. doi:10.1145/828.833.
- 5 M. R. Clarkson and F. B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.
- 6 R. Colvin and B. Dongol. A general technique for proving lock-freedom. *Sci. Comput. Program.*, 74(3):143–165, 2009. doi:10.1016/j.scico.2008.09.013.
- 7 J. Derrick and E. A. Boiten. *Refinement in Z and Object-Z - Foundations and Advanced Applications (2. ed.)*. Springer, 2014. doi:10.1007/978-1-4471-5355-9.
- 8 J. Derrick, S. Doherty, B. Dongol, G. Schellhorn, and H. Wehrheim. Brief Announcement: On Strong Observational Refinement and Forward Simulation. In S. Gilbert, editor, *DISC*, volume 209 of *LIPICs*, pages 55:1–55:4, Dagstuhl, Germany, 2021. doi:10.4230/LIPICs.DISC.2021.55.

- 9 J. Derrick, S. Doherty, B. Dongol, G. Schellhorn, and H. Wehrheim. On strong observational refinement and forward simulation. *CoRR*, 2021. [arXiv:2107.14509](https://arxiv.org/abs/2107.14509).
- 10 J. Derrick, G. Schellhorn, and H. Wehrheim. Proving linearizability via non-atomic refinement. In J. Davies and J. Gibbons, editors, *iFM*, volume 4591 of *Lecture Notes in Computer Science*, pages 195–214. Springer, 2007. doi:10.1007/978-3-540-73210-5_11.
- 11 B. Dongol and L. Groves. Contextual trace refinement for concurrent objects: Safety and progress. In K. Ogata, M. Lawford, and S. Liu, editors, *ICFEM*, volume 10009 of *Lecture Notes in Computer Science*, pages 261–278, 2016. doi:10.1007/978-3-319-47846-3_17.
- 12 J. A. Goguen and J. Meseguer. Security policies and security models. In *S&P*, pages 11–20. IEEE Computer Society, 1982. doi:10.1109/SP.1982.10014.
- 13 W. M. Golab, L. Higham, and P. Woelfel. Linearizable implementations do not suffice for randomized distributed computation. In L. Fortnow and S. P. Vadhan, editors, *STOC*, pages 373–382. ACM, 2011. doi:10.1145/1993636.1993687.
- 14 A. Gotsman and H. Yang. Liveness-preserving atomicity abstraction. In L. Aceto, M. Henzinger, and J. Sgall, editors, *ICALP*, volume 6756 of *Lecture Notes in Computer Science*, pages 453–465. Springer, 2011. doi:10.1007/978-3-642-22012-8_36.
- 15 W. O. D. Griffioen and F. W. Vaandrager. Normed simulations. In A. J. Hu and M. Y. Vardi, editors, *CAV*, volume 1427 of *LNCS*, pages 332–344, Vancouver, BC, Canada, 1998.
- 16 W. O. D. Griffioen and F. W. Vaandrager. A theory of normed simulations. *ACM Trans. Comput. Log.*, 5(4):577–610, 2004. doi:10.1145/1024922.1024923.
- 17 M. Helmi, L. Higham, and P. Woelfel. Strongly linearizable implementations: possibilities and impossibilities. In D. Kowalski and A. Panconesi, editors, *PODC*, pages 385–394. ACM, 2012. doi:10.1145/2332432.2332508.
- 18 M. Herlihy and N. Shavit. *The art of multiprocessor programming*. Morgan Kaufmann, 2008.
- 19 M. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990. doi:10.1145/78969.78972.
- 20 V. Luchangeo, M. Moir, and N. Shavit. Nonblocking k-compare-single-swap. In A. L. Rosenberg and F. M. auf der Heide, editors, *SPAA*, pages 314–323. ACM, 2003. doi:10.1145/777412.777468.
- 21 N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
- 22 N. A. Lynch and F. W. Vaandrager. Forward and backward simulations: I. untimed systems. *Inf. Comput.*, 121(2):214–233, 1995. doi:10.1006/inco.1995.1134.
- 23 J. McLean. A general theory of composition for a class of "possibilistic" properties. *IEEE Trans. Software Eng.*, 22(1):53–67, 1996. doi:10.1109/32.481534.
- 24 R. Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.
- 25 G. Schellhorn. Verification of ASM Refinements Using Generalized Forward Simulation. *Journal of Universal Computer Science (J.UCS)*, 7(11):952–979, 2001.
- 26 G. Schellhorn. Completeness of Fair ASM Refinement. *Science of Computer Programming, Elsevier*, 76, issue 9:756–773, 2009.
- 27 G. Schellhorn, J. Derrick, and H. Wehrheim. A sound and complete proof technique for linearizability of concurrent data structures. *ACM Trans. Comput. Log.*, 15(4):31:1–31:37, 2014. doi:10.1145/2629496.
- 28 G. Schellhorn, H. Wehrheim, and J. Derrick. How to prove algorithms linearisable. In P. Madhusudan and S. A. Seshia, editors, *CAV*, volume 7358 of *Lecture Notes in Computer Science*, pages 243–259. Springer, 2012. doi:10.1007/978-3-642-31424-7_21.
- 29 P. Stevens. Abstract games for infinite state processes. In D. Sangiorgi and R. de Simone, editors, *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 147–162. Springer, 1998. doi:10.1007/BFb0055621.
- 30 R. J. van Glabbeek. The linear time - branching time spectrum II. In E. Best, editor, *CONCUR*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1993. doi:10.1007/3-540-57208-2_6.

- 31 R. J. van Glabbeek, B. Luttik, and N. Trcka. Branching bisimilarity with explicit divergence. *Fundam. Informaticae*, 93(4):371–392, 2009. doi:10.3233/FI-2009-109.
- 32 D. Walker. Bisimulation and divergence. *Information and Computation*, 85:202–241, 1990.
- 33 S. Zdancewic and A. C. Myers. Observational determinism for concurrent program security. In *CSFW-16*, page 29. IEEE Computer Society, 2003. doi:10.1109/CSFW.2003.1212703.

A Proofs for Section 5

The proof of theorem 5.1 assumes that two LTLs C and A are given, for which a weak progressive simulation (F, \gg) exists. Given an arbitrary program P together with a scheduler S_C for traces over $P \times C$, the proof has to construct a scheduler S_A such that $T(P \times C, S_C)|_{\Gamma_P} = T(P \times A, S_A)|_{\Gamma_P}$. The construction is in two steps: First a function f is constructed that maps traces $\rho_C \in T(P \times C, S_C)$ to traces $f(\rho_C) \in T(P \times A)$. This function has to be carefully defined to then allow the definition of a scheduler S_A that schedules exactly all the steps of $f(\rho_C)$. Weak progressiveness is key to ensure that for an infinite trace ρ_C the trace $f(\rho_C)$ is infinite as well. This then allows to schedule actions for any prefix.

The construction of f shown in Fig. 4 first has to fix a unique sequence of abstract actions in $f(\rho_C)$ that correspond to a single step of ρ_C . To this end, a mapping m is defined. For two states $q_C \in Q_C$ and $q_A \in Q_A$ with $(q_C, q_A) \in F$ and an action $a \in \Sigma_C$, m returns a fixed sequence $\sigma \in \Sigma_A^*$ such that $(q'_C, q'_A) \in F$ holds again for the (unique) states with $q_C \xrightarrow{a}_C q'_C$ and $q_A \xrightarrow{\sigma}_A q'_A$. Mapping m chooses a triangular diagram with $\sigma = \epsilon$ only, when there is no nonempty choice, so $q_C \gg q'_C$ is implied. The existence of σ is guaranteed by the main proof obligation for a weak progressive forward simulation. To be useful for constructing traces over $P \times A$ when a step of a trace over $P \times C$ is given, we extend the definition to allow a program action $a \in \Gamma_P$ as well. In this case m just returns the one element sequence of a . Intuitively, in addition to the commuting diagrams of the forward simulation this defines commuting diagrams that map program steps one-to-one. Formally,

$$m : Q_C \times (\Sigma_{P \times C}) \times Q_A \rightarrow (\Sigma_{P \times A})^*$$

is defined to return $m(q_C, a, q_A) := a$ when $a \in \Gamma_P$, and to return the fixed sequence σ as described above when $a \in \Sigma_C$.

It is then possible to define partial functions f_0, f_1, \dots (viewed as sets of pairs) with $\text{dom}(f_n) = \{\sigma_C \in T(P \times C, S_C) : \#\sigma_C \leq n\}$, $\text{cod}(f_n) \subseteq T(P \times A)$, such that $f_0 \subseteq f_1 \subseteq \dots$ inductively as follows:

$$\begin{aligned} f_0 &= \{(\epsilon, \epsilon)\} \\ f_{n+1} &= f_n \cup \{(\sigma_C \cdot a, f(\sigma_C) \cdot \alpha) \mid \sigma_C \cdot a \in T(P \times C, S_C), \#\sigma_C = n, \\ &\quad \alpha = m(\text{state}(\sigma_C).obj, a, \text{state}(f(\sigma_C)).obj)\} \end{aligned}$$

The inductive definition maps the new action $a \in S_C(\sigma_C)$ to the corresponding sequence α that is chosen by m . In the definition $(q_P, q_C).obj := q_C$ and the final state of σ_C is $\text{state}(\sigma_C) = (q_P, q_C)$. Analogously $(q_P, q_A).obj = q_A$.

The states $(q_P, q_C) = \text{state}(\sigma_C)$ and $(q'_P, q_A) = \text{state}(f_n(\sigma_C))$ reached at the end of two corresponding traces always satisfy $q_P = q'_P$ and $(q_C, q_A) \in F$. The use of m in the construction guarantees that all the f_n are prefix-monotone: if f_n is defined on σ and $\sigma' \sqsubseteq \sigma$, then $f_n(\sigma') \sqsubseteq f_n(\sigma)$.

Now, define $f := \bigcup_n f_n$. Function f is obviously prefix-monotone as well. Intuitively, it maps all finite traces of $T(P \times C, S_C)$ to a corresponding abstract trace, where m is used in each commuting diagram to choose the abstract action sequence.

31:18 Weak Progressive Forward Simulation Is Necessary and Sufficient

If π_C is an infinite trace from $T(P \times C, S_C)$, and $\sigma_A^n := f(\pi_C^{<n})$, then $\sigma_A^0 \sqsubseteq \sigma_A^1 \sqsubseteq \sigma_A^2 \sqsubseteq \dots$. Therefore a natural choice for extending f to infinite traces is to use the limit of this ascending chain. We define $f^{lim}(\pi_C)$ to be the limit and will use function f^{lim} in several of the lemmas below. There are two cases in this definition. Either the length of σ_A^n always eventually increases. Then the sequences converges to an infinite sequence $f^{lim}(\pi_C) = \pi_A \in T(P \times A)$. Otherwise, the σ_A^n eventually become a constant finite trace σ_A and we can set $f^{lim}(\pi_C) = \sigma_A$. In this case the final state $state(\sigma_A)$ must have a diverging run, since the forward simulation is weak progressive (otherwise the well-founded relation would have to decrease infinitely often, which is impossible). In this case a diverging run from $state(\sigma_A)$ can be fixed with an infinite sequence π_A of internal actions. Adding this infinite sequence to the trace is necessary for defining the abstract scheduler, so different from setting $f^{lim}(\pi_C) := \sigma_A$ we set $f(\pi_C) := \sigma_A \cdot \pi_A$.

We will now define a scheduler S_A , that will schedule exactly those traces in $\sigma_A \in T(P \times A)$ where σ_A is a prefix of some $f(\pi_C)$ such that π_C is an infinite trace in $T(P \times C, S_C)$. Before we can do this properly, a number of lemmas is needed.

► **Lemma A.1.** $f(\sigma_C)|\Gamma_P = \sigma_C|\Gamma_P$ for all $\sigma_C \in T(P \times C, S_C)$.

Proof. This should be obvious from the construction, since the forward simulation guarantees that $m(q_C, a, q_A)|\Gamma_P = a|\Gamma_P$ for all $a \in I \cup R$, while $a \in \Gamma_P$ is mapped by identity. ◀

► **Lemma A.2.** For two finite traces $\sigma_C, \sigma'_C \in T(P \times C, S_C)$: if $f(\sigma_C)$ and $f(\sigma'_C)$ have the same program actions in Γ_P , then σ_C is a prefix of σ'_C or vice versa, and the longer one just adds internal actions of C .

Proof. Lemma A.1 implies $\sigma_C|\Gamma_P = \sigma'_C|\Gamma_P$. If the lemma would be wrong, then there would be a maximal common prefix σ_0 and two actions $a \neq a'$ such that $\sigma_0 \cdot a \sqsubseteq \sigma_C$ and $\sigma_0 \cdot a' \sqsubseteq \sigma'_C$. The case where both a and a' are external actions is impossible, because otherwise the external actions in σ_C and σ'_C would not be the same. If however one of them is internal, then $S_C(\sigma_0)$ is a one-element set, and both a and a' must be in the set, contradicting $a \neq a'$. ◀

► **Lemma A.3.** For all finite prefixes σ_A of $f^{lim}(\pi_C)$, there is a unique n , such that $f(\pi_C^{<n}) \sqsubseteq \sigma_A \sqsubset f(\pi_C^{<n}) \cdot \alpha$, where $\alpha := m(state(\pi_C^{<n}), \pi[n], state(f(\pi_C^{<n}))) \cdot obj \neq \varepsilon$.

Intuitively, each element of $f^{lim}(\pi_C)$ is added by a uniquely defined commuting diagram.

Proof. First, note that $f(\pi_C^{<n+1}) = f(\pi_C^{<n}) \cdot \alpha$. Since the lengths of $f(\pi_C^{<n})$ are increasing with n to the length of $f^{lim}(\pi_C)$ (and $f(\pi_C^{<0}) = f(\varepsilon) = \varepsilon$) n is the biggest index where the length of $f(\pi_C^{<n})$ is still less or equal to $\# \sigma_A$. ◀

► **Lemma A.4.** Assume $\pi_C, \pi'_C \in T(P \times C, S_C)$. if σ_A is a prefix of both $f(\pi_C)$ and $f(\pi'_C)$, then there is m such that $\pi_C^{<m} = \pi'^C_{<m}$ and $\sigma_A \sqsubseteq f(\pi_C^{<m})$.

The lemma says, that a common prefix of two traces in the image of f is possible only as the result of a common prefix in the domain of f .

Proof. Since $\sigma_A \sqsubseteq f(\pi_C)$ and each step from $f(\pi_C^{<n})$ to $f(\pi_C^{<n+1})$ adds at most one program action, a minimal index n can be found such that σ_A has the same program actions as $f(\pi_C^{<n})$, while $f(\pi_C^{<n-1})$ has fewer when $n \neq 0$. Note that when $f^{lim}(\pi_C)$ is finite, n is less than its length, since the diverging run attached at the end has no program actions at all. Similarly, a minimal index n' can be found such that $\sigma_A|\Gamma_P = f(\pi'^C_{<n'})$. By Lemma A.2 above, it follows that $\pi_C^{<n}$ is a prefix of $\pi'^C_{<n'}$ or vice versa, with only internal C -actions

added to the longer one. When both are equal, then $n = n'$ and m can be set to be n . However, when the two are not equal, the longer one, say $\pi'^{<n'}$ ends with an internal C -action. But then, since this action is mapped to a sequence of internal A -actions $f(\pi'^{<n'-1})$ also has the same program actions than σ_A , contradicting the minimality of n' . ◀

Equipped with these lemmas, it is now possible to define the scheduler S_A and to prove it is well-defined. We define $S_A(\sigma_A)$ for any finite prefix σ_A of any $f(\pi_C)$, where $\pi_C \in T(P \times C, S_C)$. There are two cases. Either σ_A is not a prefix of $f^{lim}(\pi_C)$. Then it has the form $f^{lim}(\pi_C)\sigma'_A$ where σ'_A is a prefix of the infinite sequence of internal actions that is used in the definition of f in this case that schedules a diverging run. The next action to be scheduled then is the next action of this sequence. Otherwise the definition uses Lemma A.3 to find unique index n , such that $f(\pi_C^{<n}) \sqsubseteq \sigma_A \sqsubset f(\pi_C^{<n} \cdot \alpha)$ where $\alpha = m(\text{state}(\pi_C^{<n}).\text{obj}, \pi[n], \text{state}(f(\pi_C^{<n})).\text{obj}) \neq \epsilon$. Since σ_A is a proper prefix, there is an event a , such that $\sigma_A \cdot a \sqsubseteq f^{lim}(\pi_C^{<n}) \cdot \alpha$, and a is an element of α . If a is an external action in Γ_P , then a must be equal to $\pi_C[n]$ (α contains either $\pi_C[n]$ if it is an external action, or no external action at all). In this case, we set $S_A(\sigma_A) := S_C(\pi_C^{<n})$. Note that a is enabled and in $S_C(\pi_C^{<n})$ in this case. Otherwise, when $a \notin \Gamma_P$, we set $S_A(\sigma_A) := \{a\}$.

► **Theorem A.5.** S_A is well-defined.

Proof. Assume that σ_A is a prefix of two traces $f(\pi_C)$ and $f(\pi'_C)$. We prove that this never leads to two different definitions of $S_A(\sigma_A)$. First, Lemma A.4 gives an index m with $\pi_C^{<m} = \pi'_C^{<m}$ and $\sigma_A \sqsubseteq f(\pi_C^{<m})$. If σ_A is a proper prefix of $f(\pi_C^{<m})$, then the n used in the construction of S_A must satisfy $n + 1 \geq m$, and the prefix $f(\pi_C^{<n+1}) = f(\pi_C^{<n}) \cdot \alpha$ on which the definition of S_A is based, is the same for both traces. The remaining case is $m = n + 1$ and $\sigma_A = f(\pi_C^{<n+1})$. In this case the next elements $\pi_C[n + 1]$ and $\pi'_C[n + 1]$ in the two traces π_C and π'_C could be different. If one of them is internal (i.e. not in Γ_P), then this is not possible, since then $S_C(\pi_C^{<n+1})$ is a one-element set that contains both of them. However, it is possible that $\pi[n + 1]$ and $\pi'[n + 1]$ are two different program events $a \neq a'$, both in Γ_P , but in $S_C(\pi_C^{<m})$. However, in this case $S_A(f(\pi_C^{<m}))$ is defined in both cases to be $S_C(\pi_C^{<n+1})$. ◀

The following lemma is the inductive step of the theorem below, that shows that S_A allows exactly all $f(\pi_C)$ as scheduled traces.

► **Lemma A.6.** Given $\sigma_A \in T(P \times A, S_A)$, for which a $\pi_C \in T(P \times C, S_C)$ exists with $\sigma_A \sqsubseteq f(\pi_C)$, then $\sigma_A \cdot a \in T(P \times A, S_A)$ (or equivalently $a \in S_A(\sigma_A)$) is equivalent to the existence of some $\pi'_C \in T(P \times C, S_C)$ such that $\sigma_A \cdot a \sqsubseteq f(\pi'_C)$.

Proof. The case, where $\sigma_A \not\sqsubseteq f^{lim}(\pi_C)$ is simple, since after $f^{lim}(\pi_C)$ a unique diverging trace is attached that is the scheduled one. Otherwise, Lemma A.3 asserts that there is a unique n such that $f^{lim}(\pi_C^{<n}) \sqsubseteq \sigma_A \sqsubset f^{lim}(\pi_C^{<n+1})$. Let $\pi_C^{<n+1} = (\pi_C^{<n}) \cdot a$ and $\alpha = m(\text{state}(\pi_C^{<n}).\text{obj}, a, \text{state}(f(\pi_C^{<n})).\text{obj})$.

Case 1: $a \notin \Sigma_C$. Then $\alpha = a$, $a_1 = a$ by definition, implying $\sigma_A = f(\pi_C^{<n})$.

“ \Rightarrow ”: If $\sigma_A \cdot a \in T(P \times A, S_A)$, then $a \in S_A(\sigma_A)$ is equivalent to $a_1 \in S_C(\sigma_A)$, since $a_1 = a$ and $S_A(\sigma)$ is defined to be equal to $S_C(\pi_C^{<n})$. Since actions in $S_C(\pi_C^{<n})$ are enabled, and every finite trace can be extended to an infinite one, there is an infinite trace π'_1 with $(\pi_C^{<n}) \cdot a_1 \sqsubseteq \pi'_1$. π'_1 has the required prefix $\pi_C \cdot a_1$ such that $\sigma_A \cdot a_1 = f(\pi_C^{<n}) \cdot a$

“ \Leftarrow ”: if π'_C exists with $\sigma_A \cdot a \sqsubseteq f(\pi'_C)$, then like in the well-definedness proof $\pi_C^{<n}$ and $\pi'_1^{<n}$ must be the same (both have the same program actions as σ_A). Therefore $a_1 = a$ is scheduled after $\pi_C^{<n}$ as required.

31:20 Weak Progressive Forward Simulation Is Necessary and Sufficient

Case 2: $a \notin \Sigma_C$. Then α is a nonempty sequence of internal actions and α is the only continuation of $f(\pi_C^{<n})$ compatible with S_A . σ_A is $f(\pi_C^{<n})$ concatenated with some proper prefix of α .

“ \Rightarrow ”: If $\sigma_A \cdot a \in T(P \times A, S_A)$, then a must be the next element in α . Then, setting $\pi'_1 := \pi_C$ we get the required prefix $f(\pi_C^{<n+1}) = f(\pi_C^{<n}) \cdot \alpha$ of which $\sigma_A \cdot a$ is still a prefix.

“ \Leftarrow ”: Assume $\sigma_A \cdot a \sqsubseteq f(\pi'_1)$. Then $\sigma_A \cdot a$ is a prefix of both $f(\pi_C)$ and $f(\pi'_1)$, so Lemma A.3 implies that there is some m , such that $\sigma_A \cdot a \sqsubseteq \pi_1^{<m} = \pi_C^{<m}$. Obviously, $m \geq n+1$, so the next element after σ_A in π'_1 is the scheduled a too. \blacktriangleleft

With this, we are now ready to prove Theorem 5.2, which implies the main Theorem 5.1.

Proof. The proof is by contradiction. If the theorem does not hold, then there is a trace $\sigma_A \sqsubseteq T(P \times A, S_A)$ of minimal length and some action a , such that $a \in S_A(\sigma_A)$ is not equivalent to the existence of some $\pi'_C \in T(P \times C, S_C)$ such that $\sigma_A \cdot a \sqsubseteq f(\pi'_C)$. However, this equivalence is asserted by Lemma A.6. \blacktriangleleft

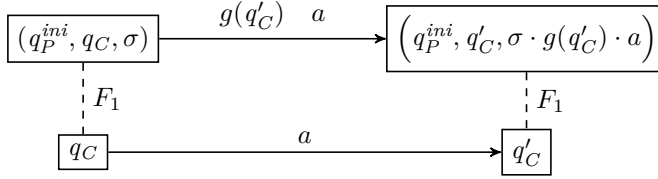
B Proofs for Section 6

► **Lemma 6.3.** (F_1, \emptyset) is a weak progressive forward simulation from C to $P \times C \times S_C$.

Proof. First of all, observe that $(q_C^{ini}, (q_P^{ini}, q_C^{ini}, \varepsilon)) \in F_1$. Now let $(q_C, (q_P, q_C, \sigma)) \in F_1$ and let $q_C \xrightarrow{a} q'_C$ be a step of C . There are two cases to consider.

Internal steps: $a \in \Sigma_C \setminus I \cup R$.

The diagram illustrates the simulation.



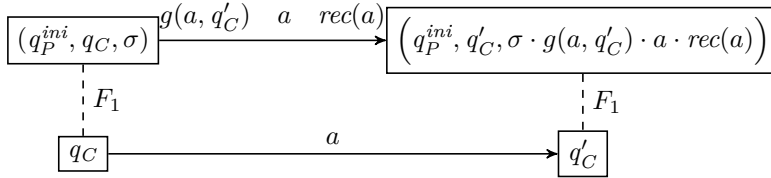
Since $last(\sigma) = q_C$, we get (by definition of S_C) that $g(q'_C) \in S_C(\sigma)$ and $a \in S_C(\sigma \cdot g(q'_C))$. Hence, the following transitions are possible:

$$(q_P^{ini}, q_C, \sigma) \xrightarrow{g(q'_C)} (q_P^{ini}, q_C, \sigma \cdot g(q'_C)) \xrightarrow{a} (q_P^{ini}, q'_C, \sigma \cdot g(q'_C) \cdot a)$$

We furthermore get $(q_C, (q_P^{ini}, q'_C, \sigma \cdot g(q'_C) \cdot a)) \in F_1$.

Invokes and returns: $a \in I \cup R$.

The diagram illustrates the simulation.



Since $last(\sigma) = q_C$, we get $g(a, q'_C) \in S_C(\sigma)$. By definition of S_C we hence get

$$(q_P^{ini}, q_C, \sigma) \xrightarrow{g(a, q'_C)} (q_a, q_C, \sigma \cdot g(a, q'_C)) \xrightarrow{a} (q_{rec(a)}, q'_C, \sigma \cdot g(a, q'_C) \cdot a) \xrightarrow{rec(a)} (q_P^{ini}, q'_C, \sigma \cdot g(a, q'_C) \cdot a \cdot rec(a))$$

We furthermore get $(q_C, (q_P^{ini}, q'_C, \sigma \cdot g(a, q'_C) \cdot a \cdot rec(a))) \in F_1$.

This is furthermore a weak progressive forward simulation as the matching steps are never empty (no triangular diagrams), so that we can take \gg_1 to be empty. ◀

► **Definition B.1.** Given an LTS L , and a subset $\Delta \subseteq \Sigma$ a state q is

■ reachable with $h \in \Delta^*$, written $\text{reach}_L(h, q)$ if $q = \text{last}(\xi)$ for an execution ξ that has history h .

■ minimally reachable, written if additionally, the execution ξ is shortest: its trace σ is either empty when $h = \epsilon$ (then the state is initial), or the last action of σ is the last of h .

We then say that an LTS is Δ -deterministic if the set of minimally reachable states for every history h consists of a single element which we write $\text{minstate}_L(h)$.

► **Lemma B.2.** The two LTSs $P \times C \times S_C$ and $P \times A \times S_A$ are Σ_P -deterministic.

Proof. We first prove that $P \times C \times S_C$ and $P \times A \times S_A$ are Γ_P -deterministic. To do so we show that given two finite traces σ_1 and σ_2 with $h = \sigma_1|_{\Gamma_P} = \sigma_2|_{\Gamma_P}$ the corresponding executions must be prefixes of each other by induction over the length of the shorter one. The initial states are the same since the initial state of deterministic systems is unique. Given that the executions agree up to step n , the next action is the same: either it is the single scheduled internal one, or the next action is the common one of the history. Since the steps are deterministic the next state is equal as well. It follows that there is only one minimally reachable state for every history, since in particular the final states of two minimal executions must agree, as their traces must be the same. For our specific program a minimal execution with history h also executes the same actions from $I \cup R$, since each such action e is followed by the corresponding record-action $\text{rec}(e)$, otherwise it would not be minimal. Therefore the two LTS are Σ_P -deterministic too. ◀

► **Lemma 6.4.** If C Δ -refines A and A is Δ -deterministic, then there exists a Δ -forward simulation.

Proof. Define the forward simulation F as

$$F = \{(q_C, q_A) \mid \exists h \in \Delta^*. \text{reach}_C(h, q_C) \wedge q_A = \text{minstate}_A(h)\}$$

where $\text{reach}_C(h, q_C)$ and $\text{minstate}_A(h)$ are from Def. B.1. The proof obligations of a forward simulation are satisfied: The initial concrete state is related to the initial abstract one by choosing $h = \epsilon$. Given $(q_C, q_A) \in F$, and $q_C \xrightarrow{a} q'_C$ there are two cases: If a is internal, then doing a stuttering step by choosing $q'_A = q_A$ is sufficient to show $(q'_C, q_A) \in F$. If $a \in \Delta$, then $(q_C, q_A) \in F$ implies that there is a history h and a minimal execution of A with this history and final state $q_A = \text{minstate}_A(h)$. Also ha is a history of C that has final state q'_C , so by refinement $h \cdot a$ is a history of A too. Therefore, there exists a minimal execution with this history $h \cdot a$ of A ending in some state q'_A . This state fits our correctness proof obligation: since the execution has a minimal prefix with history h (the prefix removes a and all internal actions after the last one of h), uniqueness of the minimal reachable state implies that it must pass through q_A . The trace σ of the remaining steps then has $a|_{\Delta} = \sigma|_{\Delta}$ and $q_A \xrightarrow{\sigma} q'_A$ as required, and $(q'_C, q'_A) \in F$ holds by definition. ◀

► **Lemma 6.5.** There exists a weak progressive Σ_P -forward simulation (F_2, \gg_2) between $P \times C \times S_C$ and $P \times A \times S_A$.

Proof. Since strong observational refinement implies Γ_P -refinement, and the abstract system is Σ_P -deterministic (Lemma B.2) Lemma 6.4 ensures that a Σ_P -forward simulation F exists. This forward simulation is weak progressive, since the concrete system never executes more

31:22 Weak Progressive Forward Simulation Is Necessary and Sufficient

than one internal action in a row ($S_C(\sigma' \cdot a)$ is external, when a is internal). Thus the well-founded order can be chosen to have $(q'_P, q'_C, \sigma') \gg (q_P, q_C, \sigma)$ iff $q_P = q'_P = q_P^{ini}$ and there is an internal action a with $q_C \xrightarrow{a} q'_C$ and $\sigma' = \sigma \cdot a$. \blacktriangleleft

► **Lemma 6.6.** (F_3, \gg_3) is a weak progressive forward $(I \cup R)$ -simulation between $P \times A \times S_A$ and A .

Proof. First we prove that F_3 is a forward simulation.

Stuttering steps The stuttering steps are $a \in \{g(q), g(e, q), div, rec(e)\}$. The proofs for each of these are trivial, but for completeness, we consider each of these in turn. We have the following transitions:

$$\begin{aligned} a = g(q). & \text{ We have } (q_P^{ini}, q_A, \sigma) \xrightarrow{g(q)} (q_P^{ini}, q_A, \sigma \cdot g(q)). \\ a = g(e, q). & \text{ We have } (q_P^{ini}, q_A, \sigma) \xrightarrow{g(e, q)} (q_e, q_A, \sigma \cdot g(e, q)). \\ a = div. & \text{ We have } (q_P^{ini}, q_A, \sigma) \xrightarrow{div} (q_{div}, q_A, \sigma). \\ a = rec(e). & \text{ We have } (q_{rec(e)}, q_A, \sigma) \xrightarrow{rec(e)} (q_P^{ini}, q_A, \sigma \cdot rec(e)). \end{aligned}$$

In each of these, if F_3 holds in the pre-state, it holds again in the post state since q_A is unchanged, and the abstract system, i.e., A does not take a step.

Non-stuttering steps All internal and external steps of A are non-stuttering. Since F_3 ensures that the states of A at the concrete and abstract states coincide, these can be trivially discharged too. In particular, the possible transitions are:

$$\begin{aligned} a = e \in (I \cup R). & \text{ We have } (q_e, q_A, \sigma) \xrightarrow{e} (q_{rec(e)}, q'_A, \sigma \cdot e). \\ a \in \Sigma_A \setminus (I \cup R). & \text{ We have } \\ & (q, q_A, \sigma) \xrightarrow{a} (q, q'_A, \sigma). \end{aligned}$$

In both cases, in A , we can take the corresponding transition $q_A \xrightarrow{a} q'_A$, preserving F_3 .

We now prove weak progressiveness of F_3 . Note that for each stuttering transition, except $g(q)$, the program state changes. We define the well-founded order to be the relation \gg_3 such that:

1. $(q_P^{ini}, _, _) \gg_3 (q_e, _, _)$
2. $(q_{rec(e)}, _, _) \gg_3 (q_P^{ini}, _, _)$
3. $(q_P^{ini}, _, _) \gg_3 (q_{div}, _, _)$

Note that from $(q_{div}, _, _)$, the only possible transition is an \xrightarrow{a} step, where $a \in \Sigma_A \setminus (I \cup R)$, which is non-stuttering. Similarly, from $(q_e, _, _)$, the only possible transition is \xrightarrow{e} , where $e \in I \cup R$. Thus, when we reach a state that is minimal wrt \gg_3 , no more stuttering is possible. Any transition from state $(q_{rec(e)}, _, _)$ is guaranteed to reduce w.r.t. \gg_3 , as are transitions corresponding to $g(e, q)$ and div from $(q_P^{ini}, _, _)$.

This leaves us with transitions corresponding to $g(q)$ from (q_P^{ini}, q_A, σ) , which may stutter infinitely often. We can show that such stuttering only exists if A contains a diverging run from q_A , i.e., div is enabled in $(q_P^{ini}, q_A, \sigma) \in Q_{P \times A \times S_A}$.

Suppose there exists an infinite run

$$(q_P^{ini}, q_A, \sigma) \xrightarrow{g(q_1)} (q_P^{ini}, q_A, \sigma \cdot g(q_1)) \xrightarrow{g(q_2)} (q_P^{ini}, q_A, \sigma \cdot g(q_1) \cdot g(q_2)) \xrightarrow{g(q_3)} \dots$$

By construction, $P \times A \times S_A$ is an “abstraction” of $P \times C \times S_C$ such that $T(P \times A, S_A)|_\Gamma = T(P \times C, S_C)|_\Gamma$, thus, $(\sigma|_\Gamma) \cdot g(q_1) \cdot g(q_2) \cdot g(q_3) \cdot \dots \in T(P \times C, S_C)|_\Gamma$. Thus, there exists a q_C such that $last(\sigma) = g(q_C)$ and

$$(q_P^{ini}, q_C, \sigma) \xrightarrow{g(q_1) \cdot \tau_1} (q_P^{ini}, q_1, \sigma \cdot g(q_1) \cdot \tau_1) \xrightarrow{g(q_2) \cdot \tau_2} (q_P^{ini}, q_2, \sigma \cdot g(q_1) \cdot \tau_1 \cdot g(q_2) \cdot \tau_2) \xrightarrow{g(q_3) \cdot \tau_3} \dots$$

where $\tau_k \in \Sigma_C \setminus (I \cup R)$ for all k . Note that the definition of S_C enforces a $\frac{g(q_k) \cdot \tau_k}{\rightarrow}$ transition for each $\frac{g(q_k)}{\rightarrow}$ transition in $P \times A \times S_A$. This execution, when restricted to the actions of C corresponds to a diverging run of C :

$$q_C \xrightarrow{\tau_1} q_1 \xrightarrow{\tau_2} q_2 \xrightarrow{\tau_3} \dots$$

Since this is an infinite run of internal actions, by definition, the action div must be offered by $P \times C \times S_C$, and enabled in (q_P^{ini}, q_C, σ) . Moreover, since $T(P \times A, S_A)|_\Gamma = T(P \times C, S_C)|_\Gamma$, div must also be possible in $P \times A \times S_A$. In particular, div must be enabled in (q_P^{ini}, q_A, σ) . Now, $P \times A \times S_A$ contains a run with final state σ . Therefore, $P \times A \times S_A$ also contains a run

$$(q_P^{ini}, q_A, \sigma) \xrightarrow{div} (q_{div}, q_A, \sigma) \xrightarrow{\tau'_1} (q_{div}, q'_1, \sigma) \xrightarrow{\tau'_2} (q_{div}, q'_2, \sigma) \xrightarrow{\tau'_3} \dots$$

where $\tau'_k \in \Sigma_A \setminus (I \cup R)$ for all k since $P \times A \times S_A$ can no longer schedule any further external actions after executing div , i.e., must schedule an internal action. Thus, we must have a diverging run in A as well. \blacktriangleleft

► **Theorem 6.7.** *Let (F_1, \gg_1) be a weak progressive Δ -forward simulation from C to B , and (F_2, \gg_2) one from B to A . Then there exists a weak progressive Δ -forward simulation (F, \gg) from C to A .*

Proof. We show that (F, \gg) is a weak progressive forward from C to A . Standard refinement results (e.g. Proposition 4.9 in [22]) imply, that refinement by forward simulation is transitive, i.e. $F_1 \circ F_2$ is a Δ -forward simulation. The definition of \gg also clearly implies that the order \gg decreases on a triangular diagram, when its abstract state has no diverging run. It remains to be shown that \gg is well-founded. An infinite descending chain $q_C^0 \gg q_C^1 \gg \dots$ leads to a contradiction as follows: by definition of \gg the states are the ones of a diverging run of C that starts with q_C^0 . Since F_1 is a forward simulation, there is a corresponding run of B with states q_B^0, q_B^1, \dots . For all k both $(q_C^k, q_B^k) \in F_1$ and $q_B^k \xrightarrow{\alpha^k} q_B^{k+1}$ hold, where each α^k is a sequence of internal actions. Some α^k may be empty, so states may occur several times. It is, however, not possible that there is a final state q_B^k such that $q_B^k = q_B^{k+1} = \dots$, since then $q_C^k \gg_1 q_C^{k+1} \gg_1 \dots$ would be implied, contradicting well-foundedness of \gg_1 . Note that q_B^k cannot start a diverging run in this case, otherwise $q_C^k \gg q_C^{k+1}$ would not hold by definition. Therefore, the states q_B^0, q_B^1, \dots are some states of an infinite diverging run too.

By applying the same argument for the upper simulation a sequence q_A^0, q_A^1, \dots of states of A can be found, such that for all k $(q_B^k, q_A^k) \in F_2$ and $q_A^k \xrightarrow{\beta^k} q_A^{k+1}$ holds. Again the β^k are sequences of internal actions, and the existence of a final state with $q_A^k = q_A^{k+1} = \dots$ would contradict well-foundedness of \gg_2 . Therefore the construction results in a diverging run from q_A^0 , contradicting the definition of $q_C^0 \gg q_C^1$ which required that no corresponding abstract state q_A^0 with a diverging run exists. \blacktriangleleft

Strategies for MDP Bisimilarity Equivalence and Inequivalence

Stefan Kiefer  

Department of Computer Science, University of Oxford, UK

Qiyi Tang  

Department of Computer Science, University of Liverpool, UK

Abstract

A labelled Markov decision process (MDP) is a labelled Markov chain with nondeterminism; i.e., together with a strategy a labelled MDP induces a labelled Markov chain. Motivated by applications to the verification of probabilistic noninterference in security, we study problems whether there exist strategies such that the labelled MDPs become bisimilarity equivalent/inequivalent. We show that the equivalence problem is decidable; in fact, it is EXPTIME-complete and becomes NP-complete if one of the MDPs is a Markov chain. Concerning the inequivalence problem, we show that (1) it is decidable in polynomial time; (2) if there are strategies for inequivalence then there are memoryless strategies for inequivalence; (3) such memoryless strategies can be computed in polynomial time.

2012 ACM Subject Classification Theory of computation → Program verification; Theory of computation → Models of computation; Mathematics of computing → Probability and statistics

Keywords and phrases Markov decision processes, Markov chains

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.32

Funding This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement 956123 (FOCETA).



Acknowledgements We thank the anonymous reviewers of this paper for their constructive feedback.

1 Introduction

Given a model of computation (e.g., finite automata), and two instances of it, are they semantically equivalent (e.g., do the automata accept the same language)? Such *equivalence* problems can be viewed as a fundamental question for almost any model of computation. As such, they permeate computer science, in particular, theoretical computer science.

In *labelled Markov chains (LMCs)*, which are Markov chains whose states (or, equivalently, transitions) are labelled with an observable letter, there are two natural and very well-studied versions of equivalence, namely *trace (or language) equivalence* and *probabilistic bisimilarity*.

The *trace equivalence* problem has a long history, going back to Schützenberger [18] and Paz [15] who studied *weighted* and *probabilistic* automata, respectively. Those models generalize LMCs, but the respective equivalence problems are essentially the same. For LMCs, trace equivalence asks if the same label sequences have the same probabilities in the two LMCs. It can be extracted from [18] that equivalence is decidable in polynomial time, using a technique based on linear algebra; see also [21, 5].

Probabilistic bisimilarity is an equivalence that was introduced by Larsen and Skou [14]. It is finer than trace equivalence, i.e., probabilistic bisimilarity implies trace equivalence. A similar notion for Markov chains, called *lumpability*, can be traced back at least to the classical text by Kemeny and Snell [10]. Probabilistic bisimilarity can also be computed in polynomial time [1, 4, 22]. Indeed, in practice, computing the bisimilarity quotient is fast and has become a backbone for highly efficient tools for probabilistic verification such as PRISM [13] and STORM [8].



© Stefan Kiefer and Qiyi Tang;

licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 32; pp. 32:1–32:22

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper, we study probabilistic bisimilarity in (*labelled*) *Markov decision processes* (*MDPs*), which are LMCs plus nondeterminism; i.e., each state may have several *actions* (or “moves”) one of which is chosen by a controller, potentially randomly. An MDP and a controller *strategy* together induce an LMC (potentially with infinite state space, depending on the complexity of the strategy). The nondeterminism in MDPs gives rise to a spectrum of equivalence queries: one may ask about the existence of strategies for two given MDPs such that the induced LMCs become trace/bisimilarity equivalent, or such that they become trace/bisimilarity *inequivalent*. Another potential dimension of this spectrum is whether to consider general strategies or more restricted ones, such as memoryless or even memoryless deterministic (MD) ones.

Much of this spectrum has been covered in previous work. It was shown in [6] that whether there exist (general) strategies such that two given MDPs become trace equivalent is undecidable. In fact, even whether there exists a strategy such that a given MDP becomes trace equivalent to a given LMC is undecidable [6, Theorem 3.1]. This points to a fundamental difficulty when dealing with a general strategy in an MDP: since the strategy may use unrestricted memory, the induced LMC can have an infinite (countable) state space, even when the MDP is finite. For this reason it is not a priori clear whether the *bisimilarity equivalence* problem, namely whether there exist general strategies such that two given MDPs become bisimilar, is even decidable.

The problem was “dodged” in [11], where trace and bisimilarity (in)equivalence problems were covered, but under the explicit assumption of *memoryless* strategies. There are good reasons to consider memoryless strategies, particularly their naturalness and simplicity in implementations, and their connection to *interval Markov chains* (see, e.g., [9, 3]) and *parametric MDPs* (see, e.g., [7, 23]). It was shown in [11, Theorem 19] that the bisimilarity equivalence problem is NP-complete for memoryless strategies.

It remained open in [11] whether the bisimilarity equivalence problem is decidable for general strategies, which would be in contrast to the undecidability of the corresponding trace equivalence problem [6]. There are also good reasons to consider general unrestricted strategies, primarily their naturalness (in their definition for MDPs) and their generality. The latter is important particularly for security applications, see below, where an attacker should be conservatively assumed to be powerful to employ an arbitrary strategy.

As one of our main results, we show that the bisimilarity problem for general strategies is decidable, in fact, EXPTIME-complete. This high computational complexity means that in order to induce two bisimilar LMCs in the two given MDPs it is generally necessary to employ complex strategies, inducing complex behaviours. We also show that the computational complexity reduces to NP if one of the MDPs is already an LMC.

The challenges of the corresponding bisimilarity *inequivalence* problems are somewhat analogous, but the results are opposite. It was shown in [11, Corollary 13] that whether there are *memoryless* strategies in two given MDPs that induce *nonbisimilar* LMCs can be decided in polynomial time and that such memoryless strategies, if they exist, can be computed in polynomial time. As our second main result we show that this extends in an almost ideal way (although the proof is nontrivial): whenever there are general strategies for inequivalence, there are memoryless ones (and thus can be computed as in [11]). This means that, very much unlike for equivalence, memoryless strategies suffice for inequivalence, inducing relatively simple inequivalent LMCs.

Complementing the theoretical nature of these questions, let us mention an application from the field of security. *Noninterference* refers to an information-flow property of a program, stipulating that information about *high* data (i.e., data with high confidentiality) may not

leak to *low* (i.e., observable) data, or, quoting [17], “that a program is secure whenever varying the initial values of high variables cannot change the low-observable (observable by the attacker) behaviour of the program”. It was proposed in [17] to reason about *probabilistic* noninterference in probabilistic multi-threaded programs by proving probabilistic bisimilarity; see also [20, 16]. More precisely, probabilistic noninterference is established if it can be shown that any two states that differ only in high data are probabilistic bisimilar, as then an attacker who only observes the low part of a state learns nothing about the high part. The observable behaviour of a multi-threaded program depends strongly on the *scheduler*, which raises the question whether bisimilarity holds under some or even under all schedulers [17]. A scheduler in this context amounts to a strategy in the corresponding MDP.

The rest of the paper is organized as follows. We give preliminaries in Section 2. In Sections 3 and 4 we prove our results on bisimilarity equivalence and inequivalence, respectively. We conclude in Section 5. Missing proofs can be found in an appendix.

2 Preliminaries

We write \mathbb{N} for the set of nonnegative integers. Let S be a finite set. We denote by $\text{Distr}(S)$ the set of probability distributions on S . For a distribution $\mu \in \text{Distr}(S)$ we write $\text{support}(\mu) = \{s \in S \mid \mu(s) > 0\}$ for its support.

A *labelled Markov chain* (LMC) is a quadruple $\langle S, L, \tau, \ell \rangle$ consisting of a nonempty set S of states¹, a nonempty finite set L of labels, a transition function $\tau : S \rightarrow \text{Distr}(S)$, and a labelling function $\ell : S \rightarrow L$.

We denote by $\tau(s)(t)$ the transition probability from s to t . Similarly, we denote by $\tau(s)(E) = \sum_{t \in E} \tau(s)(t)$ the transition probability from s to $E \subseteq S$.

An equivalence relation $R \subseteq S \times S$ is a *probabilistic bisimulation* if for all $(s, t) \in R$, $\ell(s) = \ell(t)$ and $\tau(s)(E) = \tau(t)(E)$ for each R -equivalence class E . *Probabilistic bisimilarity*, denoted by \sim , is the largest probabilistic bisimulation.

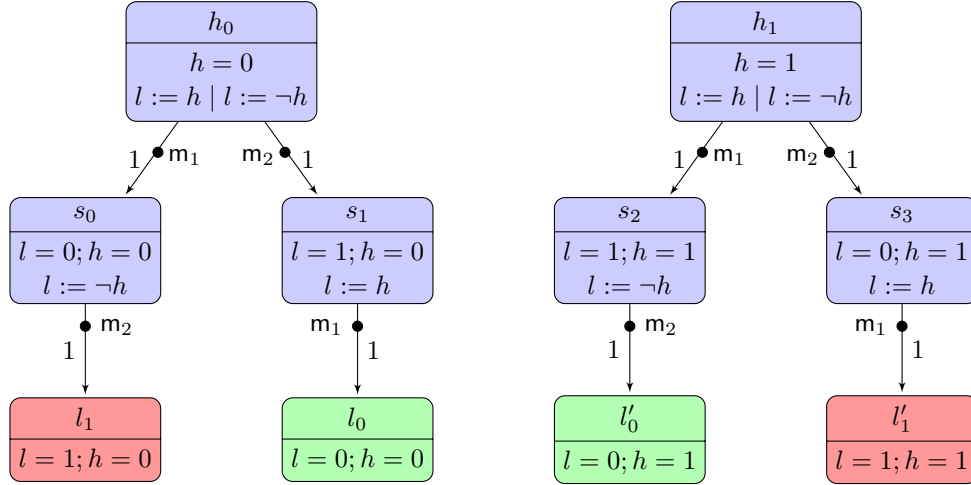
A (*labelled*) *Markov decision process* (MDP) is a tuple $\langle S, \text{Act}, L, \varphi, \ell \rangle$ consisting of a finite set S of states, a finite set Act of actions, a finite set L of labels, a partial function $\varphi : S \times \text{Act} \rightarrow \text{Distr}(S)$ denoting the probabilistic transition, and a labelling function $\ell : S \rightarrow L$. The set of available actions in a state s is $\text{Act}(s) = \{m \in \text{Act} \mid \varphi(s, m) \text{ is defined}\}$.

A path is a sequence $\rho = s_0 m_1 s_1 \cdots m_n s_n$ such that $\varphi(s_i, m_{i+1})$ is defined and $\varphi(s_i, m_{i+1})(s_{i+1}) > 0$ for all $0 \leq i < n$. The last state of ρ is $\text{last}(\rho) = s_n$. Let $\text{Paths}(\mathcal{D})$ denote the set of paths in \mathcal{D} .

A (general) strategy for an MDP is a function $\alpha : \text{Paths}(\mathcal{D}) \rightarrow \text{Distr}(\text{Act})$ that given a path ρ , returns a probability distribution on the available actions at the last state of ρ , $\text{last}(\rho)$. A memoryless strategy depends only on $\text{last}(\rho)$; so we can identify a memoryless strategy with a function $\alpha : S \rightarrow \text{Distr}(\text{Act})$ that given a state s , returns a probability distribution on the available actions at that state.

Given a general strategy α for \mathcal{D} , an LMC $\mathcal{D}(\alpha) = \langle \mathcal{P}, L, \tau, \ell' \rangle$ is induced, where $\mathcal{P} \subseteq \text{Paths}(\mathcal{D})$. For $\rho \in \mathcal{P}$, we have $\tau(\rho)(\rho m t) = \alpha(\rho)(m) \varphi(s, m)(t)$ and $\ell'(\rho) = \ell(s)$ where $s = \text{last}(\rho)$ and $m \in \text{Act}(s)$.

¹ We mainly consider LMCs with finitely many states unless otherwise stated.



■ **Figure 1** The program $l := h \mid l := \neg h$ as an MDP.

3 Bisimilarity Problems

In this section we consider the bisimilarity problem which, given an MDP and two (initial) states, asks whether there is a general strategy such that the two states are probabilistic bisimilar in the LMC induced by the general strategy.

► **Example 1.** Borrowing an example from [17, Section 4], consider the following simple program composed of two threads, involving a *high* boolean variable h (high confidentiality) and a *low* boolean variable l (observable):

$$l := h \mid l := \neg h$$

The vertical bar $|$ separates two threads. The order in which the threads are executed is determined by a scheduler. We assume that the variable l becomes visible upon program termination. Its value will be h or $\neg h$, depending on whether the left or the right thread is executed *last*. The program can be viewed as an MDP as shown in Figure 1. The top part of each state indicates its name; the bottom part indicates the values of the variables, as well as the part of the program that is yet to be executed. Different colours indicate different state labels. The two top states h_0, h_1 differ in their value of h , but this difference is not observable; thus, h_0, h_1 have the same label. The actions m_1, m_2 available in h_0, h_1 correspond to the two scheduling options of the program: m_1 means that the thread $l := h$ is scheduled next and m_2 means that the thread $l := \neg h$ is scheduled next. The strategy that in h_0, h_1 picks one of the two actions uniformly at random induces an LMC in which h_0 and h_1 are probabilistic bisimilar. In fact, the bisimilarity equivalence classes under this strategy are $\{h_0, h_1\}$, $\{h_0 m_1 s_0, h_1 m_2 s_3\}$, $\{h_0 m_2 s_1, h_1 m_1 s_2\}$, $\{h_0 m_1 s_0 m_2 l_1, h_1 m_2 s_3 m_1 l'_1\}$, $\{h_0 m_2 s_1 m_1 l_0, h_1 m_1 s_2 m_2 l'_0\}$. Since h_0, h_1 are probabilistic bisimilar, this memoryless strategy prevents a leak of the value of h : an attacker who observes l in the end learns nothing about h .

In this section we show that the bisimilarity problem is EXPTIME-complete. We prove the upper and lower bound in Sections 3.1 and 3.2, respectively.

3.1 Membership in EXPTIME

To prove that the bisimilarity problem can be decided in EXPTIME, we define and analyse an auxiliary game, the *attacker-defender game*. It is a two-person (zero-sum, non-stochastic, turn-based) game, and is defined from an MDP $\mathcal{D} = \langle S, Act, L, \varphi, \ell \rangle$ and a set of states $E_1 \subseteq S$. The two players are called *Defender* and *Attacker*. The intuition (which we will prove in Proposition 3 below) is that Defender can win the game if and only if there is a general strategy α for \mathcal{D} such that in the LMC induced by α all states in E_1 are probabilistic bisimilar. The attacker-defender game proceeds in rounds $1, 2, \dots$. At the beginning of round 1 the game is in state E_1 . Suppose at the beginning of round i the game is in state $E_i \subseteq S$. Then in round i Defender chooses (and announces publicly)

- $S' \subseteq 2^S$ such that for any $E \in S'$ and any $s, t \in E$ we have $\ell(s) = \ell(t)$ (intuitively, Defender claims for each $E' \in S'$ that there is a general strategy such that all states in E' are probabilistic bisimilar);
- a distribution $\nu \in \text{Distr}(S')$;
- for each $s \in E_i$ a memoryless strategy (possibly randomised) $\alpha^s \in \text{Distr}(Act(s))$; and
- a function $f : E_i \times Act \times S \rightarrow S'$ with $t \in f(s, m, t)$ for all $(s, m, t) \in E_i \times Act \times S$ such that for all $s \in E_i$ and all $E' \in S'$ we have

$$\nu(E') = \sum_{m \in Act(s)} \sum_{t \in S \text{ s.t. } f(s, m, t) = E'} \alpha^s(m) \varphi(s, m)(t).$$

If objects with the properties required above do not exist, Defender loses and Attacker wins. Otherwise, to complete round i , Attacker chooses from S' a set E_{i+1} , which is the state of the game at the beginning of round $i+1$. If the game goes on forever, Defender wins and Attacker loses.

Memoryless winning strategies suffice for Defender:

► **Lemma 2.** *Given an MDP $\mathcal{D} = \langle S, Act, L, \varphi, \ell \rangle$ and a set $E_1 \subseteq S$, if Defender has a winning strategy for the attacker-defender game, Defender has a memoryless winning strategy, i.e., a winning strategy that depends only on the current state of the game.*

Proof. Recall that the states in the attacker-defender game are sets $E \subseteq S$. Let $S_w \subseteq 2^S$ denote the set of those $E \subseteq S$ such that starting from E Defender can win the attacker-defender game. We define a memoryless strategy, σ' , for Defender such that starting from any $E \in S_w$, all possible successor states are also in S_w . Therefore, using σ' , starting from any $E \in S_w$, the game remains in S_w indefinitely; i.e., σ' is a winning strategy for Defender.

Let $E \in S_w$. Then Defender has a (not necessarily memoryless) winning strategy σ starting from E . According to the rules of the game, in the first round σ chooses various objects, including $S' \subseteq 2^S$. Since σ is winning for Defender, Defender has a (not necessarily memoryless) winning strategy for all $E' \in S'$, i.e., $S' \subseteq S_w$. The memoryless strategy σ' is defined so that in E it makes the same choices that σ makes in E in the first round. All possible successor states are in S_w , as required. ◀

The following proposition establishes the connection between the bisimilarity problem and the attacker-defender game:

► **Proposition 3.** *Given an MDP $\mathcal{D} = \langle S, Act, L, \varphi, \ell \rangle$ and a set $E_1 \subseteq S$, Defender has a winning strategy for the attacker-defender game if and only if there exists a general strategy α for \mathcal{D} such that in the LMC induced by α all states in E_1 are probabilistic bisimilar.*

Proof. (\implies) Assume Defender can win the attacker-defender game which starts in E_1 . By Lemma 2, Defender has a memoryless strategy. Using this memoryless strategy (the objects chosen by Defender), a general strategy α for the MDP \mathcal{D} can be constructed.

Let \mathcal{P}_i be a set of paths in \mathcal{D} such that for any $\rho \in \mathcal{P}_i$ we have that ρ begins with a state $s \in E_1$ and the number of states in ρ is i . Let $\mathcal{P} = \bigcup_{i \geq 1} \mathcal{P}_i$. A path in \mathcal{P} can be mapped to a possible state of the attacker-defender game. Let us define \mathcal{P} and such a mapping W inductively on i as follows:

- Base case $i = 1$. We have $\mathcal{P}_1 := E_1$. For any state $s \in E_1$, it is mapped to the start state of the game E_1 , that is, $W(s) := E_1$ for $s \in E_1$.
- Inductive case when $i > 1$. For a path $\rho' = \rho mt$, it belongs to \mathcal{P}_i if and only if $\rho \in \mathcal{P}_{i-1}$ and $\rho' \in \text{Paths}(\mathcal{D})$. Next, we define $W(\rho mt)$ for a path $\rho mt \in \mathcal{P}_i$. Let $s = \text{last}(\rho)$ and $E_{i-1} = W(\rho)$. If E_{i-1} is the state at the beginning of some round of the attacker-defender game, Defender chooses a set $S' \subseteq 2^S$ and a function $f : E_{i-1} \times \text{Act} \times S \rightarrow S'$ with $t \in f(s, m, t)$ for all $(s, m, t) \in E_{i-1} \times \text{Act} \times S$. We define $W(\rho mt) := f(s, m, t)$. Since $f(s, m, t) \in S'$, $W(\rho mt)$ may be chosen by Attacker as the new state.

Let $i \geq 1$. A path $\rho \in \mathcal{P}_i$ is mapped to $W(\rho)$, a possible state at the beginning of round i of the attacker-defender game. It can be shown by induction that Defender has a winning strategy for $W(\rho)$. We assume that Defender chooses memoryless strategies $\alpha^s \in \text{Distr}(\text{Act}(s))$ for $s \in W(\rho)$. We define the general strategy $\alpha : \mathcal{P} \rightarrow \text{Distr}(\text{Act})$ as $\alpha(\rho) := \alpha^s$ where $s = \text{last}(\rho)$. We show in the appendix that all states in E_1 are probabilistic bisimilar in the LMC induced by α .

(\impliedby) We define the notion of an equalisable set. A set $E \subseteq S$ is equalisable if and only if there is a general strategy such that all states in E are probabilistic bisimilar in the induced LMC.

Let $E \subseteq S$ be an arbitrary equalisable set. We define a strategy for Defender when the attacker-defender game is in state E at the beginning of some round.

By definition, there is a general strategy, say α_E , such that all states in E are probabilistic bisimilar in the LMC induced by α_E . In the induced LMC, the successor states of any state of E can be partitioned into probabilistic bisimulation classes, say B_1, \dots, B_k where k is a positive integer. The transition probability distributions v' over B_1, \dots, B_k from any state in E are the same, that is, $v'(B_i) = \sum_{smt \in B_i} \alpha_E(s)(m)\varphi(s, m)(t)$ for any $1 \leq i \leq k$ and $s \in E$.

This probability distribution will be the one chosen by Defender. In the LMC induced by α_E , a state $\rho \in B_i$ where $1 \leq i \leq k$, a successor state of $s \in E$, is of the form smt where $m \in \text{support}(\alpha_E(s))$ and $t \in \text{support}(\varphi(s, m))$. We define $E'_i = \{t \mid s \in E \wedge smt \in B_i\}$ for $1 \leq i \leq k$. We are ready to define the objects chosen by Defender when the game is in state E at the beginning of some round:

- $S' = \{E'_1, \dots, E'_k\}$;
- A probability distribution v over S' where $v(E'_i) = v'(B_i)$ for $1 \leq i \leq k$;
- for any $s \in E$ a memoryless strategy $\alpha^s = \alpha_E(s)$; and
- a function $f : E \times \text{Act} \times S \rightarrow S'$ such that $f(s, m, t) = E'_i$ for any $s \in E$ and $smt \in B_i$.

We verify in the appendix that the objects chosen by Defender satisfy the required properties. If the game starts with an equalisable set and all the future game states are equalisable sets, Defender can always choose objects with required properties and hence win the game. We prove in the appendix that all sets in S' are equalisable.

As we assume there is a general strategy such that all states in E_1 are probabilistic bisimilar in the induced LMC, E_1 by definition is an equalisable set. This completes the proof. \blacktriangleleft

Now we can prove membership in EXPTIME.

► **Lemma 4.** *The bisimilarity problem is in EXPTIME.*

Proof. As EXPTIME equals APSPACE, it suffices to construct a PSPACE-bounded alternating Turing machine M that accepts the bisimilarity problem. By the definition of alternating Turing machines, the set of control states of M is partitioned into existential and universal states. There is an existential (respectively, universal) player who controls the existential (respectively, universal) states, and the player who controls the state of the current configuration chooses a successor configuration that is consistent with the transition relation of M . The goal of the existential player is to drive the computation into an accepting configuration (defined by a special control state), and the goal of the universal player is to prevent that from happening. If the existential player has a winning strategy, M is said to accept the input; otherwise M rejects the input.

In our case, the input of M is an MDP $\mathcal{D} = \langle S, Act, L, \varphi, \ell \rangle$ and two states $s, t \in S$. We need to construct M so that the existential player has a winning strategy if and only if there exists a general strategy α for \mathcal{D} such that s and t are probabilistic bisimilar in the induced LMC $\mathcal{D}(\alpha)$. Using Proposition 3, it suffices to construct M so that the existential player has a winning strategy if and only if Defender has a winning strategy in the attacker-defender game defined by \mathcal{D} and $E_1 := \{s, t\}$.

We construct M so that it implements the attacker-defender game: the existential (respectively, universal) player in M takes the role of Defender (respectively, Attacker) in the attacker-defender game. We have to ensure that M uses only polynomial space. The state of the attacker-defender game at the beginning of each round consists of a set $E_i \subseteq S$, which can clearly be stored in polynomial space. In each round, Defender (i.e., the existential player) needs to choose a set $S' \subseteq 2^S$, a distribution v on S' , memoryless strategies α^s for $s \in E_i$, and a function f . The equations that v and α^s are required to satisfy imply that Defender can restrict herself to choosing the set S' as the image of the function f ; then S' has at most polynomially many sets of states. Further, Defender can restrict herself to choosing v and α^s such that the numbers are fractions of integers with polynomially many bits (in particular, the numbers are rational). Indeed, given S' and f , there is a linear program of polynomial size whose solutions are exactly those v and α^s that satisfy, for all $s \in E_i$ and all $E' \in S'$,

$$v(E') = \sum_{\mathbf{m} \in Act(s)} \sum_{t \in S \text{ s.t. } f(s, \mathbf{m}, t) = E'} \alpha^s(\mathbf{m}) \varphi(s, \mathbf{m})(t).$$

So if there exist any v and α^s satisfying these equations, then there also exist rational ones with polynomially many bits.

Finally, we have to ensure that M enters an accepting configuration when Defender can win the attacker-defender game (recall that Defender wins the attacker-defender game if and only if it goes on forever). Using a counter on the tape, we make M enter an accepting configuration once $2^{|S|}$ rounds of the attacker-defender game have been played without Attacker having won. This is justified as follows. If Attacker has a winning strategy for the attacker-defender game, Attacker also has a winning strategy that guarantees that every set $E_i \subseteq S$ appears at most once as the state of the game at the beginning of a round. It follows that if Attacker can win, Attacker can also win in at most $2^{|S|}$ rounds. ◀

3.2 EXPTIME-Hardness

Recall that in the previous section an intermediate technical notion (attacker-defender games) was useful to derive the EXPTIME upper bound in the previous section. In this section we show that the bisimilarity problem is EXPTIME-hard. For this lower bound we leverage another non-stochastic intermediate tool, namely intersection emptiness of deterministic tree automata. Let us introduce the required definitions.

A *ranked* alphabet Σ is a finite set of symbols such that each symbol $a \in \Sigma$ is associated with a rank, $\text{rank}(a) \in \mathbb{N} \setminus \{0\}$. A *tree* over Σ is an ordered tree in which each node is labelled with a symbol $a \in \Sigma$ (we call such a node an a -node) and every a -node has $\text{rank}(a)$ children. Since we exclude symbols of rank 0, a tree over Σ is necessarily infinite. A *deterministic top-down tree automaton (DTTA)* is a quadruple $\mathcal{A} = (Q, \Sigma, \delta, q_0)$ where Q is a finite set of states, Σ is a ranked alphabet, $\delta : Q \times \Sigma \rightarrow Q^*$ is a *partial* transition function with $|\delta(q, a)| = \text{rank}(a)$ for all q, a for which $\delta(q, a)$ is defined, and $q_0 \in Q$ is the initial state. A *run* of DTTA \mathcal{A} on tree t is a labelling of the nodes of t such that the root is labelled with q_0 and for every a -node ($a \in \Sigma$), if it is labelled with $q \in Q$, then its children, read from left to right, are labelled with $\delta(q, a)$. Note that a DTTA has at most one run on any tree. Write $L(\mathcal{A})$ for the set of trees on which \mathcal{A} has a run. Given DTTAs $\mathcal{A}_1, \dots, \mathcal{A}_k$ over the same ranked alphabet, the *intersection nonemptiness* problem asks whether $\bigcap_{i=1}^k L(\mathcal{A}_i) \neq \emptyset$, i.e., whether there is a tree on which every DTTA \mathcal{A}_i has a run. A version of this problem, for finite trees, was proved EXPTIME-hard by Seidl [19]. The version of this problem for DFAs and finite words is a well-known PSPACE-complete problem [12]. By adapting these proofs we show:

► **Lemma 5.** *The intersection nonemptiness problem is EXPTIME-hard.*

We use this result to prove the following lemma.

► **Lemma 6.** *The bisimilarity problem is EXPTIME-hard.*

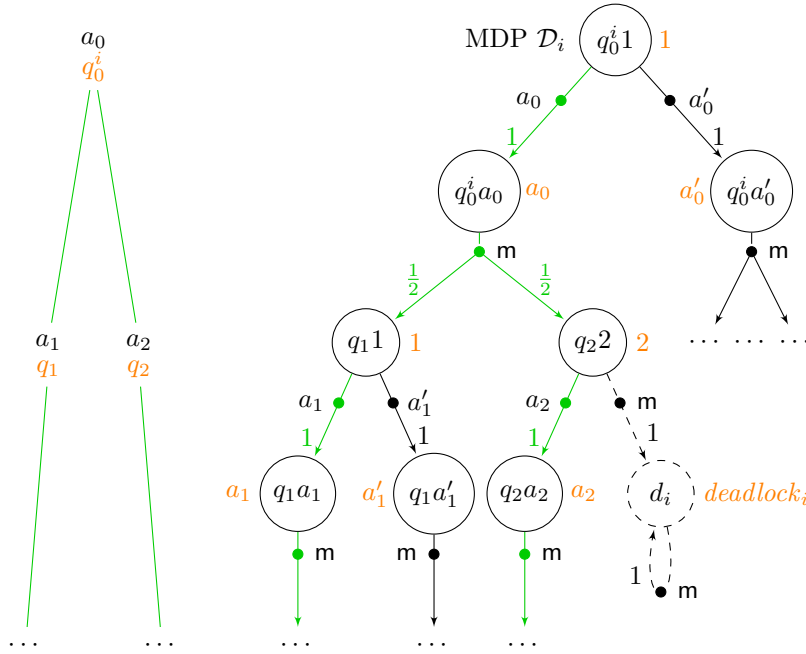
Proof. The reduction is from the intersection nonemptiness problem of DTTAs, which by Lemma 5 is EXPTIME-hard.

Given DTTAs $\mathcal{A}_1, \dots, \mathcal{A}_k$ over the same alphabet, we construct k MDPs $\mathcal{D}_1, \dots, \mathcal{D}_k$ and show that $\bigcap_{i=1}^k L(\mathcal{A}_i) \neq \emptyset \iff$ there is a general strategy for each MDP such that the k initial states in the induced LMCs are probabilistic bisimilar. Later we show how to replace the k MDPs by two MDPs, proving the statement of the lemma.

Let r be the maximum rank of symbols in the DTTAs. Let $L_{\text{order}} = \{1, \dots, r\}$ be a set of integer labels which is disjoint from the set Σ . Let m be an action different from any symbol in Σ . For each DTTA $\mathcal{A}_i = (Q_i, \Sigma, \delta_i, q_0^i)$ where $1 \leq i \leq k$, we construct an MDP $\mathcal{D}_i = \langle S_i, \text{Act}, L_i, \varphi_i, \ell_i \rangle$ as follows:

- $S_i := \{qj \mid q \in Q_i \text{ and } j \in \mathbb{N} \text{ and } 1 \leq j \leq r\} \cup \{qa \mid q \in Q_i \text{ and } a \in \Sigma \text{ and } \delta_i(q, a) \text{ is defined}\} \cup \{d_i\}$ where d_i is a special sink state only available in \mathcal{D}_i ;
- $\text{Act} := \Sigma \cup \{m\}$;
- $L_i := \Sigma \cup L_{\text{order}} \cup \{\text{deadlock}_i\}$ where deadlock_i is a special symbol only available in \mathcal{D}_i ;
- $\varphi_i(qj, a) := \{qa \mapsto 1\}$ for all $q \in Q_i, a \in \Sigma$ such that $\delta_i(q, a)$ is defined and all $1 \leq j \leq r$;
 $\varphi_i(qa, m) = \{q'j \mapsto \frac{1}{\text{rank}(a)} \mid q' \text{ is the } j\text{th symbol in } \delta(q, a)\}$ for all $q \in Q_i, a \in \Sigma$ such that $\delta_i(q, a)$ is defined; $\varphi_i(qj, m) := \{d_i \mapsto 1\}$ for all $1 \leq j \leq r$ and all $q \in Q_i$ such that $\delta_i(q, a)$ is not defined for any $a \in \Sigma$; since d_i is a sink state, we also have $\varphi_i(d_i, m) := \{d_i \mapsto 1\}$;
- $\ell_i(qj) = j$ for all $q \in Q_i$ and $1 \leq j \leq r$; $\ell_i(qa) = a$ for all $q \in Q_i$ and $a \in \Sigma$ such that $\delta_i(q, a)$ is defined; $\ell_i(d_i) = \text{deadlock}_i$.

The initial state of \mathcal{D}_i is $q_0^i 1$. Each state q of Q_i corresponds to a set of states qj in \mathcal{D}_i where the number j represents that q is the j th child. Such a state qj is assigned the label j . For each $q \in Q_i$ and $a \in \Sigma$ such that $\delta_i(q, a)$ is defined, we also have a state qa in \mathcal{D}_i . Such a state qa is assigned the label a . There is a special sink state d_i for each MDP \mathcal{D}_i . The set of actions is the same for all MDPs while each MDP \mathcal{D}_i has a special label deadlock_i which is used to label the sink state d_i . Since the only states in the MDPs that may have multiple actions are those qj states where q is a state in the automata and j is a number, we only need to specify the general strategy upon reaching those states.



■ **Figure 2** Consider a DTTA \mathcal{A}_i with $\text{rank}(a_0) = 2$, $\delta(q_0^i, a_0) = q_1 q_2$ and $\{(q_0^i, a_0'), (q_1, a_1')\} \subseteq \text{support}(\delta)$. On the right, there is the MDP \mathcal{D}_i corresponding to the DTTA \mathcal{A}_i . If $\delta(q_2, a_2)$ is defined in the DTTA \mathcal{A}_i , the state $q_2 2$ of \mathcal{D}_i has a single action a_2 taking it to the state $q_2 a_2$. Otherwise, if $\delta(q_2, a_2)$ is undefined in the DTTA \mathcal{A}_i , the state $q_2 2$ of \mathcal{D}_i has a single action m taking it to the deadlock state d_i (see the dashed transitions). The labels of the states of \mathcal{D}_i are written next to the states in orange. The left shows a part of a run of the DTTA \mathcal{A}_i on an ordered tree over Σ , that is, a labelling (in orange) of $q \in Q_i$ on the nodes of the tree. This run corresponds to a deterministic general strategy (highlighted in green) of the MDP \mathcal{D}_i on the right.

We show in the appendix that $\bigcap_{i=1}^k L(\mathcal{A}_i) \neq \emptyset$ if and only if there is a general strategy α_i for each MDP \mathcal{D}_i such that the k initial states in the induced LMCs are probabilistic bisimilar. See Figure 2 for an illustration.

It remains to replace the k MDPs $\mathcal{D}_1, \dots, \mathcal{D}_k$ by two MDPs. Specifically, we construct $\mathcal{D}'_1 = \langle S'_1, \text{Act}, L_1, \varphi'_1, \ell'_1 \rangle$ and $\mathcal{D}'_2 = \langle S'_2, \text{Act}, L'_2, \varphi'_2, \ell'_2 \rangle$ so that the following property holds: there is a general strategy $\mathcal{D}_1, \dots, \mathcal{D}_k$ respectively such that the k initial states in the induced LMCs are probabilistic bisimilar \iff there is a general strategy for \mathcal{D}'_1 and \mathcal{D}'_2 respectively such that the two initial states in the induced LMCs are probabilistic bisimilar.

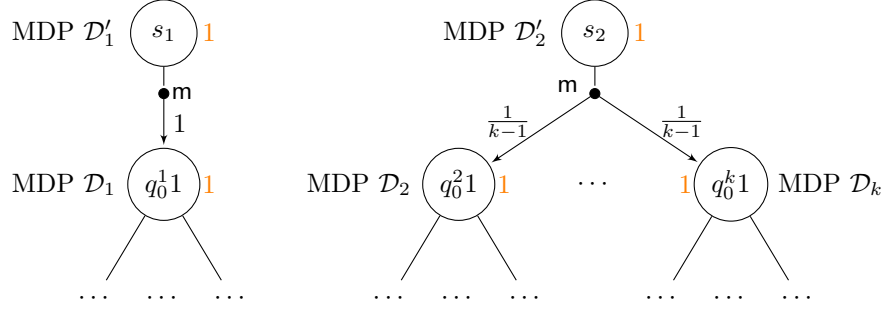
We distinguish the two cases: $k \geq 2$ and $k = 1$. When $k = 1$, for any general strategy, the initial state in the induced LMC is trivially probabilistic bisimilar with itself.

If $k \geq 2$, as shown in Figure 3, the initial state s_1 of the MDP \mathcal{D}'_1 has a single action m taking it to the initial state $q_0^1 1$ of \mathcal{D}_1 with probability one, while the initial state s_2 of the MDP \mathcal{D}'_2 has a single action m taking it to the initial states of $\mathcal{D}_2, \dots, \mathcal{D}_k$ with equal probabilities, that is,

$$\blacksquare \quad S'_1 = S_1 \dot{\cup} \{s_1\}, \quad \varphi'_1(s, a) = \begin{cases} \varphi_1(s, a) & \text{if } (s, a) \in \text{support}(\varphi_1) \\ \{q_0^1 1 \mapsto 1\} & \text{if } s = s_1 \text{ and } a = m \end{cases}$$

$$\text{and } \ell'_1(s) = \begin{cases} \ell_1(s) & \text{if } s \in S_1 \\ 1 & \text{if } s = s_1; \end{cases}$$

$$\blacksquare \quad S'_2 = \dot{\bigcup}_{i \in \{2, \dots, k\}} S_i \dot{\cup} \{s_2\}, \quad L'_2 = \dot{\bigcup}_{i \in \{2, \dots, k\}} L_i,$$



■ **Figure 3** Case $k \geq 2$. Two MDPs $\mathcal{D}'_1 = \langle S'_1, Act, L_1, \varphi'_1, \ell'_1 \rangle$ and $\mathcal{D}'_2 = \langle S'_2, Act, L'_2, \varphi'_2, \ell'_2 \rangle$ are constructed using the k MDPs $\mathcal{D}_1, \dots, \mathcal{D}_k$.

$$\varphi'_2(s, a) = \begin{cases} \varphi_i(s, a) & \text{if } (s, a) \in \text{support}(\varphi_i) \text{ where } i = 2, \dots, k \\ \{q_0^i 1 \mapsto \frac{1}{k-1} \mid i = 2, \dots, k\} & \text{if } s = s_2 \text{ and } a = m \end{cases}$$

and $\ell'_2(s) = \begin{cases} \ell_i(s) & \text{if } s \in S_i \text{ where } i \in \{2, \dots, k\} \\ 1 & \text{if } s = s_2. \end{cases}$

Consider the two MDPs \mathcal{D}'_1 and \mathcal{D}'_2 . Assume that there is a general strategy α_i for \mathcal{D}'_i where $i \in \{1, 2\}$ such that s_1 and s_2 are probabilistic bisimilar in the induced LMCs. We define a general strategy for each MDP \mathcal{D}_i as follows: the general strategy for \mathcal{D}_1 maps each path ρ in \mathcal{D}_1 to $\alpha_1(s_1 m \rho)$ and the general strategy for \mathcal{D}_i where $i > 1$ maps each path ρ in \mathcal{D}_i to $\alpha_2(s_2 m \rho)$. We have that the k initial states in the induced LMCs are probabilistic bisimilar. For the other direction, assume there is a general strategy α_i for \mathcal{D}_i where $i \in \{1, \dots, k\}$ such that the k initial states in the induced LMCs are probabilistic bisimilar. Since both s_1 and s_2 only have a single available action m , s_1 and s_2 can be made probabilistic bisimilar by the following general strategies: the strategy for \mathcal{D}'_1 maps a path $s_1 m \rho$, where ρ is in \mathcal{D}_1 , to $\alpha_1(\rho)$ and the strategy for \mathcal{D}'_2 maps a path $s_2 m \rho$, where $i > 1$ and ρ is in \mathcal{D}_i , to $\alpha_i(\rho)$. ◀

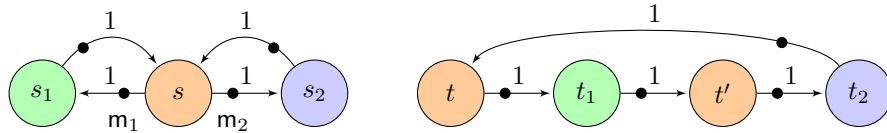
Lemmas 4 and 6 imply the main result of this section:

► **Theorem 7.** *The bisimilarity problem is EXPTIME-complete.*

3.3 The Bisimilarity Problem for an LMC and an MDP

We consider the subproblem when one MDP is restricted to be an LMC, that is, given an LMC and an MDP, and two states from the LMC and the MDP respectively, whether there exists a general strategy for the MDP to make these two states probabilistic bisimilar.

In general, memoryless strategies do not suffice for the problem. Consider the example in Figure 4. For the MDP on the left, the general strategy which in state s alternates between the two actions m_1 and m_2 witnesses that s and t are probabilistic bisimilar in the induced LMC. However, no memoryless strategy can make s and t probabilistic bisimilar.



■ **Figure 4** In this MDP no memoryless strategy witnesses $s \sim t$.

We show that this problem is NP-complete.

► **Lemma 8.** *The bimilarity problem for an LMC and an MDP is in NP.*

Proof. Given an LMC $\mathcal{M} = \langle S_1, L, \tau, \ell_1 \rangle$, an MDP $\mathcal{D} = \langle S_2, Act, L, \varphi, \ell_2 \rangle$, and two states $s_1 \in S_1$ and $s_2 \in S_2$, we decide whether there is a general strategy α for \mathcal{D} such that s_1 and s_2 are probabilistic bisimilar in the LMC $\mathcal{M} \oplus \mathcal{D}_\alpha$.

Consider the attacker-defender game defined in terms of the MDP $\mathcal{M} \oplus \mathcal{D}$ and the set $\{s_1, s_2\}$. According to Lemma 2 and Proposition 3, it suffices to check whether Defender has a memoryless winning strategy for the attacker-defender game.

Without loss of generality, assume that the LMC \mathcal{M} is a quotient LMC, that is, no two states in S_1 are probabilistic bisimilar, and all states in S_1 are reachable from s_1 . Each state $s \in S_1$ corresponds to a game state and we have $|S_1|$ game states. We guess the following components of a winning strategy for Defender:

- For each state $s \in S_1$, guess a set of states $E_s \subseteq S_2$. Intuitively, Defender claims that there is a general strategy such that all states in E_s are probabilistic bisimilar with s . Let S_w be the set of all the E_s sets. Let $E'_s = \{s\} \cup E_s$ be the game state which corresponds to the state s . The state s_2 is in E_{s_1} , and is also in E'_{s_1} .
- For each $E \in S_w$, guess a function $f_E : E \times Act \times S_2 \rightarrow S_w$ with $v \in f_E(u, m, v)$ for all $(u, m, v) \in E \times Act \times S_2$.

In the game state E'_s where $s \in S_1$, the probability distribution v over the successor game states is determined by the probability transition function of the LMC \mathcal{M} , that is, $v(E_t) = \tau(s)(t)$ for all $t \in S_1$ where $E_t \in S_w$ is the set of states which Defender claims can be made probabilistic bisimilar with t .

For each $E_s \in S_w$ and each $u \in E_s$, a memoryless strategy $\alpha_s^u \in \text{Distr}(Act(u))$ can be characterised by numbers $x_{s,u,m}$ where $m \in Act(u)$ such that $x_{s,u,m} = \alpha_s^u(m)$. We write \bar{x} for the collection $(x_{s,u,m})_{s \in S_1, u \in E_s, m \in Act(u)}$. Checking whether there is a memoryless winning strategy for Defender amounts to a feasibility test of the following linear program: $\exists \bar{x}$ such that

- $x_{s,u,m} \geq 0$ for all $s \in S_1, u \in E_s, m \in Act(u)$;
 - $\sum_{m \in Act(u)} x_{s,u,m} = 1$ for all $s \in S_1, u \in E_s$;
 - $\tau(s)(t) = \sum_{m \in Act(u)} \sum_{v \in S_2 \text{ s.t. } f_{E_s}(u,m,v)=E_t} x_{s,u,m} \varphi(u, m)(v)$ for all $s, t \in S_1$ and $u \in E_s$.
- Hence, this can be decided in polynomial time. ◀

NP-hardness follows from a reduction from the Subset Sum problem. The reduction is similar to [11, Theorem 19]. Given a set $P = \{p_1, \dots, p_n\}$ where $P \subseteq \mathbb{N}$ and $N \in \mathbb{N}$, Subset Sum asks whether there exists a set $Q \subseteq P$ such that $\sum_{p_i \in Q} p_i = N$. Subset Sum is known to be NP-complete [2].

► **Lemma 9.** *The bimilarity problem for an LMC and an MDP is NP-hard.*

By combining Lemma 8 and Lemma 9 we get:

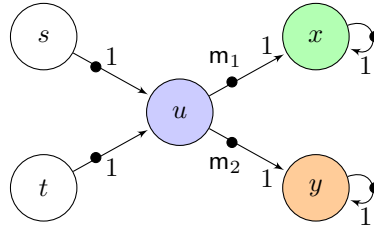
► **Theorem 10.** *The bimilarity problem for an LMC and an MDP is NP-complete.*

4 Bisimilarity Inequivalence Problem

In this section, we consider the bisimilarity inequivalence problem which, given an MDP and two initial states, asks whether there is a general strategy such that in the induced LMC the two states are not probabilistic bisimilar.

► **Example 11.** Consider again the 2-threaded program and MDP from Example 1 and Figure 1. This time we take the view of an eavesdropper. The memoryless strategy that, in h_0 and h_1 , chooses action m_1 with probability 0.4 and action m_2 with probability 0.6 induces an LMC in which h_0 and h_1 are not probabilistic bisimilar. Thus, if the eavesdropper has control over the scheduling and chooses a suitable strategy, they can glean information about the value of h . In this sense, the bisimilarity inequivalence problem can be viewed as the problem whether a program is safe from such information leaks, even if an adversary controls the scheduler.

► **Example 12.** In general, a single memoryless strategy does not suffice for the bisimilarity inequivalence problem. Consider the MDP in Figure 5. The general strategy which in state



■ **Figure 5** In this MDP no memoryless strategy witnesses $s \not\sim t$.

u selects the action m_1 after seeing the path su and selects the action m_2 after seeing the path tu witnesses that s and t are not probabilistic bisimilar in the induced LMC. However, given any memoryless strategy, s and t are probabilistic bisimilar in the induced LMC.

We write $\rho[\alpha]$ for the state ρ in the LMC \mathcal{D}_α induced by a strategy α . In the following, we show that if there is a general strategy such that s and t are not probabilistic bisimilar in the induced LMC, there are memoryless strategies σ and τ such that $s[\sigma]$ and $t[\tau]$ ² are not probabilistic bisimilar. These two memoryless strategies are not necessarily the same and they can be combined to form a general witnessing strategy for the bisimilarity inequivalence problem. Take the MDP in Figure 5 as an example. Although no single memoryless strategy witnesses the probabilistic bisimilarity inequivalence of s and t , with the memoryless strategies σ and τ where $\sigma(u) = m_1$ and $\tau(u) = m_2$, we have that $s[\sigma]$ and $t[\tau]$ are not probabilistic bisimilar.

A partition of the states S is a set X consisting of pairwise disjoint subsets E of S with $\bigcup_{E \in X} E = S$. Recall that $\varphi(s, m)(s')$ is the transition probability from s to s' when choosing action m . Similarly, $\varphi(s, m)(E) = \sum_{s' \in E} \varphi(s, m)(s')$ is the transition probability from $s \in S$ to $E \subseteq S$ when choosing action m . We write $\varphi(s, m)(X)$ to denote the probability distribution $(\varphi(s, m)(E))_{E \in X}$. We define $\varphi(s)(X) = \{\varphi(s, m)(X) : m \in Act(s)\}$, which is a set of probabilistic distributions over the partition X when choosing all available actions of s .

Abusing the notation slightly, for a memoryless strategy α we write $\alpha(s)(s')$ for the transition probability from s to s' in the LMC induced by α , that is, $\alpha(s)(s') = \sum_{m \in Act(s)} \alpha(s)(m) \varphi(s, m)(s')$. Similarly, the transition probability from s to $E \subseteq S$ in the LMC induced by a memoryless strategy is $\alpha(s)(E) = \sum_{s' \in E} \alpha(s)(s')$ and the probability distribution on a partition X is $\alpha(s)(X) = (\alpha(s)(E))_{E \in X}$.

The assumption that for two states s, t we have $s[\sigma] \sim t[\tau]$ for all general strategies σ and τ has consequences on s , on t , and on their successors, as detailed in the following lemma.

² Here s and t are paths of length 1.

► **Lemma 13.** *Let $s, t \in S$ with $s[\sigma] \sim t[\tau]$ for all general strategies σ and τ .*

1. *For all general strategies σ and τ , we have $s[\sigma] \sim s[\tau]$ and $t[\sigma] \sim t[\tau]$;*
2. *For all successors u of s and all general strategies σ and τ , we have $u[\sigma] \sim u[\tau]$.*

Given an MDP $\mathcal{D} = \langle S, Act, L, \varphi, \ell \rangle$, define a *superbisimulation* relation to be any equivalence relation $R \subseteq S \times S$ such that $(s, t) \in R$ if and only if $\ell(s) = \ell(t)$ and $\alpha(s)(X) = \alpha(t)(X)$ for all memoryless strategies α where $X = S/R$. The union of superbisimulations is a superbisimulation. Let superbisimilarity $\approx_{\mathcal{D}}$ be the largest superbisimulation, i.e., the union of all superbisimulations. The subscript \mathcal{D} can be omitted if it is clear from the context. We write $s \approx t$ if $(s, t) \in \approx$.

Let $\bar{S} := S \times \{0, 1\}$. We define an MDP $\bar{\mathcal{D}} = (\bar{S}, Act, L, \bar{\varphi}, \bar{\ell})$ where $\bar{\varphi}((s, i))(m)((t, i)) = \varphi(s, m)(t)$ for all $s, t \in S, i \in \{0, 1\}$ and $m \in Act$ and $\bar{\ell}((s, i)) = \ell(s)$ for all $(s, i) \in \bar{S}$. The MDP $\bar{\mathcal{D}}$ is basically made up of two disjoint copies of the original MDP \mathcal{D} .

The following lemma is a counterpart to Lemma 13. It spells out consequences of the assumption that $(s, 0)$ and $(t, 1)$ are superbisimilar in $\bar{\mathcal{D}}$.

► **Lemma 14.** *Let $s, t \in S$ with $(s, 0) \approx_{\bar{\mathcal{D}}} (t, 1)$.*

1. *Let R be any superbisimulation that contains $((s, 0), (t, 1))$. For any successor $(u, 0)$ of $(s, 0)$, there exists a successor $(v, 1)$ of $(t, 1)$ such that $((u, 0), (v, 1)) \in R$. Similarly, for any successor $(v, 1)$ of $(t, 1)$, there exists a successor $(u, 0)$ of $(s, 0)$ such that $((u, 0), (v, 1)) \in R$. In other words, any successor of $(s, 0)$ is superbisimilar with some successor of $(t, 1)$ and vice versa.*
2. *We have $(s, 1) \approx_{\bar{\mathcal{D}}} (t, 0)$, $(s, 0) \approx_{\bar{\mathcal{D}}} (s, 1)$ and $(t, 0) \approx_{\bar{\mathcal{D}}} (t, 1)$.*

The following theorem, whose proof is based on Lemmas 13 and 14, is the main technical result of this section. It provides a superbisimilarity-based characterisation of s and t being bisimilar under all general strategies.

► **Theorem 15.** *For all $s, t \in S$, we have $(s, 0) \approx (t, 1) \iff \forall$ general strategies $\sigma, \tau : s[\sigma] \sim t[\tau]$.*

Proof. (\Leftarrow) Let $S' = \{s \in S \mid \forall \text{ general strategies } \sigma, \tau : s[\sigma] \sim s[\tau]\}$ and $\bar{S}' = \{(s, i) \in \bar{S} \mid s \in S'\}$.

Let $R := \{((s, i), (t, j)) \in \bar{S} \times \bar{S} \mid \forall \text{ general strategies } \sigma, \tau : s[\sigma] \sim t[\tau]\}$.

Firstly, R is an equivalence relation on \bar{S}' :

- $R \subseteq \bar{S}' \times \bar{S}'$: Assume for all general strategies $\sigma, \tau : s[\sigma] \sim t[\tau]$. By Lemma 13, we have that $s[\sigma] \sim s[\tau]$ and $t[\sigma] \sim t[\tau]$ for all general strategies σ and τ . Both s and t are in S' , hence, $(s, i), (t, i) \in \bar{S}'$ for all $i \in \{0, 1\}$.
- R is reflexive. (trivial)
- R is symmetric. (trivial)
- R is transitive. (trivial)

By Lemma 13, all successors of a state $s \in S'$ in the MDP \mathcal{D} are in S' , hence, all successors of a state $(s, i) \in \bar{S}'$ are in \bar{S}' . Let $\bar{\mathcal{D}}'$ be the sub-MDP of $\bar{\mathcal{D}}$ that contains \bar{S}' and all the transitions between \bar{S}' .

To show that $(s, i) \approx (t, j)$ for any $((s, i), (t, j)) \in R$, we show that R is a superbisimulation of $\bar{\mathcal{D}}'$. The details can be found in the appendix.

(\Rightarrow) Let $S' = \{s \mid \exists (t, 1) \text{ such that } (s, 0) \approx (t, 1)\}$. Define a relation $R = \{(s, t) \mid (s, 0) \approx (t, 1)\}$ on S' . By Lemma 14, R is an equivalence relation. Let $X = S'/R$.

By Item 1 of Lemma 14, all successors of a state in S' are in S' . Let \mathcal{D}' be the sub-MDP that contains S' and all the transitions between S' . Let σ, τ be two arbitrary general strategies of \mathcal{D}' .

Let $R' = \{(\rho_1[\mu_1], \rho_2[\mu_2]) \mid \rho_1, \rho_2 \text{ are paths in } \mathcal{D}', (\text{last}(\rho_1), \text{last}(\rho_2)) \in R \text{ and } \mu_1, \mu_2 \in \{\sigma, \tau\}\}$ be a relation on the states of the LMC $\mathcal{D}'_\sigma \oplus \mathcal{D}'_\tau$, the disjoint union of the induced LMCs \mathcal{D}'_σ and \mathcal{D}'_τ .

Since R is an equivalence relation, it is not hard to see that R' is also an equivalence relation. We show in the appendix that R' is a probabilistic bisimulation on the states of the LMC $\mathcal{D}'_\sigma \oplus \mathcal{D}'_\tau$.

For any $(s, 0) \approx (t, 1)$, we have $(s[\sigma], t[\tau]) \in R'$, and hence, $s[\sigma]$ and $t[\tau]$ are probabilistic bisimilar in the LMC $\mathcal{D}'_\sigma \oplus \mathcal{D}'_\tau$. Since σ and τ are arbitrary general strategies for \mathcal{D}' and the sub-MDP \mathcal{D}' has all the available actions and successors of S' from \mathcal{D} , we have $s[\sigma] \sim t[\tau]$ for all general strategies σ and τ for \mathcal{D} . ◀

By Theorem 15, to decide whether there exist general strategies σ and τ such that $s[\sigma] \not\sim t[\tau]$, it suffices to decide whether $(s, 0) \not\approx (t, 1)$, which can be done by running [11, Algorithm 2] on the MDP $\bar{\mathcal{D}}$. This partition refinement algorithm is polynomial-time and the relation computed is superbisimilarity. By [11, Theorem 12, Corollary 13], if two states s and t are not superbisimilar, we can compute in polynomial time a memoryless strategy that witnesses $s \not\sim t$. Since the two states $(s, 0)$ and $(t, 1)$ are from two disjoint MDPs, if $(s, 0) \not\approx (t, 1)$, we can also compute in polynomial time two memoryless strategies σ and τ that witness $(s, 0)[\sigma] \not\sim (t, 1)[\tau]$, equivalently $(s)[\sigma] \not\sim (t)[\tau]$. Hence we have proved the following theorem.

► **Theorem 16.** *The bisimilarity inequivalence problem is in P. For any positive instance of the problem, there are memoryless strategies σ and τ such that $s[\sigma] \not\sim t[\tau]$. Further, for any positive instance of the problem, we can compute in polynomial time memoryless strategies σ and τ that witness $s[\sigma] \not\sim t[\tau]$.*

5 Conclusion

In this paper we have settled the decidability and complexity of the bisimilarity equivalence and inequivalence problems of MDPs under general strategies. Let us review the key technical steps.

We have proved that bisimilarity equivalence is decidable, albeit with a high, EXPTIME, computational complexity. For the EXPTIME upper bound we have provided a reduction to a non-stochastic two-player game, the attacker-defender game, which can be decided in EXPTIME. For the EXPTIME lower bound we have provided a reduction from the intersection emptiness problem for deterministic tree automata, which we have shown to be EXPTIME-hard. Further, we have obtained NP-completeness for the case that one of the MDPs is a Markov chain.

We have also shown that the bisimilarity inequivalence problem has much lower computational complexity, as it can be decided in polynomial time. This extends an earlier result that the corresponding inequivalence problem for memoryless strategies is in P. The key novel technique we have developed here is the notion of superbisimilarity, whose definition is similar to bisimilarity but with a different quantification over strategies.

References

- 1 Christel Baier. Polynomial time algorithms for testing probabilistic bisimulation and simulation. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification*, pages 50–61, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

- 2 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.
- 3 Benoît Delahaye. Consistency for parametric interval Markov chains. In Étienne André and Goran Frehse, editors, *2nd International Workshop on Synthesis of Complex Parameters, SynCoP 2015, April 11, 2015, London, United Kingdom*, volume 44 of *OASICS*, pages 17–32. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.
- 4 Salem Derisavi, Holger Hermanns, and William H. Sanders. Optimal state-space lumping in Markov chains. *Inf. Process. Lett.*, 87(6):309–315, 2003.
- 5 L. Doyen, T.A. Henzinger, and J.-F. Raskin. Equivalence of labeled Markov chains. *International Journal on Foundations of Computer Science*, 19(3):549–563, 2008.
- 6 Nathanaël Fijalkow, Stefan Kiefer, and Mahsa Shirmohammadi. Trace refinement in labelled Markov decision processes. *Logical Methods in Computer Science*, 16(2), 2020.
- 7 Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Probabilistic reachability for parametric Markov models. *Int. J. Softw. Tools Technol. Transf.*, 13(1):3–19, 2011.
- 8 Christian Hensel, Sebastian Junges, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. The probabilistic model checker Storm, 2020. [arXiv:arXiv:2002.07080](https://arxiv.org/abs/2002.07080).
- 9 Bengt Jonsson and Kim Guldstrand Larsen. Specification and refinement of probabilistic processes. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 266–277. IEEE Computer Society, 1991.
- 10 John G. Kemeny and J. Laurie Snell. *Finite Markov Chains*. Van Nostrand, 1960.
- 11 Stefan Kiefer and Qiyi Tang. Comparing labelled Markov decision processes. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, volume 182 of *LIPICs*, pages 49:1–49:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. [doi:10.4230/LIPICs.FSTTCS.2020.49](https://doi.org/10.4230/LIPICs.FSTTCS.2020.49).
- 12 Dexter Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 254–266. IEEE Computer Society, 1977. [doi:10.1109/SFCS.1977.16](https://doi.org/10.1109/SFCS.1977.16).
- 13 Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
- 14 Kim Guldstrand Larsen and Arne Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.
- 15 Azaria Paz. *Introduction to Probabilistic Automata*. Academic Press, 1971.
- 16 Andrei Popescu, Johannes Hölzl, and Tobias Nipkow. Formalizing probabilistic noninterference. In Georges Gonthier and Michael Norrish, editors, *Certified Programs and Proofs – Third International Conference*, volume 8307 of *Lecture Notes in Computer Science*, pages 259–275. Springer, 2013. [doi:10.1007/978-3-319-03545-1_17](https://doi.org/10.1007/978-3-319-03545-1_17).
- 17 Andrei Sabelfeld and David Sands. Probabilistic noninterference for multi-threaded programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 200–214. IEEE Computer Society, 2000. [doi:10.1109/CSFW.2000.856937](https://doi.org/10.1109/CSFW.2000.856937).
- 18 Marcel-Paul Schützenberger. On the definition of a family of automata. *Information and Control*, 4:245–270, 1961.
- 19 Helmut Seidl. Haskell overloading is DEXPTIME-complete. *Inf. Process. Lett.*, 52(2):57–60, 1994. [doi:10.1016/0020-0190\(94\)00130-8](https://doi.org/10.1016/0020-0190(94)00130-8).
- 20 Geoffrey Smith. Probabilistic noninterference through weak probabilistic bisimulation. In *16th IEEE Computer Security Foundations Workshop (CSFW-16 2003)*, pages 3–13. IEEE Computer Society, 2003. [doi:10.1109/CSFW.2003.1212701](https://doi.org/10.1109/CSFW.2003.1212701).
- 21 Wen-Guey Tzeng. A polynomial-time algorithm for the equivalence of probabilistic automata. *SIAM Journal on Computing*, 21(2):216–227, 1992.

- 22 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, 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 38–52. Springer, 2010.
- 23 Tobias Winkler, Sebastian Junges, Guillermo A. Pérez, and Joost-Pieter Katoen. On the complexity of reachability in parametric markov decision processes. In Wan J. 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 14:1–14:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICs.CONCUR.2019.14.

A Proofs of Section 3

► **Proposition 3.** *Given an MDP $\mathcal{D} = \langle S, Act, L, \varphi, \ell \rangle$ and a set $E_1 \subseteq S$, Defender has a winning strategy for the attacker-defender game if and only if there exists a general strategy α for \mathcal{D} such that in the LMC induced by α all states in E_1 are probabilistic bisimilar.*

Proof. (\implies) We show in the following that all states in E_1 are probabilistic bisimilar in the LMC induced by α defined in the main text.

Given the MDP \mathcal{D} and the general strategy α , an LMC $\mathcal{D}(\alpha) = \langle \mathcal{P}, L, \tau_\alpha, \ell_\alpha \rangle$ is induced. To show that all states in E_1 are probabilistic bisimilar in the LMC $\mathcal{D}(\alpha)$, we show that for any two states ρ_1 and ρ_2 of $\mathcal{D}(\alpha)$, if $\rho_1, \rho_2 \in \mathcal{P}$ and $W(\rho_1) = W(\rho_2)$, we have $\rho_1 \sim \rho_2$. It suffices to show the relation $\sim_W \subseteq \mathcal{P} \times \mathcal{P}$ defined by $\rho_1 \sim_W \rho_2$ if and only if $W(\rho_1) = W(\rho_2)$ is a probabilistic bisimulation.

Let $\rho_1, \rho_2 \in \mathcal{P}$ be two states in the LMC $\mathcal{D}(\alpha)$ and $E = W(\rho_1) = W(\rho_2)$ be a state at the beginning of some round of the attacker-defender game. Let $s_1 = \text{last}(\rho_1)$ and $s_2 = \text{last}(\rho_2)$. We have $\ell_\alpha(\rho_1) = \ell(s_1) = \ell(s_2) = \ell_\alpha(\rho_2)$ since $s_1, s_2 \in E$ and all states in E have the same label. Defender has a winning strategy for E and we assume that she chooses a set $S' \subseteq 2^S$, a distribution v on S' , and a function f . Consider a set $E' \in \text{support}(v)$. By definition of \sim_W , for all $\rho_1 m_1 t_1, \rho_2 m_2 t_2 \in \mathcal{P}$ such that $W(\rho_1 m_1 t_1) = W(\rho_2 m_2 t_2) = E'$ we have $\rho_1 m_1 t_1 \sim_W \rho_2 m_2 t_2$. We also have

$$\begin{aligned}
& v(E') \\
= & \sum_{m_1 \in \text{Act}(s_1)} \sum_{t_1 \in S \text{ s.t. } f(s_1, m_1, t_1) = E'} \alpha^{s_1}(m_1) \varphi(s_1, m_1)(t_1) \\
& \quad \text{[property satisfied by } v \text{]} \\
= & \sum_{m_1 \in \text{Act}(s_1)} \sum_{t_1 \in E'} \alpha(\rho_1)(m_1) \varphi(s_1, m_1)(t_1) \\
= & \sum_{m_1 \in \text{Act}(s_1)} \sum_{t_1 \in E'} \tau_\alpha(\rho_1)(\rho_1 m_1 t_1). \quad \text{[definition of } \tau_\alpha \text{]} \\
= & \sum_{\rho_1 m_1 t_1 \in \mathcal{P}_{i+1} \text{ and } t_1 \in E'} \tau_\alpha(\rho_1)(\rho_1 m_1 t_1).
\end{aligned}$$

$$\text{Similarly, we have } v(E') = \sum_{\rho_2 m_2 t_2 \in \mathcal{P}_{i+1} \text{ and } t_2 \in E'} \tau_\alpha(\rho_2)(\rho_2 m_2 t_2).$$

Thus, the relation \sim_W is a probabilistic bisimulation. Consider any state $s \in \mathcal{P}_1$ of $\mathcal{D}(\alpha)$. We have $W(s) = E_1$, it concludes that all states in E_1 are probabilistic bisimilar.

(\Leftarrow) We verify that the objects chosen by Defender satisfies the required properties. For all $s \in E$ and all $E'_i \in S'$ where $1 \leq i \leq k$ we have

$$\begin{aligned}
& v(E'_i) \\
&= v'(B_i) \\
&= \sum_{smt \in B_i} \alpha_E(s)(\mathbf{m})\varphi(s, \mathbf{m})(t) \\
&= \sum_{smt \in B_i} \alpha^s(\mathbf{m})\varphi(s, \mathbf{m})(t) \\
&= \sum_{\mathbf{m} \in Act(s)} \sum_{t \in S \text{ s.t. } f(s, \mathbf{m}, t) = E'_i} \alpha^s(\mathbf{m})\varphi(s, \mathbf{m})(t).
\end{aligned}$$

If the game starts with an equalisable set and all the future game states are equalisable sets, Defender can always choose objects with required properties and hence win the game. It remains to show that all sets in S' are equalisable. Consider the probabilistic bisimulation class B_i which is used to construct $E'_i \in S'$. We define a general strategy α_i for any path that starts with a state $t \in E'_i$. Let ρ be such a path. We define $\alpha_i(\rho) := \alpha_E(sm\rho)$ where $s \in E$ and $smt \in B_i$. Basically, after going along the path ρ , the general strategy α_i plays as $\alpha_E(sm\rho)$. The LMC induced by the general strategy α_i can be seen as part of the LMC induced by α_E where a state ρ in the former LMC corresponds to the state $sm\rho$ in the latter. That is, a state $t \in E'_i$ in the LMC induced by α_i corresponds to a state $smt \in B_i$ in the LMC induced by α_E . Since all states in B_i are probabilistic bisimilar in the LMC induced by α_E , all states in E'_i in the LMC induced by α_i are also probabilistic bisimilar. \blacktriangleleft

► **Lemma 5.** *The intersection nonemptiness problem is EXPTIME-hard.*

Proof. We give a polynomial-time reduction from the problem of acceptance of a word by a PSPACE-bounded alternating Turing machine. Let $M = (P_{\exists}, P_{\forall}, \Gamma, \Delta, p_0, p_{acc}, p_{rej})$ be a PSPACE-bounded alternating Turing machine, where $P = P_{\exists} \cup P_{\forall}$ is the finite set of (control) states partitioned into existential states P_{\exists} and universal states P_{\forall} , and Γ is the tape alphabet, and $\Delta \subseteq P \times \Gamma \times P \times \Gamma \times \{-1, +1\}$ is the transition relation, and $p_0, p_{acc}, p_{rej} \in P$ are the initial, accepting, rejecting state, respectively. A transition $(p, a, p', a', d) \in \Delta$ means that if M is in state p and its read-write head reads letter a , then M rewrites the contents of the current cell with the letter a' , moves the head in direction d (either left if $d = -1$, or right if $d = +1$), and changes its state to p' . Such a transition is called *outgoing* from (p, a) . We assume that for all $(p, a) \in P \times \Gamma$ there is at least one outgoing transition, and for all $(p, a) \in P_{\forall} \times \Gamma$ there are exactly two outgoing transitions. A *configuration* of M is given by the current state $p \in P$, the tape content (from Γ^*), and the position of the head. If the current state p is existential, i.e., $p \in P_{\exists}$, and the head reads $a \in \Gamma$, then the *existential* player picks a transition that is outgoing from (p, a) . Similarly for the universal states and the *universal* player. Starting from an input word $w \in \Gamma^*$ on the tape, strategies of the two players define a *computation*, i.e., a sequence of configurations. It is the goal of the existential player to form a computation that reaches p_{acc} ; the goal of the universal player is to avoid this. We can assume that all computations reach either p_{acc} or p_{rej} and no configuration is repeated before that (this is achieved, e.g., using a counter on the tape), and after reaching p_{acc} or p_{rej} the control state no longer changes. If the existential player has a strategy to reach p_{acc} no matter what strategy the universal player uses, then we say that M *accepts* w . Since $\text{APSPACE} = \text{EXPTIME}$, there is a (fixed) PSPACE-bounded alternating Turing machine, say $M = (P_{\exists}, P_{\forall}, \Gamma, \Delta, p_0, p_{acc}, p_{rej})$, such that it is EXPTIME-complete to decide if it accepts a given input word.

Let $w \in \Gamma^n$ be the input word. The Turing machine M uses only space polynomial in n , say N . We construct, in polynomial time, DTTAs $\mathcal{A}_0, \dots, \mathcal{A}_N$ such that $\bigcap_{i=0}^N L(\mathcal{A}_i) \neq \emptyset$ if and only if M accepts w . To do so, we encode a strategy of the existential player by a tree whose branches encode the computations that are consistent with the existential player's strategy; the strategy of the universal player effectively chooses one branch in the tree, which encodes the computation defined by both players' strategies. We want to construct DTTAs $\mathcal{A}_0, \dots, \mathcal{A}_N$ so that $\bigcap_{i=0}^N L(\mathcal{A}_i)$ contains exactly those trees that encode a winning strategy of the existential player. Each \mathcal{A}_i ensures some aspect of correctness of such strategy trees; when \mathcal{A}_i encounters a problem with a tree, it uses the partiality of its transition function δ so that it does not have a run on that tree.

The trees consist of blocks of the form $a_1 \cdots a_{m-1} \# a_m \cdots a_N p$, which encode a configuration. Here, $a_1 \cdots a_N \in \Gamma^N$ is the tape content, the position of the symbol $\#$ indicates the position of the head (reading a_m), and $p \in P$ is the current control state. As ranked alphabet we take $\Sigma = \Gamma \cup \{\#\} \cup P$ (we can assume this is a union of disjoint sets), where all symbols have rank 1, except those in P_\forall , which have rank 2. As a result, a p -node at the end of a block, where $p \in P_\exists$, has one child, which starts another block encoding a successor configuration. Similarly, if $p \in P_\forall$, then the node has two children, both of which start another block encoding successor configurations.

We construct, in polynomial time, DTTA \mathcal{A}_0 so that it ensures that the input tree starts with $\# w_1 \cdots w_n \sqcup \cdots \sqcup p_0$, encoding the initial configuration of M ; here $w = w_1 \cdots w_n$ is the input word, which is followed by $N - n$ blank symbols \sqcup . DTTA \mathcal{A}_0 also ensures that p_{rej} occurs *nowhere* in the tree. It also ensures that the blocks, which encode configurations as described above, are well-formed in that each of them consists of N symbols from Γ , a single occurrence of $\#$ in front of one of the symbols from Γ , and a $p \in P$ at the end.

Let $i \in \{1, \dots, N\}$. We construct, in polynomial time, DTTA \mathcal{A}_i so that it ensures the following properties for all blocks in the input tree.

- If the symbol $\#$ precedes the i th symbol from Γ in the block, say a , and the block ends with $p \in P_\exists$ and, in the directly following block, the symbol $\#$ precedes the $(i + d)$ th symbol ($i \in \mathbb{N}$) from Γ in the block and the i th symbol from Γ in the block is a' and the block ends with $p' \in \Gamma$, then $(p, a, p', a', d) \in \Delta$.
- If the symbol $\#$ precedes the i th symbol from Γ in the block, say a , and the block ends with $p \in P_\forall$ and $(p, a, p_1, a_1, d_1), (p, a, p_2, a_2, d_2) \in \Delta$ are the two outgoing transitions from (p, a) (we assume that these two transitions are ordered in some way), then in the left (respectively, right) successor block the symbol $\#$ precedes the $(i + d_1)$ th (respectively, $(i + d_2)$ th) symbol from Γ and the i th symbol from Γ in the block is a_1 (respectively, a_2) and the block ends with p_1 (respectively, p_2).
- If the symbol $\#$ does not precede the i th symbol from Γ in the block, say a , then the i th symbol from Γ in the (either one or two) directly following block(s) is also a .

In this way, $\bigcap_{i=0}^N L(\mathcal{A}_i)$ contains exactly those trees that encode a winning strategy of the existential player. \blacktriangleleft

► **Lemma 6.** *The bisimilarity problem is EXPTIME-hard.*

Proof. We show in the following that $\bigcap_{i=1}^k L(\mathcal{A}_i) \neq \emptyset$ if and only if there is a general strategy α_i for each MDP \mathcal{D}_i such that the k initial states in the induced LMCs are probabilistic bisimilar.

(\implies) Assume $\bigcap_{i=1}^k L(\mathcal{A}_i) \neq \emptyset$. Let t be an ordered tree over Σ in the intersection $\bigcap_{i=1}^k L(\mathcal{A}_i)$. There is a deterministic general strategy α_i for each MDP \mathcal{D}_i corresponding to t . An example of a deterministic general strategy of an MDP corresponding to a run tree is given in Figure 2.

We define a function f_i which maps a path starting from the root of the run tree of \mathcal{A}_i (t labelled with states of \mathcal{A}_i) to a path in the MDP \mathcal{D}_i :

- $f_i(a) = q_0^i 1$ where the root of the tree t is an a -node (the root of t is labelled with $q_0^i 1$);
- $f_i(xa) = f_i(x)\text{last}(x)(q\text{last}(x))\mathbf{m}(q'j)$ where $x \in \Sigma^+$, $a \in \Sigma$, $q \in Q_i$, l is a number and $ql = \text{last}(f_i(x))$, a is the j th child of its parent and is labelled with $q' \in Q_i$.

For a path $x \in \Sigma^+$ of t , $f_i(x)$ is a path in \mathcal{D}_i which ends with a state of the form qj where $q \in Q_i$ and j is a number. We now define the deterministic general strategy α_i for the MDP \mathcal{D}_i . For a path ρ in the MDP \mathcal{D}_i , we have

$$\alpha_i(\rho) = \begin{cases} a & \text{if } f_i(x) = \rho \text{ for a path } x \in \Sigma^+ \text{ of the tree } t \text{ and } a = \text{last}(x) \\ \mathbf{m} & \text{if } \text{last}(\rho) \text{ is of the form } q_i a \text{ where } q_i \in Q_i \text{ and } a \in \Sigma \\ \mathbf{m}' & \text{otherwise, } \mathbf{m}' \text{ is an arbitrary available action of } \text{last}(\rho) \end{cases}$$

Every state in the induced LMC $\mathcal{D}_i(\alpha_i)$ corresponds to a tree path. The states ρ in the induced LMC $\mathcal{D}_i(\alpha_i)$, where $\text{last}(\rho)$ is of the form $q_i j$ such that $q_i \in Q_i$ and j is a number, correspond to the tree path x such that $f_i(x) = \rho$. The states $\rho a(q_i a)$ where $q_i \in Q_i$ and $a \in \Sigma$ in the induced LMC $\mathcal{D}_i(\alpha_i)$ correspond to the tree path x such that $f_i(x) = \rho$. Let P be the union of the states of the induced LMCs. Let $R \subseteq P \times P$ be a relation in which $(\rho, \rho') \in R$ if and only if either (1) there exist a tree path $x \in \Sigma^+$ and $i, j \in \{1, \dots, k\}$ such that $f_i(x) = \rho$ and $f_j(x) = \rho'$ or (2) both states correspond to the same tree path and $\text{last}(\rho)$ (respectively, $\text{last}(\rho')$) is of the form qa where $q \in Q_i$ for some i and $a \in \Sigma$. In case (1), we have the pairs of states ρ and ρ' in the induced LMCs where $\text{last}(\rho)$ (respectively, $\text{last}(\rho')$) is of the form qj where $q \in Q_i$ for some i and j is a number. We can show that the relation R is a probabilistic bisimulation. Hence, the initial states $q_0^i 1$ of the LMCs $\mathcal{D}_i(\alpha_i)$ are probabilistic bisimilar.

(\Leftarrow) Assume that there is a general strategy α_i for each MDP \mathcal{D}_i such that the k initial states in the induced LMCs are probabilistic bisimilar. Since the general strategies are possibly randomised, there might be multiple trees embedded in each of the induced LMC. We extract one common tree from these LMCs. We will then show that this tree witnesses the intersection nonemptiness of the k DTTAs as there is a run on this common tree for every DTTA \mathcal{A}_i .

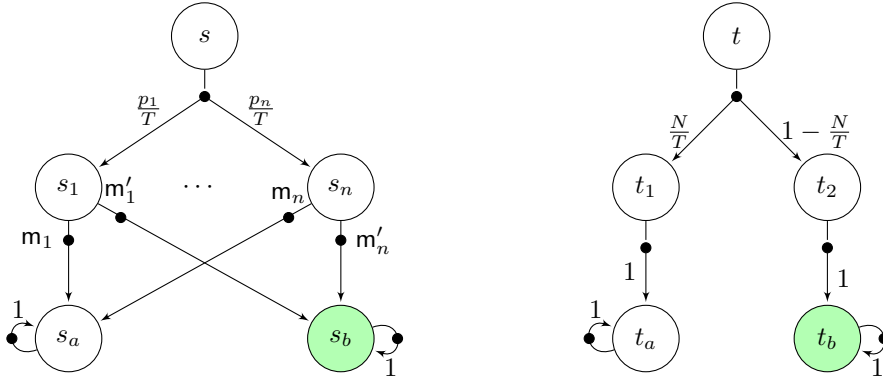
Let us prioritise the symbols in Σ such that each symbol has a different priority. In the following, we show how to obtain an ordered tree t_i for the MDP \mathcal{D}_i level by level. The process is similar to the construction of the LMC induced by the general strategy α_i . We first add the initial state $q_0^i 1$ (it is also a path with only one state) to the tree and make it the root of the tree. We call this node $q_0^i 1$. For every node ρ in the tree without children, we assign a symbol from Σ to it and add the children as follows. Since the LMCs induced by the general strategies are probabilistic bisimilar, there is no state in the LMCs labelled with deadlock_i . For a node ρ in the tree, we have that $\text{last}(\rho)$ is of the form qj where q is a state in the DTTA \mathcal{A}_i and j is a number. The strategy α_i over the path ρ is a distribution over the set $\text{support}(\alpha_i(\rho)) \subseteq \text{Act}_i(qj)$, a subset of the available actions at qj . Let $a \in \Sigma$ be the one with the highest priority in $\text{support}(\alpha_i(\rho))$ and now make the node ρ an a -node. From now on, we only consider paths in the MDP \mathcal{D}_i with prefix $\rho a(qa)$ and discard all the other paths with prefix ρ . The strategy α_i on the path $\rho a(qa)$ is a Dirac distribution on the only available action \mathbf{m} . Each of the $\text{rank}(a)$ successor of the path $\rho a(qa)$ is of the form $\rho a(qa)\mathbf{m}(q'j')$, which is then added as the j' th child of the node ρ . The tree is infinite and is over Σ .

For all the MDPs, the ordered tree constructed in the above way is the same. The nodes have the same symbols from Σ . We label each node ρ in t_i with $q \in \mathcal{A}_i$, where $qj = \text{last}(\rho)$. It is not hard to verify that this labelling of the tree t_i is a run of the DTTA \mathcal{A}_i on t_i . Thus, the tree is in the intersection of the languages of the DTTAs. \blacktriangleleft

► **Lemma 9.** *The bisimilarity problem for an LMC and an MDP is NP-hard.*

Proof. Given an instance of Subset Sum $\langle P, N \rangle$ where $P = \{p_1, \dots, p_n\}$ and $N \in \mathbb{N}$, we construct an MDP \mathcal{D} ; see Figure 6. Let $T = \sum_{p_i \in P} p_i$. In the MDP, state s transitions to state s_i with probability p_i/T for all $1 \leq i \leq n$. Each state s_i has two available actions, each transitions to s_a and s_b by taking the action m_i and m'_i , respectively. State t transitions to t_1 and t_2 with probability N/T and $1 - N/T$, respectively. All the remaining states have only one available action transitioning to the successor state with probability one. States s_b and t_b have label b and all other states have label a .

It suffices to consider memoryless strategies in the MDP constructed since the only states that we need to specify a strategy are the s_i states and there is only one path from s to any of the s_i state.



■ **Figure 6** The MDP \mathcal{D} in the reduction for NP-hardness of the bisimilarity problem. All states have the same label a except s_b and t_b which have label b .

Next, we show that $\langle P, N \rangle \in \text{Subset Sum}$ if and only if there exists a general strategy α such that s and t are probabilistic bisimilar in the induced LMC \mathcal{D}_α .

Intuitively, making s_i probabilistic bisimilar with t_1 simulates the membership of p_i in Q . Conversely, making s_i probabilistic bisimilar with t_2 simulates the membership of p_i in $P \setminus Q$.

(\implies) Let $Q \subseteq P$ be the set such that $\sum_{p_i \in Q} p_i = N$. Let α be an memoryless deterministic strategy such that if $p_i \in Q$ then $\alpha(s_i) = m_i$ and $\alpha(s_i) = m'_i$ otherwise. It is clear that in the induced LMC, all states s_i where $p_i \in Q$ are probabilistic bisimilar with t_1 and all the other states are probabilistic bisimilar with t_2 . Since $\sum_{p_i \in Q} p_i = N$, s and t are probabilistic bisimilar in the induced LMC.

(\impliedby) Assume there is a memoryless strategy α such that s and t are probabilistic bisimilar in the induced LMC \mathcal{D}_α . We have $t_1 \not\sim t_2$. Let S_1 be the set of successor states of s that are probabilistic bisimilar to t_1 . Then, $\sum_{s_i \in S_1} \frac{p_i}{T} = \tau(s)(S_1) = \tau(t)(t_1) = \frac{N}{T}$. Let $Q = \{p_i \in P \mid s_i \in S_1\}$. We have $Q \subseteq P$ be the set such that $\sum_{p_i \in Q} p_i = N$. ◀

B Proofs of Section 4

► **Lemma 13.** *Let $s, t \in S$ with $s[\sigma] \sim t[\tau]$ for all general strategies σ and τ .*

1. *For all general strategies σ and τ , we have $s[\sigma] \sim s[\tau]$ and $t[\sigma] \sim t[\tau]$;*
2. *For all successors u of s and all general strategies σ and τ , we have $u[\sigma] \sim u[\tau]$.*

Proof.

1. Assume for all general strategies $\sigma, \tau : s[\sigma] \sim t[\tau]$. Since bisimulation is an equivalence relation, we have that for all general strategies $\sigma, \tau : s[\sigma] \sim s[\tau]$. Similarly, we have $t[\sigma] \sim t[\tau]$ for all general strategies σ and τ .
2. Assume $s[\sigma] \sim t[\tau]$ for all general strategies σ and τ . By Item 1, we have $s[\sigma] \sim s[\tau]$ for all general strategies σ and τ . Assume there exist a successor u of s and general strategies σ, τ such that $u[\sigma] \not\sim u[\tau]$.

Let σ' be a general strategy that in s takes an action m to reach u with positive chance in the first step and plays the strategy σ once u is reached, that is, $\varphi(s, m)(u) > 0$, $\sigma'(s)(m) = 1$ and $\sigma'(sm\rho_u) = \sigma(\rho_u)$ where ρ_u is any path starting with u .

Let τ' be the same general strategy as σ' except that it plays the strategy τ once u is reached, that is, $\tau'(\rho) = \sigma'(\rho)$ for any path $\rho \neq sm\rho_u$ and $\tau'(sm\rho_u) = \tau(\rho_u)$ where ρ_u is any path starting with u .

In the LMCs $\mathcal{D}_{\sigma'}$ and $\mathcal{D}_{\tau'}$, for all the other successor states v of s in \mathcal{D} where $v \neq u$, we have $smv[\sigma'] \sim smv[\tau']$. However, since $smu[\sigma'] \not\sim smu[\tau']$, we have $s[\sigma'] \not\sim s[\tau']$, which contradicts that $s[\sigma] \sim s[\tau]$ holds for all general strategies σ and τ . \blacktriangleleft

► **Lemma 14.** *Let $s, t \in S$ with $(s, 0) \approx_{\bar{\mathcal{D}}} (t, 1)$.*

1. *Let R be any superbisimulation that contains $((s, 0), (t, 1))$. For any successor $(u, 0)$ of $(s, 0)$, there exists a successor $(v, 1)$ of $(t, 1)$ such that $((u, 0), (v, 1)) \in R$. Similarly, for any successor $(v, 1)$ of $(t, 1)$, there exists a successor $(u, 0)$ of $(s, 0)$ such that $((u, 0), (v, 1)) \in R$. In other words, any successor of $(s, 0)$ is superbisimilar with some successor of $(t, 1)$ and vice versa.*
2. *We have $(s, 1) \approx_{\bar{\mathcal{D}}} (t, 0)$, $(s, 0) \approx_{\bar{\mathcal{D}}} (s, 1)$ and $(t, 0) \approx_{\bar{\mathcal{D}}} (t, 1)$.*

Proof. Assume $(s, 0) \approx (t, 1)$.

1. Let R be a superbisimulation such that $((s, 0), (t, 1)) \in R$. Let $X = \bar{S}/R$. For a contradiction, assume for a successor of $(s, 0)$, say $(u, 0)$, there exists no successor $(v, 1)$ of $(t, 1)$ such that $((u, 0), (v, 1)) \in R$. Then, $(u, 0)$ is in an equivalence class $E \in X$ in which there are no successors of $(t, 1)$. Let α be a memoryless strategy such that $\alpha((s, 0))((u, 0)) > 0$. We have $\alpha((s, 0))(E) > 0 = \alpha((t, 1))(E)$, which contradicts that $((s, 0), (t, 1)) \in R$.
2. If $(s, 0) \approx (t, 1)$ then, by symmetry, $(s, 1) \approx (t, 0)$.

Let $R = \{((s, i), (s, i)) \mid s \in S \wedge i \in \{0, 1\}\} \cup \{((s, 0), (s, 1)), ((s, 1), (s, 0)) \mid \exists t \in S \text{ such that } (s, 0) \approx (t, 1)\}$. To show $(s, 0) \approx (s, 1)$ and $(t, 0) \approx (t, 1)$, it suffices to show that R is a superbisimulation.

Firstly, note that R is an equivalence relation.

For any $((s, i), (s, j)) \in R$, clearly we have $\bar{\ell}((s, i)) = \bar{\ell}((s, j))$. Let $X = \bar{S}/R$. It remains to show that for all $((s, i), (s, j)) \in R$ it holds that $\alpha((s, i))(X) = \alpha((s, j))(X)$ for all memoryless strategies α . Assume $((s, i), (s, j)) \in R$. If $i = j$, it is trivially true. Otherwise, $j = 1 - i$. There must exist a state $t \in S$ such that $(t, j) \approx (s, i)$. For any successor (u, i) of (s, i) , by Item 1, there is some successor (v, j) of (t, j) such that $((u, i), (v, j)) \in R$. Thus, for all successors (u, i) of (s, i) , $((u, i), (u, j))$ is in R and $(u, i), (u, j)$ are in the same equivalence class in X . Thus, we have $\alpha((s, i))(X) = \alpha((s, j))(X)$ for all memoryless strategies α .

Hence, R is a superbisimulation. \blacktriangleleft

► **Theorem 15.** *For all $s, t \in S$, we have $(s, 0) \approx (t, 1) \iff \forall$ general strategies $\sigma, \tau : s[\sigma] \sim t[\tau]$.*

Proof. (\Leftarrow) To show that $(s, i) \approx (t, j)$ for any $((s, i), (t, j)) \in R$, it suffices to show that R is a superbisimulation of $\bar{\mathcal{D}}'$.

Let α be an arbitrary memoryless strategy for $\bar{\mathcal{D}}'$. We define a relation $R_\alpha = \{((s, i), (t, j)) \in \bar{S}' \times \bar{S}' \mid (s, i)[\alpha] \sim (t, j)[\alpha]\}$. Then, R_α is a probabilistic bisimulation on \bar{S}' and $X_\alpha = \bar{S}'/R_\alpha$ is the set of probabilistic bisimulation classes.

Next, we show $R = R_\alpha$. It is obvious that $R \subseteq R_\alpha$. To show $R_\alpha \subseteq R$, we assume $((s, i), (t, j)) \in \bar{S}' \times \bar{S}'$ and $(s, i)[\alpha] \sim (t, j)[\alpha]$. Since $(s, i) \in \bar{S}'$, we have $(s, i)[\alpha] \sim (s, i)[\sigma]$ for all general strategies σ . Similarly, we have $(t, j)[\alpha] \sim (t, j)[\tau]$ for all general strategies τ . Since probabilistic bisimulation is transitive, we have $(s, i)[\sigma] \sim (s, i)[\alpha] \sim (t, j)[\alpha] \sim (t, j)[\tau]$ for all general strategies σ and τ .

Let $X = \bar{S}'/R$. We have $X_\alpha = \bar{S}'/R_\alpha = \bar{S}'/R = X$. Let $((s, i), (t, j)) \in R$. We have $((s, i), (t, j)) \in R_\alpha$ and $\alpha((s, i))(X) = \alpha((s, i))(X_\alpha) = \alpha((t, j))(X_\alpha) = \alpha((t, j))(X)$. Since α was arbitrary, R is a superbisimulation of $\bar{\mathcal{D}}'$.

(\Rightarrow) We show that the relation R' defined in the proof in the main text is a probabilistic bisimulation on the states of the LMC $\mathcal{D}'_\sigma \oplus \mathcal{D}'_\tau$.

Let X' be the partition of the states of the LMC $\mathcal{D}'_\sigma \oplus \mathcal{D}'_\tau$ with respect to R' . Let $(\rho_1[\mu_1], \rho_2[\mu_2]) \in R'$ with $\text{last}(\rho_1) = s$ and $\text{last}(\rho_2) = t$. To show R' is a probabilistic bisimulation on the states of the LMC $\mathcal{D}'_\sigma \oplus \mathcal{D}'_\tau$, it suffices to show that (1) $\rho_1[\mu_1]$ and $\rho_2[\mu_2]$ have the same label; (2) the probability distributions over X' from $\rho_1[\mu_1]$ and from $\rho_2[\mu_2]$ are the same.

Since $(\rho_1[\mu_1], \rho_2[\mu_2]) \in R'$, we have $(s, 0) \approx (t, 1)$. It follows that $\rho_1[\mu_1]$ and $\rho_2[\mu_2]$ have the same label. Furthermore, we have $\alpha_1(s)(X) = \alpha_2(t)(X)$ for all memoryless strategies α_1 and α_2 for \mathcal{D} . Let $\alpha_1(s) = \mu_1(\rho_1)$, $\alpha_2(t) = \mu_2(\rho_2)$. The successors of $(\rho_1[\mu_1], (\rho_2[\mu_2]))$ can be partitioned with respect to R' and each class can be identified by a set $E \in X$. We define $E_1 = \{\rho_1 m u[\mu_1] \mid m \in \text{Act}(s) \wedge u \in E\}$, which is the set of successors of $\rho_1[\mu_1]$ corresponding to E . Similarly, define $E_2 = \{\rho_2 m u[\mu_2] \mid m \in \text{Act}(t) \wedge u \in E\}$, which is the set of successors of $\rho_2[\mu_2]$ corresponding to E . We have that the transition probability from $\rho_1[\mu_1]$ to E_1 is $\sum_{m \in \text{Act}(s)} \mu_1(\rho_1)(m) \varphi(s, m)(E_1)$, which is equal to $\alpha_1(s)(E)$. Similarly, the transition probability from $\rho_2[\mu_2]$ to E_2 is $\sum_{m \in \text{Act}(t)} \mu_2(\rho_2)(m) \varphi(t, m)(E_2)$, which is equal to $\alpha_2(t)(E)$. Thus, the probability distribution over X' from $\rho_1[\mu_1]$ is equal to that from $\rho_2[\mu_2]$, and we conclude that R' is a probabilistic bisimulation on the states of the LMC $\mathcal{D}'_\sigma \oplus \mathcal{D}'_\tau$. \blacktriangleleft

Pareto-Rational Verification

Véronique Bruyère

Université de Mons (UMONS), Mons, Belgium

Jean-François Raskin

Université libre de Bruxelles (ULB), Brussels, Belgium

Clément Tamines

Université de Mons (UMONS), Mons, Belgium

Abstract

We study the rational verification problem which consists in verifying the correctness of a system executing in an environment that is assumed to behave rationally. We consider the model of rationality in which the environment only executes behaviors that are Pareto-optimal with regard to its set of objectives, given the behavior of the system (which is committed in advance of any interaction). We examine two ways of specifying this behavior, first by means of a deterministic Moore machine, and then by lifting its determinism. In the latter case the machine may embed several different behaviors for the system, and the universal rational verification problem aims at verifying that all of them are correct when the environment is rational. For parity objectives, we prove that the Pareto-rational verification problem is **co-NP**-complete and that its universal version is in **PSPACE** and both **NP**-hard and **co-NP**-hard. For Boolean Büchi objectives, the former problem is Π_2 **P**-complete and the latter is **PSPACE**-complete. We also study the case where the objectives are expressed using LTL formulas and show that the first problem is **PSPACE**-complete, and that the second is **2EXPTIME**-complete. Both problems are also shown to be fixed-parameter tractable for parity and Boolean Büchi objectives.

2012 ACM Subject Classification Software and its engineering → Formal methods; Theory of computation → Logic and verification; Theory of computation → Solution concepts in game theory

Keywords and phrases Rational verification, Model-checking, Pareto-optimality, ω -regular objectives

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.33

Related Version *Full Version*: <https://arxiv.org/abs/2202.13485> [19]

Funding This work is partially supported by the two PDR projects *Subgame perfection in graph games* and *Rational* (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), and the COST Action 16228 *GAMENET* (European Cooperation in Science and Technology).

1 Introduction

Formal verification is essential to ensure the correctness of systems responsible for critical tasks. Many advancements have been made in the field of formal verification both in terms of theoretical foundations and tool development, and computer-aided verification techniques, such as *model-checking* [4, 7], are now widely used in industry. In the classical approach to verification, it is assumed that the system designer provides (i) a model of the system to verify, together with (ii) a model of the environment in which the system will be executed, and (iii) a specification φ (e.g. an ω -regular property) that must be enforced by the system. Those models are usually nondeterministic automata that cover all possible behaviors of both the system and the environment. The model-checking algorithm is then used to decide if all executions of the system in the environment are correct with regard to φ . Unfortunately, in some settings, providing a faithful and sufficiently precise model of the environment may



© Véronique Bruyère, Jean-François Raskin, and Clément Tamines;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 33; pp. 33:1–33:20



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

be *difficult or even impossible*. This is particularly true in heterogeneous systems composed of software entities interacting with human users, e.g. self-driving cars interacting with human drivers. Alternative approaches are thus needed in order to verify such complex multi-agent systems. One possible solution to this problem is to consider more *declarative* ways of modeling the environment. Instead of considering an operational model of each agent composing the environment, in this paper, we propose instead to identify the *objectives* of those agents. We then consider only the behaviors of the environment that concur to those objectives, instead of all behaviors described by some model of the system. We study the problem of *rational verification*: the system needs to be proven correct with regard to property φ , not in all the executions of the environment, but only in those executions that are *rational* with regard to the objectives of the environment.

There are several ways to model rationality. For instance, a famous model of rational behavior for the agents is the concept of *Nash equilibrium* (NE) [39]. Some promising exploratory works, based on the concept of NE, exist in the literature, like in verification of non-repudiation and fair exchange protocols [35, 23], planning of self-driving cars interacting with human drivers [45], or the automatic verification of an LTL specification in multi-agent systems that behave according to an NE [32]. Another classical approach is to model the environment as a single agent with multiple objectives. In that setting, trade-offs between (partially) conflicting objectives need to be made, and a rational agent will behave in a way to satisfy a *Pareto-optimal* set of its objectives. Pareto-optimality and multi-objective formalisms have been considered in computer science, see for instance [41] and references therein, and in formal methods, see e.g. [2, 12].

Nevertheless, we have only scratched the surface and there is a lack of a general theoretical background for marrying concepts from game theory and formal verification. This is the motivation of our work. We consider the setting in which a designer specifies the behavior of a system and identifies its objective Ω_0 as well as the multiple objectives $\{\Omega_1, \dots, \Omega_t\}$ of the environment in a underlying game arena G . The behavior of the system is usually modeled by the designer using a *deterministic Moore machine* that describes the strategy of the system opposite the environment. The designer can also use the model of *nondeterministic Moore machine* in order to describe a set of multiple possible strategies for the system instead of some single specific strategy. Given a strategy σ_0 for the system, the environment being rational only executes behaviors induced by σ_0 which result in a Pareto-optimal payoff with regard to its set of objectives $\{\Omega_1, \dots, \Omega_t\}$. When the Moore machine \mathcal{M} is deterministic, the *Pareto-rational verification* (PRV) problem asks whether all behaviors that are induced by the machine \mathcal{M} in G and that are Pareto-optimal for the environment all satisfy the objective Ω_0 of the system (a toy example giving intuition on this problem is proposed in the full version). When the Moore machine is nondeterministic, the *universal PRV* problem asks whether for all strategies σ_0 of the system described by \mathcal{M} , all behaviors induced by σ_0 that are Pareto-optimal for the environment satisfy Ω_0 . The latter problem is a clear generalization of the former and is conceptually more challenging, as it asks to verify the correctness of the possibly infinite set of strategies described by \mathcal{M} . The universal PRV problem is also a well motivated problem, as typically, in the early stages of a development cycle, not all implementation details are fixed, and the use of nondeterminism is prevailing. In this last setting, we want to guarantee that a positive verification result is transferred to all possible implementations of the nondeterministic model of the system.

Technical Contributions. We introduce the Pareto-rational verification (PRV) problem and its universal variant. The objective Ω_0 of the system and the set $\{\Omega_1, \dots, \Omega_t\}$ of objectives of the environment are ω -regular objectives. We consider several ways of specifying these

■ **Table 1** Summary of complexity results for the PRV problem and UPRV problem.

Objective	PRV problem complexity	UPRV problem complexity
Parity	co-NP-complete (Theorem 5)	PSPACE, NP-hard, co-NP-hard (Theorem 10)
Boolean Büchi	Π_2 P-complete (Theorem 5)	PSPACE-complete (Theorem 10)
LTL	PSPACE-complete (Theorem 15)	2EXPTIME-complete (Theorem 14)

objectives: either by using parity conditions (a canonical way to specify ω -regular objectives), Boolean Büchi conditions (a generic way to specify Büchi, co-Büchi, Streett, Rabin, and other objectives), or using LTL formulas. Our technical results, some of which are summarized in Table 1, are as follows.

First, we study the complexity class of the PRV problem. We prove that it is co-NP-complete for parity objectives, Π_2 P-complete for Boolean Büchi objectives, and PSPACE-complete for LTL objectives.

Second, we consider the universal variant of the PRV problem. We prove that it is in PSPACE and both NP-hard and co-NP-hard for parity objectives, PSPACE-complete for Boolean Büchi objectives, and 2EXPTIME-complete for LTL objectives.

Third, we establish the fixed-parameter tractability (FPT) of the universal PRV problem where the parameters are the number t of objectives of the environment as well as the highest priorities used in the parity objectives or the size of the formulas used in the Boolean Büchi objectives. For the particular case of the PRV problem with parity conditions, the parameters reduce to t only. Since this number is expected to be limited in practice, our result is of practical relevance. We additionally provide an alternative, possibly more efficient in practice, FPT algorithm for solving the PRV problem which exploits counterexamples and builds an under-approximation of the set of Pareto-optimal payoffs on demand.

Related Work. The concept of nondeterminism for strategies has been studied in the particular context of two-player zero-sum games where one player is opposed to the other one, under the name of permissive strategy, multi-strategy, or nondeterministic strategy in [5, 9, 10, 38, 44]. Those works concern synthesis and not verification.

Several fundamental results have been obtained on multi-player games played on graphs where the objectives of the players are Boolean or quantitative (see e.g. the book chapter [31] or the surveys [11, 13, 14]). Several notions of rational behavior of the players have been studied such as NEs, subgame perfect equilibria (SPEs) [46], secure equilibria [21], or profiles of admissible strategies [6]. The existing results in the literature are mainly focused on the existence of equilibria or the synthesis of such equilibria when they exist. Multidimensional energy and mean-payoff objectives for two-player games played on graphs have been studied in [20, 49, 50] and the Pareto curve of multidimensional mean-payoff games has been studied in [12]. Two-player games with heterogeneous multidimensional quantitative objectives have been investigated in [16].

Recent results concern the synthesis of strategies for a system in a way to satisfy its objective when facing an environment that is assumed to behave rationally with respect to the objectives of all his components. In [29, 36, 37], the objectives are expressed as LTL formulas and the considered models of rationality are NEs or SPEs. Algorithmic questions about this approach are studied in [24] for different types of ω -regular objectives. In [18], the objectives are ω -regular and the environment is assumed to behave rationally by playing in a way to obtain Pareto-optimal payoffs with respect to its objectives. We consider the concepts of [18] as a foundation for Pareto-rational verification.

The previously mentioned results all deal with the existence or the synthesis of solutions. Rational verification (instead of synthesis) is studied in [32] (see also the survey [1]), where the authors study how to verify a given specification for a multi-agent system with agents that behave rationally according to an NE when all objectives are specified by LTL formulas. They prove that this problem is 2EXPTIME-complete and design an algorithm that reduces this problem to solving a collection of parity games. This approach is implemented in the Equilibrium Verification Environment tool. In this paper, we study Pareto-optimality as a model of rationality instead of the concepts of NE or SPE. Our framework is more tractable as the PRV problem is PSPACE-complete for LTL specifications.

2 Definitions and the Pareto-Rational Verification Problem

We start by recalling several classical concepts of game theory, and in particular the model of (nondeterministic) Moore machines. We then present the verification problem studied in this paper and illustrate it on an example. We end the section by discussing the complexity of useful checks performed in several algorithms throughout this paper.

2.1 Definitions

Game Arena and Plays. A *game arena* is a tuple $G = (V, V_0, V_1, E, v_0)$ where (V, E) is a finite directed graph such that: (i) V is the set of vertices and (V_0, V_1) forms a partition of V where V_0 (resp. V_1) is the set of vertices controlled by Player 0 (resp. Player 1), (ii) $E \subseteq V \times V$ is the set of edges such that each vertex v has at least one successor v' , i.e., $(v, v') \in E$, and (iii) $v_0 \in V$ is the initial vertex. We denote by $|G|$ the size of G . A *sub-arena* G' with a set $V' \subseteq V$ of vertices and initial vertex $v'_0 \in V'$ is a game arena defined from G as expected. A *single-player* game arena is a game arena where $V_0 = \emptyset$ and $V_1 = V$.

A *play* in a game arena G is an infinite sequence of vertices $\rho = v_0 v_1 \dots \in V^\omega$ such that it starts with the initial vertex v_0 and $(v_j, v_{j+1}) \in E$ for all $j \in \mathbb{N}$. *Histories* in G are finite non-empty sequences $h = v_0 \dots v_j \in V^+$ defined similarly. The set of plays in G is denoted by Plays_G and the set of histories (resp. histories ending with a vertex in V_i) is denoted by Hist_G (resp. $\text{Hist}_{G,i}$). Notations Plays , Hist , and Hist_i are used when G is clear from the context. The set of vertices occurring (resp. occurring infinitely often) in a play ρ is written $\text{Occ}(\rho)$ (resp. $\text{Inf}(\rho)$).

Strategies and Moore Machines. A *strategy* σ_i for Player i is a function $\sigma_i: \text{Hist}_i \rightarrow V$ assigning to each history $hv \in \text{Hist}_i$ a vertex $v' = \sigma_i(hv)$ such that $(v, v') \in E$. A play $\rho = v_0 v_1 \dots$ is *consistent* with σ_i if $v_{j+1} = \sigma_i(v_0 \dots v_j)$ for all $j \in \mathbb{N}$ such that $v_j \in V_i$. Consistency is naturally extended to histories. The set of plays (resp. histories) consistent with strategy σ_i is written Plays_{σ_i} (resp. Hist_{σ_i}).

A strategy σ_i for Player i is *finite-memory* [30] if it can be encoded by a *deterministic Moore machine* $\mathcal{M} = (M, m_0, \alpha_U, \alpha_N)$ where M is the finite set of states (the memory of the strategy), $m_0 \in M$ is the initial memory state, $\alpha_U: M \times V \rightarrow M$ is the update function, and $\alpha_N: M \times V_i \rightarrow V$ is the next-move function. Such a machine defines the strategy σ_i such that $\sigma_i(hv) = \alpha_N(\hat{\alpha}_U(m_0, h), v)$ for all histories $hv \in \text{Hist}_i$, where $\hat{\alpha}_U$ extends α_U to histories as expected. In this paper, we consider the broader notion of *nondeterministic* Moore machine \mathcal{M} (see e.g. [5]) with a next-move function $\alpha_N: M \times V_i \rightarrow 2^V$. Such a machine embeds a (possibly infinite) set of strategies σ_i for Player i such that $\sigma_i(hv) \in \alpha_N(\hat{\alpha}_U(m_0, h), v)$ for all histories $hv \in \text{Hist}_i$ ¹. We denote by $\llbracket \mathcal{M} \rrbracket$ the set of all strategies defined by \mathcal{M} . The *size* of \mathcal{M} is equal to the number $|M|$ of its memory states. Example 1 illustrates these concepts.

¹ Notice that this definition is different from simply making the machine deterministic by fixing a single next vertex $v' \in \alpha_N(m, v)$ for each $m \in M$ and $v \in V_i$.

When \mathcal{M} is a deterministic Moore machine with $|M| = 1$, then it defines a *memoryless* strategy σ_i where $\sigma_i(hv) = \sigma_i(h'v)$ for all $hv, h'v$ ending with the same vertex $v \in V_i$. When \mathcal{M} is a nondeterministic Moore machine with $|M| = 1$ and such that $\alpha_N(m_0, v) = \{v' \mid (v, v') \in E\}$, then $\llbracket \mathcal{M} \rrbracket$ is exactly the set of *all possible strategies* for Player i .

Objectives. An *objective* for Player i is a set of plays $\Omega \subseteq \text{Plays}$. A play ρ *satisfies* the objective Ω if $\rho \in \Omega$. The *opposite objective* of Ω is written $\bar{\Omega} = \text{Plays} \setminus \Omega$. We consider the following objectives in this paper:

- Let $c : V \rightarrow \{0, \dots, d\}$ be a function called a *priority function* which assigns an integer to each vertex in the arena (we assume that d is even). The set of priorities occurring infinitely often in a play ρ is $\text{Inf}(c(\rho)) = \{c(v) \mid v \in \text{Inf}(\rho)\}$. The *parity* objective $\text{Parity}(c) = \{\rho \in \text{Plays} \mid \min(\text{Inf}(c(\rho))) \text{ is even}\}$ asks that the minimum priority visited infinitely often be even. The opposite objective $\bar{\Omega}$ of a parity objective Ω is again a parity objective (the priority function c' of $\bar{\Omega}$ is such that $c'(v) = c(v) + 1$ for all $v \in V$).
- Given m sets T_1, \dots, T_m such that $T_i \subseteq V$, $i \in \{1, \dots, m\}$, and ϕ a Boolean formula over the set of variables $X = \{x_1, \dots, x_m\}$, the *Boolean Büchi*² [27, 17] objective $\text{BooleanBüchi}(\phi, T_1, \dots, T_m) = \{\rho \in \text{Plays} \mid \rho \text{ satisfies } (\phi, T_1, \dots, T_m)\}$ is the set of plays whose valuation of the variables in X satisfy formula ϕ . Given a play ρ , its valuation is such that $x_i = 1$ if and only if $\text{Inf}(\rho) \cap T_i \neq \emptyset$ and $x_i = 0$ otherwise. That is, a play satisfies the objective if the Boolean formula describing the sets to be visited (in)finitely often by a play is satisfied. It is assumed that negations only appear in literals of ϕ and we denote by $|\phi|$ the size of ϕ equal to the number of symbols in $\{\wedge, \vee, \neg\} \cup X$ in ϕ . The opposite objective $\bar{\Omega}$ of a Boolean Büchi objective Ω is again a Boolean Büchi objective (the formula $\neg\phi$ of $\bar{\Omega}$ is obtained from ϕ by replacing each symbol \vee (resp. \wedge) by \wedge (resp. \vee) and each literal by its negation).

We recall that parity and Boolean Büchi objectives Ω are *prefix-independent*, i.e., whenever $\rho \in \Omega$, then all suffixes of ρ also satisfy Ω .

Zero-Sum Games. A two-player *zero-sum game* $\mathcal{G} = (G, \Omega)$ is a game on a game arena G where Player 0 has objective Ω and Player 1 has the opposite objective $\bar{\Omega}$. Given an initial vertex v_0 , we say that a player is *winning from* v_0 if he has a strategy such that all plays starting with v_0 and consistent with this strategy satisfy his objective. We assume that the reader is familiar with this concept, see e.g. [30].

Lattices and Antichains. A *complete lattice* is a partially ordered set (S, \leq) where S is a set, $\leq \subseteq S \times S$ is a partial order on S , and for every pair of elements $s, s' \in S$, their greatest lower bound and their least upper bound both exist. A subset $A \subseteq S$ is an *antichain* if all of its elements are pairwise incomparable with respect to \leq . Given $T \subseteq S$ and an antichain $A \subseteq S$, we denote by $\lceil T \rceil$ the set of maximal elements of T (which is thus an antichain) and by $\downarrow^< A$ the set of all elements $s \in S$ for which there exists some $s' \in A$ such that $s < s'$. Given two antichains $A, A' \subseteq S$, we write $A \sqsubseteq A'$ when for all $s \in A$, there exists $s' \in A'$ such that $s \leq s'$, and we write $A \sqsubset A'$ when $A \sqsubseteq A'$ and $A \neq A'$.

2.2 Pareto-Rational Verification Problem

We start by recalling the class of two-player games considered in this paper and the notion of payoffs in those games.

² This objective is also called *Emerson-Lei* objective.

Stackelberg-Pareto Games. A *Stackelberg-Pareto game* (SP game) $\mathcal{G} = (G, \Omega_0, \Omega_1, \dots, \Omega_t)$ is composed of a game arena G , an objective Ω_0 for Player 0, and $t \geq 1$ objectives $\Omega_1, \dots, \Omega_t$ for Player 1 [18]. An SP game where all objectives are parity (resp. Boolean Büchi) objectives is called a *parity* (resp. *Boolean Büchi*) *SP game*.

Payoffs. The *payoff* of a play $\rho \in \text{Plays}$ is the vector of Booleans $\text{pay}(\rho) \in \{0, 1\}^t$ such that for all $i \in \{1, \dots, t\}$, $\text{pay}_i(\rho) = 1$ if $\rho \in \Omega_i$, and $\text{pay}_i(\rho) = 0$ otherwise. Notice that we omit to include the objective of Player 0 when discussing the payoff of a play. Instead we say that a play ρ is *won* by Player 0 if $\rho \in \Omega_0$ and we write $\text{won}(\rho) = 1$, otherwise it is *lost* by Player 0 and we write $\text{won}(\rho) = 0$. We write $(\text{won}(\rho), \text{pay}(\rho))$ for the *extended payoff* of ρ . A payoff p (resp. extended payoff (w, p)) is *realizable* if there exists a play $\rho \in \text{Plays}$ such that $\text{pay}(\rho) = p$ (resp. $(\text{won}(\rho), \text{pay}(\rho)) = (w, p)$); we say that ρ *realizes* p (resp. (w, p)).

We consider the following partial order on payoffs. Given two payoffs $p = (p_1, \dots, p_t)$ and $p' = (p'_1, \dots, p'_t)$ such that $p, p' \in \{0, 1\}^t$, we say that p' is *larger* than p and write $p \leq p'$ if $p_i \leq p'_i$ for all $i \in \{1, \dots, t\}$. Moreover, when it also holds that $p_i < p'_i$ for some i , we say that p' is *strictly larger* than p and we write $p < p'$. Notice that the pair $(\{0, 1\}^t, \leq)$ is a complete lattice with size 2^t and that the size of any antichain on $(\{0, 1\}^t, \leq)$ is thus upper bounded by 2^t .

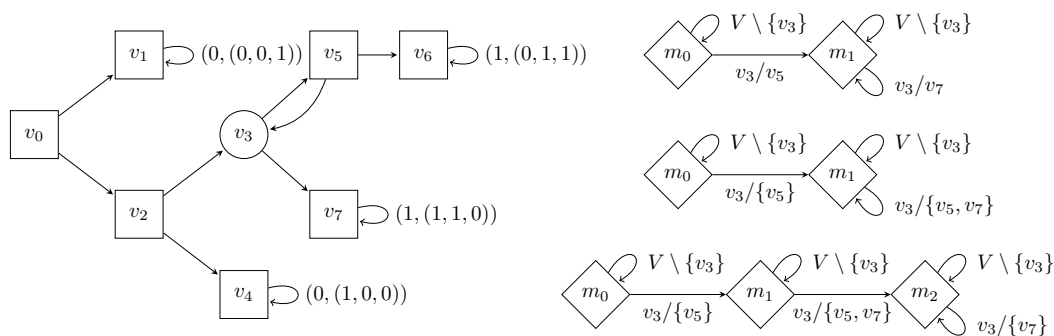
Let $\mathcal{G} = (G, \Omega_0, \Omega_1, \dots, \Omega_t)$ be an SP game and let σ_0 be a strategy of Player 0. We can consider the set of payoffs of plays consistent with σ_0 which are *Pareto-optimal*, i.e., maximal with respect to \leq . We write this set $P_{\sigma_0} = \max\{\text{pay}(\rho) \mid \rho \in \text{Plays}_{\sigma_0}\}$. Notice that this set is an antichain. In this paper, we study the following *verification* problem.

► **Problem.** Let \mathcal{G} be an SP game and let \mathcal{M} be a nondeterministic Moore machine for Player 0. The *universal Pareto-rational verification problem* (UPRV problem) is to decide whether for all $\sigma_0 \in \llbracket \mathcal{M} \rrbracket$, it holds that every play $\rho \in \text{Plays}_{\sigma_0}$ such that $\text{pay}(\rho) \in P_{\sigma_0}$ satisfies the objective of Player 0. When \mathcal{M} is deterministic, we consider the single strategy $\sigma_0 \in \llbracket \mathcal{M} \rrbracket$ and speak about the *Pareto-rational verification problem* (PRV problem).

The UPRV problem models the situation where the system may employ one of several possible strategies in a nondeterministic manner and we therefore want to verify that all of them are correct. We do so in the context where the environment is rational and only executes behaviors which result in a Pareto-optimal payoff with regard to its set of objectives. In the following sections, we study the complexity of the (U)PRV problem in terms of $|G|$ the size of the game arena, $|M|$ the size of the Moore machine, t the number of objectives of Player 1, $\max d_i$ the maximum of all maximum priorities d_i according to each parity objective Ω_i in case of parity SP games, and $\max |\phi_i|$ the maximum of all sizes $|\phi_i|$ such that ϕ_i is the formula for objective Ω_i in case of Boolean Büchi SP games.

► **Example 1.** Consider the parity SP game \mathcal{G} with arena G depicted in Figure 1 (left) in which Player 1 has $t = 3$ objectives [18]. The vertices of Player 0 (resp. Player 1) are depicted as circles (resp. squares)³. We do not explicitly define the parity objective Ω_0 of Player 0 nor the three parity objectives of Player 1. Instead, the extended payoff of plays reaching vertices from which they can only loop is displayed in the arena next to those vertices, and we set the extended payoff of play $v_0v_2(v_3v_5)^\omega$ to $(0, (0, 1, 0))$.

³ This convention is used throughout this paper.



■ **Figure 1** A parity SP game (left), one deterministic Moore machine \mathcal{M}_t and two nondeterministic Moore machines \mathcal{M}_c and \mathcal{M}_b (respectively top, center, and bottom right).

Consider the memoryless strategy σ_0 of Player 0 such that he chooses to always move to v_5 from v_3 . The set of payoffs of plays consistent with σ_0 is $\{(0, 0, 1), (0, 1, 0), (1, 0, 0), (0, 1, 1)\}$ and the set of those that are Pareto-optimal is $P_{\sigma_0} = \{(1, 0, 0), (0, 1, 1)\}$. Notice that play $\rho = v_0 v_2 v_4^\omega$ is consistent with σ_0 , has payoff $(1, 0, 0) \in P_{\sigma_0}$ and is lost by Player 0. Together with \mathcal{G} , strategy σ_0 is therefore a negative instance of the PRV problem.

Let us now consider the finite-memory strategy σ'_0 such that $\sigma'_0(v_0 v_2 v_3) = v_5$ and $\sigma'_0(v_0 v_2 v_3 v_5 v_3) = v_7$. Contrarily to the previous strategy, \mathcal{G} and σ'_0 constitute a positive instance of the PRV problem. Indeed, the set of Pareto-optimal payoffs is $P_{\sigma'_0} = \{(0, 1, 1), (1, 1, 0)\}$ and Player 0 wins every play consistent with σ'_0 whose payoff is in this set. A deterministic Moore machine \mathcal{M}_t for σ'_0 is depicted in Figure 1 (top right). It has two memory states with state m_1 indicating that v_3 has been visited. Each edge from m to m' is labeled by v/v' with an optional v' such that $\alpha_U(m, v) = m'$ and $\alpha_N(m, v) = v'$ if $v \in V_0$.

Finally, we provide two nondeterministic Moore machines in Figure 1 (center right and bottom right). Each edge from m to m' is now labeled by v/T such that $\alpha_N(m, v) = T \subseteq V$ when $v \in V_0$. Let us show that the SP game \mathcal{G} with machine \mathcal{M}_c (resp. machine \mathcal{M}_b) is a negative (resp. positive) instance of the UPRV problem.

One can check that the memoryless strategy σ_0 mentioned above (always move to v_5 from v_3) belongs to the set $\llbracket \mathcal{M}_c \rrbracket$. It follows that \mathcal{G} and \mathcal{M}_c are a negative instance of the UPRV problem. Notice that all the other strategies σ_0^k , $k \geq 1$, of $\llbracket \mathcal{M}_c \rrbracket$ are such that $\sigma_0^k(hv_3) = v_5$ except when $h = v_0 v_2 (v_3 v_5)^k$ in which case $\sigma_0^k(hv_3) = v_7$ (the strategy allows to cycle between v_3 and v_5 k times before dictating that v_7 be visited).

The machine \mathcal{M}_b has three memory states such that m_1 (resp. m_2) records one visit (resp. at least two visits) to v_3 . The set $\llbracket \mathcal{M}_b \rrbracket$ contains exactly two strategies: one is the finite-memory strategy σ'_0 given before and the other one is the strategy σ''_0 such that $\sigma''_0(v_0 v_2 v_3) = \sigma''_0(v_0 v_2 v_3 v_5 v_3) = v_5$ and $\sigma''_0(v_0 v_2 v_3 (v_5 v_3)^2) = v_7$. One can verify that \mathcal{G} and \mathcal{M}_b are a positive instance of the UPRV problem. \lrcorner

► **Remark 2.** In the sequel, we often consider the Cartesian product $G \times \mathcal{M}$ with initial vertex (v_0, m_0) of the arena G of \mathcal{G} with the (nondeterministic) Moore machine \mathcal{M} for Player 0. When \mathcal{M} is nondeterministic, this finite graph $G \times \mathcal{M}$ is a two-player game arena (as the vertices of Player 0 may have several successors). The strategies σ'_0 for Player 0 in this product correspond exactly to the strategies $\sigma_0 \in \llbracket \mathcal{M} \rrbracket$. With this in mind, we can reformulate the UPRV problem to take a game arena as input. Given $G' = G \times \mathcal{M}$, the UPRV problem is to decide whether for all strategies σ'_0 of Player 0 in G' , every play $\rho \in \text{Plays}_{\sigma'_0}$ such that $\text{pay}(\rho) \in P_{\sigma'_0}$ satisfies the objective of Player 0. When \mathcal{M} is deterministic, this

product is a finite graph whose infinite paths, starting from the initial vertex, are exactly the plays consistent with the single strategy $\sigma_0 \in \llbracket \mathcal{M} \rrbracket$. This graph can be seen as a single-player game arena G' (as every vertex of Player 0 only has a single successor). In that setting, given a single-player arena $G' = G \times \mathcal{M}$, the PRV problem is to decide whether every play $\rho \in \text{Plays}_{G'}$ such that $\text{pay}(\rho) \in \max\{\text{pay}(\rho) \mid \rho \in \text{Plays}_{G'}\}$ satisfies the objective of Player 0.

Payoff Realizability and Lassoes. In order to study the (U)PRV problem, we need to perform specific checks on payoffs as described in the next proposition (the proof of which can be found in the full version).

► **Proposition 3.** *Let $\mathcal{G} = (G, \Omega_1, \dots, \Omega_t)$ be an SP game and let p (resp. (w, p)) be a payoff (resp. extended payoff). The existence of a play ρ realizing payoff p (resp. extended payoff (w, p)) can be decided with the following complexities.*

- *For parity objectives: in time polynomial in $|G|$, t , and $\max d_i$.*
 - *For Boolean Büchi objectives: in time polynomial in $|G|$, and exponential in t and $\max |\phi_i|$.*
- Checking whether a realizable payoff p is Pareto-optimal is decided with the same complexities.*

We also need the next property which shows that when a play satisfies a parity or a Boolean Büchi objective, there exists another such play that is a lasso of polynomial size.

► **Lemma 4** ([8]). *For any play $\rho \in \text{Plays}$, there exists a lasso $\rho' = gh^\omega$ such that ρ and ρ' start with the same vertex, $\text{Occ}(\rho) = \text{Occ}(\rho')$, $\text{Inf}(\rho) = \text{Inf}(\rho')$, and $|gh|$ is quadratic in $|G|$.*

Related Synthesis Problem. Our verification problem is related to the *Stackelberg-Pareto Synthesis problem* introduced in [18]. This synthesis problem asks, given a two-player SP game, whether there exists a strategy σ_0 for Player 0 such that every play in Plays_{σ_0} with a Pareto-optimal payoff satisfies the objective of Player 0. This problem is solved in [18] for parity and reachability objectives. It is shown that the problem is NEXPTIME-complete, and that finite-memory strategies are sufficient for Player 0 to have a solution σ_0 to the problem.

3 Complexity Class of the PRV problem

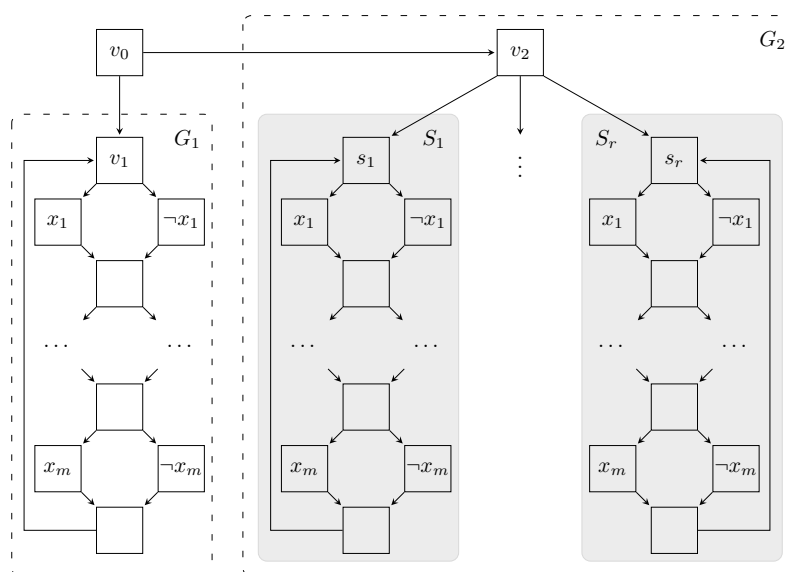
In this section, we provide the complexity class of the PRV problem for both parity SP games and Boolean Büchi SP games. The complexity class of the UPRV problem is studied in Section 4. In this whole section, we assume that an instance of the PRV problem is an SP game with a *single-player* game arena (see Remark 2). This is not problematic with respect to the algorithmic complexities since the size of the single-player game arena is $|G| \cdot |M|$.

► **Theorem 5.** *The PRV problem is co-NP-complete for parity SP games and $\Pi_2\text{P}$ -complete for Boolean Büchi SP games.*

We now detail the arguments used to show the co-NP-completeness for parity SP games, and refer the reader to the full version for the completeness result for Boolean Büchi SP games.

Membership to co-NP. The co-NP-membership stated in Theorem 5 is easily proved by showing that the complement of the PRV problem is in NP. Given a single-player SP game \mathcal{G} , we guess a payoff $p \in \{0, 1\}^t$, and we check (i) whether p is realizable and Pareto-optimal, and (ii) whether there exists a play ρ with payoff p which is lost by Player 0. In the case of parity objectives, those two checks can be performed in polynomial time by Proposition 3.

The proof of co-NP-hardness is more involved. In order to show this result, we provide a reduction from the co-3SAT problem to the PRV problem.



■ **Figure 2** The single-player arena G used in the reduction from co-3SAT for parity objectives.

The co-3SAT Problem. We consider a formula $\psi = D_1 \wedge \dots \wedge D_r$ in 3-Conjunctive Normal Form (3CNF) consisting of r clauses, each containing exactly 3 literals over the set of variables $X = \{x_1, \dots, x_m\}$. We assume that each variable x occurs as a literal $\ell \in \{x, \neg x\}$ in at least one clause of ψ . The satisfiability problem, called 3SAT, is to decide whether there exists a valuation of the variables in X such that the formula ψ evaluates to true. This problem is well-known to be NP-complete [25, 34]. We can consider the complement of this problem, which is to decide for such a formula ψ whether *all valuations* of the variables in X falsify the formula i.e., make at least one of the clauses evaluate to false. This problem, called co-3SAT, being the complement of an NP-complete problem, is co-NP-complete [40].

Intuition of the Reduction. Given an instance of co-3SAT, we create a parity SP game \mathcal{G} with a single-player game arena G consisting of two sub-arenas G_1 and G_2 reachable from the initial vertex v_0 as depicted in Figure 2. The intuition behind this construction is the following. A play in the arena starts in v_0 and will either enter G_1 through v_1 and stay in that sub-arena forever or enter G_2 through v_2 , visit some vertex s_i with $i \in \{1, \dots, r\}$, and stay forever in the corresponding sub-arena S_i . The objectives are devised such that a payoff contains one objective per literal of X and one objective per literal, per clause of ψ . A play in G_1 has a payoff corresponding to a valuation of X and the literals in the clauses of ψ satisfied by that valuation. In addition, the objective of Player 0 is not satisfied in those plays. Therefore, it must be the case that the payoffs of plays in G_1 are not Pareto-optimal in order for the instance of the PRV problem to be positive. This is only the case when the instance of co-3SAT is also positive due to the fact that plays in G_2 , which all satisfy the objective of Player 0, then have payoffs strictly larger than that of plays in G_1 . This is not the case if some play in G_1 corresponds to a valuation of X which satisfies ψ .

Structure of a Payoff. We now detail the objectives used in the reduction and the corresponding structure of a payoff in G . Player 0 has a single parity objective Ω_0 . Player 1 has $1 + 2 \cdot m + 3 \cdot r$ parity objectives (assuming each clause is composed of exactly 3 literals). The payoff of a play in G therefore consists in a vector of $1 + 2 \cdot m + 3 \cdot r$ Booleans for the following objectives:

$$(\Omega_1, \Omega_{x_1}, \Omega_{\neg x_1}, \dots, \Omega_{x_m}, \Omega_{\neg x_m}, \Omega_{\ell^{1,1}}, \Omega_{\ell^{1,2}}, \Omega_{\ell^{1,3}}, \dots, \Omega_{\ell^{r,1}}, \Omega_{\ell^{r,2}}, \Omega_{\ell^{r,3}}).$$

The objective Ω_0 is equal to objective $\Omega_1 = \text{Parity}(c)$ with $c(v) = 2$ if $v \in G_2$ and $c(v) = 1$ otherwise. It is direct to see that these objectives are only satisfied for plays in G_2 . We define the objective $\Omega_x = \text{Parity}(c)$ (resp. $\Omega_{\neg x} = \text{Parity}(c')$) with $c(x) = 2$ and $c(\neg x) = 1$ (resp. $c'(\neg x) = 2$ and $c'(x) = 1$) for the vertices labelled x and $\neg x$ in G_1 and G_2 , and such that every other vertex has priority 2 according to c (resp. c'). Objective Ω_x (resp. $\Omega_{\neg x}$) is satisfied if and only if vertex x (resp. $\neg x$) is visited infinitely often and $\neg x$ (resp. x) is not. If both x and $\neg x$ are visited infinitely often, neither Ω_x nor $\Omega_{\neg x}$ are satisfied. These objectives are used to encode valuations of X into payoffs. The objective $\Omega_{\ell^{i,j}}$ corresponds to the objective for the j^{th} literal of the i^{th} clause of ψ , written $\ell^{i,j} \in \{x_k, \neg x_k\}$ for some $k \in \{1, \dots, m\}$, we define the priority function for this objective later for each sub-arena.

Payoff of Plays Entering Sub-Arena G_1 . We define the priority function c of objective $\Omega_{\ell^{i,j}}$ in G_1 such that $c(\ell^{i,j}) = 2$ and $c(\neg \ell^{i,j}) = 1$ for vertices labeled $\ell^{i,j}$ and $\neg \ell^{i,j}$ in G_1 . Notice that a play in G_1 corresponds to repeatedly making the choice of visiting x_i or $\neg x_i$ for $i \in \{1, \dots, m\}$. We call plays which visit both x_i and $\neg x_i$ infinitely often for some i *unstable* plays and those which visit infinitely often either x_i or $\neg x_i$ for each i *stable* plays. We introduce the following lemma on the stability of plays in G_1 (proved in the full version).

► **Lemma 6.** *Unstable plays in G_1 do not have a Pareto-optimal payoff.*

In the sequel, we therefore only consider stable plays ρ in G_1 . The objective Ω_0 of Player 0 and Ω_1 of Player 1 are not satisfied in ρ and such a play satisfies either the objective Ω_{x_i} or $\Omega_{\neg x_i}$ for each $x_i \in X$. The part of the payoff of ρ for these objectives can be seen as a valuation of the variables in X , expressed as a vector of $2 \cdot m$ Booleans. The objective $\Omega_{\ell^{i,j}}$ is satisfied in the payoff of ρ if and only if the literal $\ell^{i,j}$ is satisfied by that valuation. That is if either $\ell^{i,j} = x_k$ and Ω_{x_k} is satisfied or $\ell^{i,j} = \neg x_k$ and $\Omega_{\neg x_k}$ is satisfied, for $x_k \in X$. Given a positive instance of the co-3SAT problem, it holds that none of the valuations of X satisfy the formula ψ . Therefore, since stable plays in G_1 encode valuations of X and the corresponding satisfied literals of the clauses of ψ , the next lemma holds (see full version).

► **Lemma 7.** *Given a positive instance of the co-3SAT problem and any stable play ρ in G_1 , there exists a clause D_i for $i \in \{1, \dots, r\}$ such that $\Omega_{\ell^{i,j}}$ is not satisfied in ρ for $j \in \{1, 2, 3\}$.*

In order for the instance of the PRV problem to be positive in case of a positive instance of co-3SAT, since plays in G_1 do not satisfy the objective of Player 0, it must be the case that the payoff of these plays are not Pareto-optimal when considering the whole arena G . Therefore, given any play in G_1 , there must exist a play with a strictly larger payoff in G_2 which also satisfies the objective of Player 0.

Payoff of Plays Entering Sub-Arena G_2 . We define the priority function c of objective $\Omega_{\ell^{i,j}}$ in G_2 such that $c(s_i) = 1$ and $c(v) = 2$ for $v \neq s_i$ in G_2 . Therefore, any play entering S_i satisfies every objective for the literals of the clauses of ψ , except for objectives $\Omega_{\ell^{i,j}}$, $j \in \{1, 2, 3\}$. After entering a sub-arena S_j , plays in G_2 can visit infinitely often either or both x_i and $\neg x_i$ for $i \in \{1, \dots, m\}$ and we therefore introduce the following lemma on the stability of plays in G_2 , the proof of which is given in the full version.

► **Lemma 8.** *Unstable plays in G_2 do not have a Pareto-optimal payoff.*

We therefore only consider stable plays in G_2 . Such a play ρ satisfies either the objective Ω_{x_i} or $\Omega_{\neg x_i}$ for each $x_i \in X$. The objectives corresponding to the literals in the clauses of ψ which are satisfied in ρ only depend on the sub-arena S_j entered by ρ . It can easily be shown that every such objective is satisfied by ρ except for $\Omega_{\ell^j,1}$, $\Omega_{\ell^j,2}$ and $\Omega_{\ell^j,3}$ for clause D_j .

Correctness. Finally, we briefly discuss the correctness of this reduction (a full proof is provided in the full version). In case of a positive instance of the co-3SAT problem, for every valuation of X (and therefore every stable play ρ in G_1), this valuation does not satisfy some clause D_i of ϕ (and therefore ρ does not satisfy any objective $\Omega_{\ell^i,j}$ by Lemma 7). It follows that there exists a play with a strictly larger payoff in G_2 given the form of the payoff of plays in G_2 discussed above (and the fact that they satisfy objective Ω_1 while ρ does not). In case of a negative instance of co-3SAT, this is not the case as some stable play in G_1 corresponds to a valuation which satisfies ϕ and therefore satisfies at least one objective for each clause D_i . As plays in G_2 do not satisfy any objective for some clause, G_1 contains a Pareto-optimal play lost by Player 0, and the instance of the PRV problem is negative.

► **Remark 9.** As stated in Theorem 5, the lower bound for the PRV problem is stronger for Boolean Büchi objectives than for parity objectives. We can show that this difference in complexity is even more apparent if we consider the following variant of the *complement* of the PRV problem in which we *fix a payoff* for Player 1: given a single-player SP game and a payoff p , decide whether there exists a play with payoff p not satisfying Ω_0 and p is Pareto-optimal. While this problem is in P for parity SP games (indeed p does not need to be guessed anymore), it is BH_2 -complete for Boolean Büchi SP games. We refer the reader to the full version for details about this additional complexity result.

4 Complexity Class of the UPRV problem

We study in this section the complexity class of the UPRV problem for parity and Boolean Büchi SP games. Our results are summarized in the following theorem.

- **Theorem 10.** *The UPRV problem is*
- PSPACE-complete for Boolean Büchi SP games,
 - in PSPACE, NP-hard and co-NP-hard for parity SP games.

We show the PSPACE-membership stated in Theorem 10 in the following proposition.

► **Proposition 11.** *The UPRV problem is in PSPACE for both Boolean Büchi SP games and parity SP games.*

Proof. Let \mathcal{G} be an SP game and \mathcal{M} be a nondeterministic Moore machine for Player 0. By Remark 2, the strategies of $\llbracket \mathcal{M} \rrbracket$ are exactly the strategies of the product $G' = G \times \mathcal{M}$. In the sequel, we will shift from G to G' and conversely without mentioning it explicitly.

To prove Proposition 11, it is enough to show that the complement of the UPRV problem is in NPSpace, since $\text{NPSpace} = \text{PSPACE}$ and as the PSPACE class is closed under complementation. The complement of the UPRV problem is to decide whether there exists a strategy $\sigma_0 \in \llbracket \mathcal{M} \rrbracket$ and a play $\rho \in \text{Plays}_{\sigma_0}$ such that $\text{pay}(\rho) \in P_{\sigma_0}$ and ρ is lost by Player 0.

Our algorithm works as follows in G' (we detail its correctness and complexity later):

1. guess a lasso $\rho' = g'h'^\omega$ in $\text{Plays}_{G'}$ such that $g'h'$ has polynomial size,
2. check that ρ' is lost by Player 0,

33:12 Pareto-Rational Verification

3. check that for each vertex v of ρ' controlled by Player 1, Player 0 is winning from v in the two-player *zero-sum* game $\mathcal{H} = (G', \Omega')$ with arena G' and objective $\Omega' = \{\rho^* \in \text{Plays}_{G'} \mid \neg(\text{pay}(\rho^*) > \text{pay}(\rho'))\}$.

Let us prove that this algorithm is correct. (i) Assume first that there exists a strategy $\sigma_0 \in \llbracket \mathcal{M} \rrbracket$ and a play $\rho \in \text{Plays}_{\sigma_0}$ such that $\text{pay}(\rho) \in P_{\sigma_0}$ and ρ is lost by Player 0. We see this play ρ as a play in G' . By Lemma 4 there exists a lasso $\rho' = g'h'^\omega$ of polynomial size in G' which realises the same extended payoff and such that $\text{Occ}(\rho) = \text{Occ}(\rho')$. This lasso is what is guessed in step 1 of the algorithm. By our assumptions on ρ , we know that it satisfies the check of step 2. It remains to explain why the second check also succeeds in step 3. From each vertex v of ρ' (and thus of ρ) controlled by Player 1, Player 0 is winning in \mathcal{H} thanks to his strategy σ_0 . Indeed, any play $\rho'_1 \in \text{Plays}_{G'}$ consistent with σ_0 cannot have a payoff strictly larger than $\text{pay}(\rho') \in P_{\sigma_0}$, and parity and Boolean Büchi objectives are prefix-independent. (ii) Assume now that the two checks of our algorithm succeed for the guessed lasso ρ' . Let us define a strategy σ_0 for Player 0 in G' (which is also a strategy $\sigma_0 \in \llbracket \mathcal{M} \rrbracket$) as follows: first we define σ_0 in a way to produce play ρ' ; second after each history hvv' such that hv is prefix of ρ' and hvv' is not (meaning that v belongs to Player 1), σ_0 acts as the winning strategy of Player 0 from v in \mathcal{H} . We have thus proved that there exist a strategy $\sigma_0 \in \llbracket \mathcal{M} \rrbracket$ and a play $\rho' \in \text{Plays}_{\sigma_0}$ such that $\text{pay}(\rho') \in P_{\sigma_0}$ and ρ' is lost by Player 0.

Let us now show that our nondeterministic algorithm executes in polynomial space. Step 1 requires polynomial space to store $g'h'$. The check of step 2 requires to verify that $\rho' \notin \Omega_0$ such that Ω_0 is either a parity or a Boolean Büchi objective. This can be done by looking at the cycle h' in polynomial space. Let us now study step 3. We are going to show that $\mathcal{H} = (G', \Omega')$ is a zero-sum game with a Boolean Büchi objective Ω' , known to be solvable in PSPACE [33]. Let us denote by $p = (p_1, \dots, p_t)$ the payoff of ρ' . The objective Ω' is equal to

$$\left(\bigcap_{p_i=0} \bar{\Omega}_i \right) \cup \left(\bigcup_{\substack{p_i=1 \\ p_j=0}} (\bar{\Omega}_i \cap \Omega_j) \right) \quad (1)$$

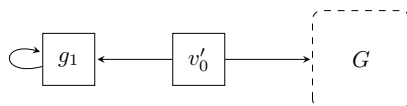
where the first disjunct expresses plays with payoffs less than or equal to p and the second disjunct expresses plays with payoffs incomparable with p . Recall that any parity objective can be expressed as a Boolean Büchi objective using a formula of size $\mathcal{O}(d^2)$ where d is the highest priority in the parity objective (see e.g. [3]). Therefore, for both parity and Boolean Büchi SP games, the objective Ω' is a Boolean Büchi objective defined by a formula of polynomial size. \blacktriangleleft

We now turn to the hardness results stated in Theorem 10. The co-NP hardness of the UPRV problem for parity SP games is easily obtained from the co-NP hardness of the PRV problem (Theorem 5). We consider the other hardness results in the following proposition.

► **Proposition 12.** *The UPRV problem is NP-hard for parity SP games, and PSPACE-hard for Boolean Büchi SP games.*

We prove the NP-hardness for parity SP games and refer the reader to the full version for the PSPACE-hardness for Boolean Büchi SP games. For this purpose, we reduce the following co-NP-hard problem to an instance of the complement of the UPRV problem.

Generalized Parity Game. Let us consider a two-player zero-sum generalized parity game $(G, \Omega_a \wedge \Omega_b)$ where the objective of Player 0 is a conjunction $\Omega_a \wedge \Omega_b$ of two parity objectives. Deciding whether Player 0 has a winning strategy from a vertex v_0 in G is co-NP-hard [22].



■ **Figure 3** The arena G' used in the reduction from zero-sum games with two parity objectives.

Intuition of the Reduction. Given a zero-sum generalized parity game $(G, \Omega_a \wedge \Omega_b)$ and a vertex v_0 , we construct an instance of the UPRV problem with the game arena G' depicted in Figure 3. In G' , the dashed box labeled G represents the arena of the zero-sum game and we assume that the edge from v'_0 goes to v_0 in G . Equivalently, the dashed box is the Cartesian product of G and the nondeterministic machine \mathcal{M} with one memory state embedding all possible strategies of Player 0 (see Remark 2). Notice that given a play ρ' of G' reaching G , we can retrieve a corresponding play ρ from v_0 in G . Any strategy σ_0 of Player 0 in G' is a strategy in $\llbracket \mathcal{M} \rrbracket$ and the converse also holds. We will see that the proposed construction is such that Player 0 has a winning strategy from v_0 in $(G, \Omega_a \wedge \Omega_b)$ if and only if the corresponding instance of the UPRV problem is *negative*.

Objectives. Player 0 has a single parity objective Ω_0 and Player 1 has two parity objectives Ω_1 and Ω_2 . We first extend the priority function c_a of Ω_a (resp. c_b of Ω_b) to G' such that $c_a(g_1) = c_a(v'_0) = c_b(g_1) = c_b(v'_0) = 1$ and consider the corresponding objective Ω'_a (resp. Ω'_b) in G' . Notice that $\Omega'_a = \Omega_a$ (resp. $\Omega'_b = \Omega_b$) when considering only the plays of sub-arena G in G' . We define the actual objectives used in the reduction as follows. Player 0 has objective $\Omega_0 = \text{Parity}(c)$ with a priority function c defined such that Ω_0 is only satisfied in plays reaching G . The first (resp. second) objective of Player 1 is such that $\Omega_1 = \overline{\Omega'_a}$ (resp. $\Omega_2 = \overline{\Omega'_b}$). Notice that objective Ω_1 (resp. Ω_2) is satisfied in plays reaching G if and only if the objective Ω_a (resp. Ω_b) is *not* satisfied in those plays. The play $v'_0 g_1^\omega$ is consistent with any strategy of Player 0 and has extended payoff $(0, (0, 0))$. Any play reaching G is of the form $\rho' = v'_0 \rho$ where ρ is a play in G starting from the initial vertex v_0 . The extended payoff for such a play ρ' is $(1, (0, 0))$ if ρ satisfies Ω_a and Ω_b ; $(1, (0, 1))$ if ρ satisfies Ω_a and not Ω_b ; $(1, (1, 0))$ if ρ satisfies Ω_b and not Ω_a ; and $(1, (1, 1))$ if ρ does not satisfy Ω_a nor Ω_b .

Correctness. If the instance of the UPRV problem is negative, it holds there exists a strategy $\sigma_0 \in \llbracket \mathcal{M} \rrbracket$ such that some play in Plays_{σ_0} has a Pareto-optimal payoff and is lost by Player 0. Since the play $v'_0 g_1^\omega$ with payoff $(0, 0)$ is the only one in G' not to satisfy Ω_0 , its payoff must be Pareto-optimal. It follows that all plays in G that are consistent with σ_0 have payoff $(0, 0)$ and therefore satisfy the conjunction $\Omega_a \wedge \Omega_b$. Hence, σ_0 is a winning strategy for Player 0 from v_0 in the zero-sum game $(G, \Omega_a \wedge \Omega_b)$. Conversely, if Player 0 has a winning strategy from v_0 in $(G, \Omega_a \wedge \Omega_b)$, it holds that this strategy is in $\llbracket \mathcal{M} \rrbracket$ and such that all consistent plays in G satisfy the conjunction $\Omega_a \wedge \Omega_b$ and therefore has payoff $(0, 0)$. It is easily checked that the instance of the UPRV problem is negative.

5 Fixed-Parameter Complexity

In this section, we study the fixed-parameter complexity of the (U)PRV problem. We refer the reader to [26] for the concept of fixed-parameter tractability (FPT). We recall that given an SP game $\mathcal{G} = (G, \Omega_0, \dots, \Omega_t)$, $\max d_i$ is the maximum of all maximum priorities d_i according to each objective Ω_i in case of parity SP games, and that $\max |\phi_i|$ is the maximum of all sizes $|\phi_i|$ such that each ϕ_i defines objective Ω_i in case of Boolean Büchi SP games.

► **Theorem 13.** *The UPRV problem is in FPT*

- *with parameters t and $\max d_i$ for parity SP games (with an exponential in t and $\max d_i$),*
- *with parameters t and $\max |\phi_i|$ for Boolean Büchi SP games (with an exponential in t and $\max |\phi_i|$).*

The proof of this theorem uses a deterministic variant of the algorithm given in the proof of Proposition 11. Instead of guessing a lasso, we loop over each possible payoff p for which we test whether there exists a play ρ with payoff p not satisfying Ω_0 , and such that Player 0 has a winning strategy from each vertex v of ρ in the zero-sum game $\mathcal{H} = (G \times \mathcal{M}, \Omega')$ with Ω' defined in (1). Whether Player 0 is winning from v in \mathcal{H} can be checked with an FPT algorithm with parameter $|\phi'|$ (with an exponential in $|\phi'|$) where ϕ' defines the Boolean Büchi objective Ω' [17, 15].⁴ Details of the proof are given in the full version.

A direct corollary of Theorem 13 is that the PRV problem is also in FPT. Nevertheless, we provide in the full version a simpler FPT algorithm for the PRV problem leading to an improved complexity for parity SP games (with a sole exponential in t). A second, more clever, variation is given in Algorithm 1 where instead of computing the antichain P_{σ_0} by going through the entire lattice of payoffs, we compute an *under-approximation* (with respect to \sqsubseteq) of P_{σ_0} on demand by using *counterexamples*. The algorithm systematically searches for plays ρ losing for Player 0 and maintains an antichain A of realizable payoffs to eliminate previous counterexamples. Initially, this antichain A is empty. A potential counterexample is a play ρ losing for Player 0 and such that for all payoffs p of A , $\text{pay}(\rho)$ is not strictly smaller than p , that is, $\text{pay}(\rho) \notin \downarrow^< A$ (line 3). When a potential counterexample ρ exists, there are two possible cases. First, there exists a play ρ' winning for Player 0 and such that $\text{pay}(\rho') > \text{pay}(\rho)$ (line 4). The payoff of ρ' is added to A and a new approximation A of P_{σ_0} is computed (by keeping only the maximal elements, line 5). Second, if such a play ρ' does not exist, then we have identified a counterexample (the play ρ), showing that the instance of the PRV problem is negative (line 7). If there are no more potential counterexamples, then the instance is positive (line 9), otherwise we iterate. This algorithm is guaranteed to terminate as $A \sqsubset [A \cup \{\text{pay}(\rho')\}]$ in line 5. Algorithm 1 is shown to be correct and in FPT in the full version of the paper, where we also evaluate it to show its efficiency in practice.

■ **Algorithm 1** Counterexample-based algorithm for the PRV problem.

Input: A single-player SP game resulting from the Cartesian product of the arena G of an SP game and a deterministic Moore machine \mathcal{M} for Player 0.

Output: Whether the instance of the PRV problem is positive.

```

1  $A \leftarrow \emptyset$ 
2 repeat
3   if  $\exists \rho \in \text{Plays}$  such that  $\text{won}(\rho) = 0$  and  $\text{pay}(\rho) \notin \downarrow^< A$  then
4     if  $\exists \rho' \in \text{Plays}$  such that  $\text{won}(\rho') = 1$  and  $\text{pay}(\rho') > \text{pay}(\rho)$  then
5        $A \leftarrow [A \cup \{\text{pay}(\rho')\}]$ 
6     else
7       return False
8   else
9     return True

```

⁴ The FPT algorithm in [17] is linear in the number of symbols \vee, \wedge of ϕ' and double exponential in the number of variables of ϕ' . This complexity is improved in [15] by replacing the double exponential in $|\phi'|$ by a single one.

6 LTL Pareto-Rational Verification

We now show that when the objectives are expressed using Linear Temporal Logic (LTL) formulas, the PRV problem retains the PSPACE-completeness of the LTL model-checking problem, and the UPRV problem retains the 2EXPTIME-completeness of solving LTL games. We do not investigate the fixed-parameter complexity in this context as the completeness to PSPACE (resp. 2EXPTIME) already holds when Player 1 has a single objective.

LTL (Universal) Pareto-Rational Verification Problem. A labeled game arena G_λ is a game arena where a labeling function $\lambda : V \rightarrow 2^{AP}$ maps each vertex to a set of propositional variables in AP . An LTL SP game $\mathcal{G} = (G_\lambda, \phi_0, \phi_1, \dots, \phi_t)$ is composed of a labeled game arena G_λ , an LTL formula ϕ_0 for Player 0 and $t \geq 1$ LTL formulas ϕ_1, \dots, ϕ_t for Player 1. The difference with regular SP games is thus that the goal of the players is expressed using LTL formulas over the set of propositional variables AP . The payoff of plays in G_λ is defined as expected. Given an LTL SP game, we consider the two verification problems described in Section 2 and call them the *LTL PRV problem* and *LTL UPRV problem*.

► **Theorem 14.** *The LTL UPRV problem is 2EXPTIME-complete.*

Proof. We first prove that the LTL UPRV problem is in 2EXPTIME. Given an LTL SP game \mathcal{G} and a nondeterministic Moore machine \mathcal{M} , we proceed as follows. We first perform the Cartesian product $G' = G_\lambda \times \mathcal{A}_0 \times \mathcal{A}_1 \times \dots \times \mathcal{A}_t$ of the arena G_λ with a Deterministic Parity Automaton (DPA) \mathcal{A}_i for each LTL formula ϕ_i , $i \in \{0, \dots, t\}$. The size of each automaton is at most double exponential in the size of its corresponding LTL formula, and the number of priorities it uses is exponential [48, 42, 28]. We thus have a parity SP game \mathcal{G}' with arena G' of double exponential size. We then use the FPT algorithm of Theorem 13 on this SP game \mathcal{G}' , which is polynomial in $|G'|$ and exponential in the parameters t and $\max d'_i$ (the maximum priority used in the parity objectives). Therefore this algorithm is polynomial in $|G_\lambda|$, single exponential in t , and double exponential in the size of LTL formulas ϕ_i , $i \in \{0, \dots, t\}$. This shows the 2EXPTIME-easiness.

Let us now prove the 2EXPTIME-hardness result by adapting the reduction of Proposition 12 for the case of the LTL UPRV problem.

- We consider the problem of deciding whether Player 0 has a winning strategy from v_0 in a two-player zero-sum game (G_λ, ϕ) where the ϕ is the LTL objective of Player 0. This problem is 2EXPTIME-complete [43].
- Given such a zero-sum game (G_λ, ϕ) and a vertex v_0 , we construct an instance of the UPRV problem on the same game arena G' depicted in Figure 3. In this arena, G is replaced by G_λ and both v'_0 and g_1 are labelled with the set $\{x\}$ containing the single atomic proposition x which does not appear in ϕ . The nondeterministic machine \mathcal{M} considered in the reduction is again the one with a single memory state that embeds every possible strategy of Player 0. The objective Ω_0 of Player 0 is defined by LTL formula ϕ_0 and the single objective Ω_1 of Player 1 is defined by LTL formula ϕ_1 as follows:

- $\phi_0 = \neg \bigcirc x$,
- $\phi_1 = (\neg \bigcirc x) \wedge (\neg \bigcirc \phi)$

where \bigcirc is the next operator in LTL. It is direct to see that objective Ω_0 is not satisfied by the play $v'_0 g_1^\omega$ and is satisfied by all plays reaching G_λ . The objective Ω_1 is not satisfied by the play $v'_0 g_1^\omega$ and is satisfied by plays reaching G_λ if and only if the formula ϕ is not satisfied in those plays.

- Using similar arguments as used in the proof of Proposition 12, the following holds. A strategy $\sigma_0 \in \llbracket \mathcal{M} \rrbracket$ makes the instance of the LTL UPRV problem negative if every play $v'_0 \rho$ reaching G_λ and consistent with this strategy falsifies objective Ω_1 of Player 1 (as no payoff is then strictly larger than that of play $v'_0 g_1^\omega$, lost by Player 0). If this is the case, it follows that strategy σ_0 is a winning strategy for Player 0 from v_0 in the zero-sum game (G_λ, ϕ) as every play ρ consistent with this strategy satisfies formula ϕ . The converse is also true. Player 0 therefore has a winning strategy from v_0 in (G_λ, ϕ) if and only if the corresponding instance of the LTL UPRV problem is negative. It follows that the LTL UPRV problem is 2EXPTIME-hard for LTL SP games (as $\text{co-2EXPTIME} = 2\text{EXPTIME}$). ◀

► **Theorem 15.** *The LTL PRV problem is PSPACE-complete.*

The proof of this theorem relies on two variants of the LTL model-checking problem that are both PSPACE-complete [47].

LTL Model-Checking Problem. Given a finite transition system T , an initial state, and an LTL formula ψ , the *LTL existential* (resp. *universal*) *model-checking problem* is to decide whether ψ is satisfied in at least one infinite path (resp. all infinite paths) of T starting from the initial state. Notice that a finite transition system is the same model as a single-player labeled game arena and that an infinite path in T corresponds to a play in this arena.

Proof of Theorem 15. We first show that the LTL PRV problem is in PSPACE. Given an LTL SP game \mathcal{G} , we proceed as follows. For each payoff $p \in \{0, 1\}^t$, we check (i) whether it is realizable and Pareto-optimal, if yes (ii) whether there exists a play ρ such that $\text{pay}(\rho) = p$ and $\text{won}(\rho) = 0$. If for some payoff p , both tests succeed, then the given instance \mathcal{G} is negative, otherwise it is positive (this approach is similar to the simpler FPT algorithm for the PRV problem provided in the full version). Checking that a payoff p is realizable reduces to solving the LTL existential model-checking problem for the formula $\psi = (\bigwedge_{p_i=1} \phi_i) \wedge (\bigwedge_{p_i=0} \neg \phi_i)$, this test can be performed in polynomial space. The second check in (i) and the last check in (ii) are similarly executed in polynomial space. The LTL PRV problem is hence in PSPACE.

We now prove that the LTL PRV problem is PSPACE-hard by showing that we can transform any instance of the LTL universal model-checking problem into an instance of the LTL PRV problem such that the instance of the former is positive if and only if the corresponding instance of the latter is positive as well. Let T be transition system and ψ be an LTL formula. Given our previous remark, T can be seen as a single-player labeled arena G_λ for some labeling function λ . We create the following LTL SP game $\mathcal{G} = (G_\lambda, \psi, \phi_1)$ played on $G_\lambda = T$ where the objective of Player 0 is to satisfy the formula ψ and the sole objective of Player 1 is to satisfy the formula $\phi_1 = \text{true}$. It is direct to see that any play in G_λ satisfies the objective of Player 1 and therefore that every play in G_λ is Pareto-optimal. It follows that the given instance of the LTL PRV problem is positive if and only if every play in G_λ satisfies the formula ψ . This corresponds exactly to the LTL universal model-checking problem. ◀

References

- 1 Alessandro Abate, Julian Gutierrez, Lewis Hammond, Paul Harrenstein, Marta Kwiatkowska, Muhammad Najib, Giuseppe Perelli, Thomas Steeples, and Michael J. Wooldridge. Rational verification: game-theoretic verification of multi-agent systems. *Appl. Intell.*, 51(9):6569–6584, 2021. doi:10.1007/s10489-021-02658-y.

- 2 Rajeev Alur, Aldric Degorre, Oded Maler, and Gera Weiss. On omega-languages defined by mean-payoff conditions. In Luca de Alfaro, editor, *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5504 of *Lecture Notes in Computer Science*, pages 333–347. Springer, 2009. doi:10.1007/978-3-642-00596-1_24.
- 3 Christel Baier, Frantisek Blahoudek, Alexandre Duret-Lutz, Joachim Klein, David Müller, and Jan Strejcek. Generic emptiness check for fun and profit. 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 445–461. Springer, 2019. doi:10.1007/978-3-030-31784-3_26.
- 4 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.
- 5 Julien Bernet, David Janin, and Igor Walukiewicz. Permissive strategies: from parity games to safety games. *RAIRO Theor. Informatics Appl.*, 36(3):261–275, 2002. doi:10.1051/ita:2002013.
- 6 Dietmar Berwanger. Admissibility in infinite games. 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 188–199. Springer, 2007. doi:10.1007/978-3-540-70918-3_17.
- 7 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.
- 8 Patricia Bouyer, Romain Brenguier, Nicolas Markey, and Michael Ummels. Pure Nash equilibria in concurrent deterministic games. *Log. Methods Comput. Sci.*, 11(2), 2015. doi:10.2168/LMCS-11(2:9)2015.
- 9 Patricia Bouyer, Marie Duflot, Nicolas Markey, and Gabriel Renault. Measuring permissivity in finite games. In Mario Bravetti and Gianluigi Zavattaro, editors, *CONCUR 2009 – Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings*, volume 5710 of *Lecture Notes in Computer Science*, pages 196–210. Springer, 2009. doi:10.1007/978-3-642-04081-8_14.
- 10 Patricia Bouyer, Erwin Fang, and Nicolas Markey. Permissive strategies in timed automata and games. *Electron. Commun. Eur. Assoc. Softw. Sci. Technol.*, 72, 2015. doi:10.14279/tuj.eceasst.72.1015.
- 11 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.
- 12 Romain Brenguier and Jean-François Raskin. Pareto curves of multidimensional mean-payoff games. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification – 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II*, volume 9207 of *Lecture Notes in Computer Science*, pages 251–267. Springer, 2015. doi:10.1007/978-3-319-21668-3_15.
- 13 Véronique Bruyère. Computer aided synthesis: A game-theoretic approach. In Émilie Charlier, Julien Leroy, and Michel Rigo, editors, *Developments in Language Theory – 21st International Conference, DLT 2017, Liège, Belgium, August 7-11, 2017, Proceedings*, volume 10396 of *Lecture Notes in Computer Science*, pages 3–35. Springer, 2017. doi:10.1007/978-3-319-62809-7_1.

- 14 Véronique Bruyère. Synthesis of equilibria in infinite-duration games on graphs. *ACM SIGLOG News*, 8(2):4–29, 2021. doi:10.1145/3467001.3467003.
- 15 Véronique Bruyère, Baptiste Fievet, Jean-François Raskin, and Clément Tamines. Stackelberg-Pareto synthesis (extended version). *CoRR*, abs/2203.01285, 2022. doi:10.48550/arXiv.2203.01285.
- 16 Véronique Bruyère, Quentin Hautem, and Jean-François Raskin. On the complexity of heterogeneous multidimensional games. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPICs*, pages 11:1–11:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.CONCUR.2016.11.
- 17 Véronique Bruyère, Quentin Hautem, and Jean-François Raskin. Parameterized complexity of games with monotonically ordered omega-regular objectives. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, volume 118 of *LIPICs*, pages 29:1–29:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPICs.CONCUR.2018.29.
- 18 Véronique Bruyère, Jean-François Raskin, and Clément Tamines. Stackelberg-Pareto synthesis. In Serge Haddad and Daniele Varacca, editors, *32nd International Conference on Concurrency Theory, CONCUR 2021, August 24-27, 2021, Virtual Conference*, volume 203 of *LIPICs*, pages 27:1–27:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICs.CONCUR.2021.27.
- 19 Véronique Bruyère, Jean-François Raskin, and Clément Tamines. Pareto-rational verification. *CoRR*, abs/2202.13485, 2022. arXiv:2202.13485.
- 20 Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Generalized mean-payoff and energy games. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPICs*, pages 505–516. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPICs.FSTTCS.2010.505.
- 21 Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdzinski. Games with secure equilibria. *Theor. Comput. Sci.*, 365(1-2):67–82, 2006. doi:10.1016/j.tcs.2006.07.032.
- 22 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.
- 23 Krishnendu Chatterjee and Vishwanath Raman. Synthesizing protocols for digital contract signing. In Viktor Kuncak and Andrey Rybalchenko, editors, *Verification, Model Checking, and Abstract Interpretation – 13th International Conference, VMCAI 2012, Philadelphia, PA, USA, January 22-24, 2012. Proceedings*, volume 7148 of *Lecture Notes in Computer Science*, pages 152–168. Springer, 2012. doi:10.1007/978-3-642-27940-9_11.
- 24 Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The complexity of rational synthesis. 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 121:1–121:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.ICALP.2016.121.
- 25 Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971. doi:10.1145/800157.805047.

- 26 R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer New York, 2012. URL: <https://books.google.be/books?id=HyTjBwAAQBAJ>.
- 27 E. Allen Emerson and Chin-Laung Lei. Modalities for model checking: Branching time logic strikes back. *Sci. Comput. Program.*, 8(3):275–306, 1987. doi:10.1016/0167-6423(87)90036-0.
- 28 Javier Esparza, Jan Kretínský, Jean-François Raskin, and Salomon Sickert. From LTL and limit-deterministic Büchi automata to deterministic parity automata. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22–29, 2017, Proceedings, Part I*, volume 10205 of *Lecture Notes in Computer Science*, pages 426–442, 2017. doi:10.1007/978-3-662-54577-5_25.
- 29 Dana Fisman, Orna Kupferman, and Yoav Lustig. Rational synthesis. In Javier Esparza and Rupak Majumdar, editors, *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 190–204. Springer, 2010. doi:10.1007/978-3-642-12002-2_16.
- 30 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.
- 31 Erich Grädel and Michael Ummels. *Solution Concepts and Algorithms for Infinite Multiplayer Games*, pages 151–178. Amsterdam University Press, 2008. URL: <http://www.jstor.org/stable/j.ctt46mwfz.11>.
- 32 Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael J. Wooldridge. Automated temporal equilibrium analysis: Verification and synthesis of multi-player games. *Artif. Intell.*, 287:103353, 2020. doi:10.1016/j.artint.2020.103353.
- 33 Paul Hunter and Anuj Dawar. Complexity bounds for regular games. In Joanna Jedrzejowicz and Andrzej Szepietowski, editors, *Mathematical Foundations of Computer Science 2005, 30th International Symposium, MFCS 2005, Gdansk, Poland, August 29 – September 2, 2005, Proceedings*, volume 3618 of *Lecture Notes in Computer Science*, pages 495–506. Springer, 2005. doi:10.1007/11549345_43.
- 34 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.
- 35 Steve Kremer and Jean-François Raskin. A game-based verification of non-repudiation and fair exchange protocols. *J. Comput. Secur.*, 11(3):399–430, 2003. URL: <http://content.iospress.com/articles/journal-of-computer-security/jcs185>, doi:10.3233/jcs-2003-11307.
- 36 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016. doi:10.1007/s10472-016-9508-8.
- 37 Orna Kupferman and Noam Shenwald. The complexity of LTL rational synthesis. In Dana Fisman and Grigore Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems – 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 25–45. Springer, 2022. doi:10.1007/978-3-030-99524-9_2.
- 38 Michael Luttenberger. Strategy iteration using non-deterministic strategies for solving parity games. *CoRR*, abs/0806.2923, 2008. arXiv:0806.2923.
- 39 John F. Nash. Equilibrium points in n -person games. In *PNAS*, volume 36, pages 48–49. National Academy of Sciences, 1950.

- 40 Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
- 41 Christos H. Papadimitriou and Mihalis Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 86–92. IEEE Computer Society, 2000.
- 42 Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *Log. Methods Comput. Sci.*, 3(3), 2007. doi:10.2168/LMCS-3(3:5)2007.
- 43 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190. ACM Press, 1989. doi:10.1145/75277.75293.
- 44 Stéphane Riedweg and Sophie Pinchinat. You can always compute maximally permissive controllers under partial observation when they exist. In *Proc. 2005 American Control Conference, Portland, Oregon, June 8-10 2005*, volume 4, pages 2287–2292, Portland, Oregon, June 2005.
- 45 Dorsa Sadigh, Shankar Sastry, Sanjit A. Seshia, and Anca D. Dragan. Planning for autonomous cars that leverage effects on human actions. In David Hsu, Nancy M. Amato, Spring Berman, and Sam Ade Jacobs, editors, *Robotics: Science and Systems XII, University of Michigan, Ann Arbor, Michigan, USA, June 18 – June 22, 2016*, 2016. doi:10.15607/RSS.2016.XII.029.
- 46 Reinhard Selten. Spieltheoretische Behandlung eines Oligopolmodells mit Nachfrageträgheit. *Zeitschrift für die gesamte Staatswissenschaft*, 121:301–324 and 667–689, 1965.
- 47 A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *J. ACM*, 32(3):733–749, 1985. doi:10.1145/3828.3837.
- 48 A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theor. Comput. Sci.*, 49:217–237, 1987. doi:10.1016/0304-3975(87)90008-9.
- 49 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.
- 50 Yaron Velner and Alexander Rabinovich. Church synthesis problem for noisy input. In Martin Hofmann, editor, *Foundations of Software Science and Computational Structures – 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6604 of *Lecture Notes in Computer Science*, pages 275–289. Springer, 2011. doi:10.1007/978-3-642-19805-2_19.

Concurrent Games with Multiple Topologies

Shaul Almagor ✉ 

Department of Computer Science, Technion, Haifa, Israel

Shai Guendelman ✉

Department of Computer Science, Technion, Haifa Israel

Abstract

Concurrent multi-player games with ω -regular objectives are a standard model for systems that consist of several interacting components, each with its own objective. The standard solution concept for such games is Nash Equilibrium, which is a “stable” strategy profile for the players.

In many settings, the system is not fully observable by the interacting components, e.g., due to internal variables. Then, the interaction is modelled by a partial information game. Unfortunately, the problem of whether a partial information game has an NE is not known to be decidable. A particular setting of partial information arises naturally when processes are assigned IDs by the system, but these IDs are not known to the processes. Then, the processes have full information about the state of the system, but are uncertain of the effect of their actions on the transitions.

We generalize the setting above and introduce Multi-Topology Games (MTGs) – concurrent games with several possible topologies, where the players do not know which topology is actually used. We show that extending the concept of NE to these games can take several forms. To this end, we propose two notions of NE: Conservative NE, in which a player deviates if she can strictly add topologies to her winning set, and Greedy NE, where she deviates if she can win in a previously-losing topology. We study the properties of these NE, and show that the problem of whether a game admits them is decidable.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory; Theory of computation → Automata over infinite objects

Keywords and phrases Concurrent games, Nash Equilibrium, Symmetry, Partial information

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.34

Related Version *Arxiv Version:* <https://arxiv.org/abs/2207.02596>

1 Introduction

Concurrent multi-player games of infinite duration over graphs are a standard modelling tool for representing systems that consist of several interacting components, each having its own objective. Each player in the game corresponds to a component in the interaction. In each round of the game each of the player chooses an action and the next state of the game is determined by the current state and the vector of actions chosen. A strategy for a player is then a mapping from the history of the game so far to the next action.

A strategy profile (i.e., a tuple of strategies, one for each player) induces an infinite trace of states, and the goal of each player is to direct the game into a trace that satisfies her specification. This is modeled by augmenting the game with ω -regular objectives describing the objectives of the players.

Unlike traditional zero-sum games, here the objectives of the players do not necessarily contradict each other. Accordingly, the typical questions about these games concern their stability. Specifically, the most well-known stability measure is Nash Equilibrium (NE): an NE is a strategy profile such that no single player can improve her outcome by unilaterally deviating from the profile. The problem of whether a multi-player game with ω -regular objectives has an NE was shown to be decidable in [6].

In many settings, the players only have partial information about the system, or can view only certain parts of it. This happens when e.g., the system has private and global variables, and the players model threads that can only view the global variables. To this end,



© Shaul Almagor and Shai Guendelman;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 34; pp. 34:1–34:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

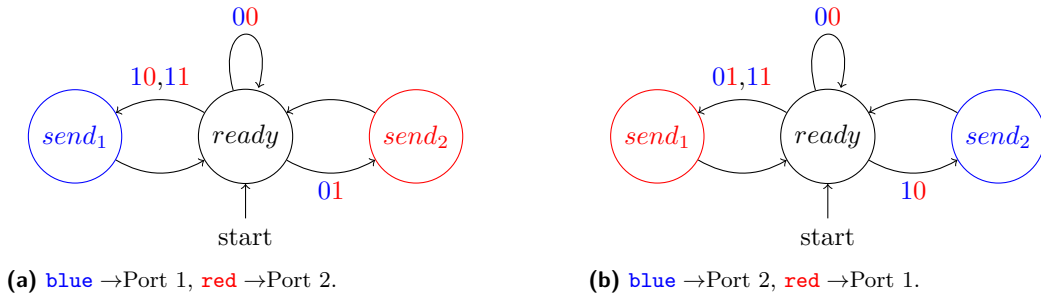
games with *partial information* have been extensively studied in various forms [3, 5, 9, 10]. However, in contrast to the full-information setting, the problem of deciding whether a partial-information multi-player game of infinite duration has a Nash equilibrium is not known to be decidable, and is known to be undecidable in the case of stochastic games [25].

In this work, we introduce and study *Multi-Topology Games (MTG)*. Intuitively, an MTG is a concurrent multi-player game with several transition functions (i.e., topologies). Then, players are fully aware of the possible topologies of the game, but do not know which topology they currently play on. Thus, MTGs capture a restricted form of partial information.

As we now demonstrate, MTGs naturally model the sort of partial information that arises in the context of *process symmetry*.

► **Example 1.** Consider a virtual router with multiple ports. When the router is initialized, several processes are plugged in. The router assigns each process to a port id, but the id is not revealed to the processes. Each process attempts to send messages, and its goal is to have its messages delivered (where some messages may be dropped due to heavy traffic). While the processes know exactly how the router works, they do not know which port they are assigned to. Therefore, their strategies must be oblivious to their port number.

As a concrete example, consider the concurrent game in Figure 1 with players {blue, red}. When both players know the port assignment, for example, blue → Port 1 and red → Port 2, then blue can win by always taking action 1, and red will lose in any strategy. However, if the port assignment is not known then in order for either player to win under both port assignments, the players must coordinate e.g., by taking turns trying to send a message. Thus, a-priori, the game has two possible topologies: Figure 1a and Figure 1b.



■ **Figure 1** Router game from Example 1. The players are blue and red, and the router has two ports 1, 2. In every round each player can try to *send* (action 1), or *wait* (action 0). The labels on the edges describe the actions of the players. The first is the action of the blue player, and the second is the action of the red player. From *ready*, if only the player in Port $i \in \{1, 2\}$ tries to send, the game transitions to $send_i$. If both players try to send, the router prioritizes the request from Port 1. The objective of the player Port i is to visit $send_i$ infinitely many times. Note that $send_i$ is colored according to the player that tries to reach it in each port assignment.

These type of settings are commonly referred to as *process symmetry* [12, 15, 18, 19, 1], and have been studied in several contexts (e.g., model checking with symmetry reductions). However, to our knowledge this setting has not been studied in games. In Section 3.1 we demonstrate how MTGs can model the general setting of process symmetry in games. ◻

In an MTG, a strategy for a player maps sequences of states to an action, and hence does not depend on a certain topology. Unlike standard games, a strategy profile in an MTG no longer induces a single trace, but rather a set of traces, one per topology. Thus, a player can no longer be said to be “winning” or “losing” in a strategy profile, as this may vary between topologies. In particular, it is not clear how analogues of Nash equilibrium and social optimum should be defined.

To this end, we propose two versions of Nash equilibria, corresponding to two extremities: in a Conservative NE (CNE), a player deviates if she can increase (w.r.t. containment) the set of topologies she wins in. In a Greedy NE (GNE), a player deviates if she can win in a currently-losing topology (even at the cost of losing some of the currently-winning topologies).

We study the properties of CNE and GNE and compare their strictness, showing that a GNE is also a CNE, but the converse does not hold. We also compare their properties to those of the standard notion of NE. Our main technical contribution is showing that the problem of whether a game has a CNE (resp. GNE) is decidable.

Related Work. A central work concerning NE in concurrent games is [6], where the problem of deciding whether a concurrent game admits an NE was studied for various winning conditions. Apart from establishing tight complexity bounds, this work also introduced the *suspect game* – a useful technique for reasoning about concurrent games. Interestingly, the suspect game does not seem to be adaptable to reason about MTGs, suggesting a fundamental difference between the models.

Zero-sum concurrent reachability games were studied in [13], where fundamental techniques for reasoning about them were developed. We remark that the zero-sum setting is technically very different to ours, due to the non-adversarial nature of the players.

Concurrent games can be formulated in the turn-based setting using partial information. The latter were extensively studied, e.g., in [9, 22, 10, 3, 8, 14], typically in the zero-sum setting.

Finally, the work in [3] extends strategy logic [11] with imperfect information. The authors show that, in general, the model checking problem for this logic is undecidable, but it is decidable in some special cases. Unfortunately, these cases do not readily capture MTGs.

Paper organization. In Section 2 we present the basic definitions of concurrent games. In Section 3 we formally define MTGs, introduce two notions of equilibria for them, and study their properties. In Section 4 we give our main technical result, establishing the decidability of detecting CNE in MTGs. In Section 5 we establish the decidability of detecting GNE. Finally, in Section 6 we discuss our results and some extensions, and detail future directions.

2 Preliminaries

A *concurrent parity game* is a tuple $\mathcal{G} = \langle \text{Pla}, \text{S}, s_0, \text{Act}, \delta, (\alpha_p)_{p \in \text{Pla}} \rangle$ where the components are as follows. Pla is a finite set of players, S is a finite set of states, $s_0 \in \text{S}$ is an initial state, Act is a finite set of actions. The transition function $\delta : \text{S} \times \text{Act}^{\text{Pla}} \rightarrow \text{S}$ maps a state and an *action profile* (i.e., $\mathbf{a} = (a_p)_{p \in \text{Pla}} \in \text{Act}^{\text{Pla}}$) to the next state. Every player $p \in \text{Pla}$ has a parity objective $\alpha_p \subseteq \text{S}^\omega$, as we describe below.

A *play* of \mathcal{G} is an infinite sequence of states $\rho = s_0, s_1, \dots \in \text{S}^\omega$ such that for every step $i \in \mathbb{N}$ there exists an action profile \mathbf{a} such that $s_{i+1} = \delta(s_i, \mathbf{a})$. For $k \geq 1$ we denote the length- k prefix of $\rho_{\leq k} = s_0, \dots, s_{k-1} \in \text{S}^+$. We denote by $\text{Inf}(\rho)$ the set of states that occur infinitely often in ρ . A *parity objective* is given by a function $\Omega : \text{S} \rightarrow \{0, \dots, d\}$ for some $d \in \mathbb{N}$. Then, ρ satisfies the objective if $\min\{\Omega(s) \mid s \in \text{Inf}(\rho)\}$ is even. Thus, the objective α_p is the set of all plays that satisfy the parity function of Player p . In the following, we mostly use the parity function implicitly, and so we do not include Ω in the description of \mathcal{G} .

The *description size* of \mathcal{G} , denoted $|\mathcal{G}|$ is the number of bits required to represent the components of \mathcal{G} .

► **Remark 2 (Game representation).** Note that we assume an explicit representation of the transition function as a table. In particular, we describe for every state the transition on every action profile in Act^{Pla} . Thus, the size of the transition functions is exponential in $|\text{Pla}|$.

This is in contrast with a more succinct representation, i.e., representing the transition function as a circuit. We choose this focus to eliminate the complexity effect of succinct representation.

A *history* of \mathcal{G} is a finite prefix of a play $h \in \mathcal{S}^+$. A *strategy* for Player p is a function $\sigma : \mathcal{S}^+ \rightarrow \text{Act}$ that maps a history to the next action of Player p . A *strategy profile* $\sigma = (\sigma_p)_{p \in \text{Pla}}$ is vector of strategies, one for each player. We denote the set of all strategies by $\Sigma_{\mathcal{G}}$ and the set of all strategy profiles by $\Sigma_{\mathcal{G}}^{\text{Pla}}$ (we omit the subscript \mathcal{G} when it is clear from context). A strategy profile σ can be thought as a function that maps histories to action profiles: given a history $h \in \mathcal{S}^+$ we have $\sigma(h) = (\sigma_p(h))_{p \in \text{Pla}} \in \text{Act}^{\text{Pla}}$.

For a strategy profile σ we define its *outcome* to be the infinite sequence of states (i.e. play) in \mathcal{G} that is taken when all the players follow their strategies in σ . Formally, $\text{out}_{\mathcal{G}}(\sigma) = s_0 s_1 \dots \in \mathcal{S}^{\omega}$ where s_0 is the initial state, and for every $i \geq 1$ we have $s_i = \delta(s_{i-1}, \sigma(s_0, \dots, s_{i-1}))$. Consider a play $\rho \in \mathcal{S}^{\omega}$. The set of *winners* in ρ is the set of players whose objectives are met in ρ . Formally, $\text{Win}_{\mathcal{G}}(\rho) = \{p \in \text{Pla} \mid \rho \in \alpha_p\} \subseteq \text{Pla}$. The set of winners in a strategy profile σ is then $\text{Win}_{\mathcal{G}}(\sigma) = \text{Win}_{\mathcal{G}}(\text{out}_{\mathcal{G}}(\sigma))$. Player p is said to be *losing* if she is not winning.

► **Remark 3 (Action visibility).** Note that strategies are defined to “see” only the history of visited states, and not the history of actions taken by the other players. This is a standard and natural assumption [6, 10] for concurrent models. There are, however, works (e.g., [2]) where players can view the entire action history. The latter approach is slightly easier to reason about, as players have full information on the game progress.

A strategy profile σ is a *Nash Equilibrium (NE)* if, intuitively, no single player can benefit from unilaterally changing her strategy. Since the objectives in our setting are binary, “benefiting” amounts to moving from the set of losers to the set of winners. We refer to such a change as a *beneficial deviation*. Formally, consider a strategy profile σ , a player $p \in \text{Pla}$ and a strategy $\sigma'_p \in \Sigma_{\mathcal{G}}$ for Player p . We denote by $\sigma[p \mapsto \sigma'_p] \in \Sigma_{\mathcal{G}}^{\text{Pla}}$ the strategy profile obtained from σ by replacing σ_p with σ'_p . Then, σ is an NE if for every player $p \in \text{Pla}$ and every strategy $\sigma'_p \in \Sigma_{\mathcal{G}}$ for Player p , if $p \in \text{Win}_{\mathcal{G}}(\sigma[p \mapsto \sigma'_p])$ then $p \in \text{Win}_{\mathcal{G}}(\sigma)$. Viewed contrapositively: if p loses when \mathcal{G} is played with σ , then p also loses after changing her strategy.

3 Multi-Topology Games

A *multi-topology game (MTG)* is a tuple $\mathcal{G} = \langle \text{Pla}, \mathcal{S}, s_0, \text{Act}, \text{Top}, (\delta_t)_{t \in \text{Top}}, (\alpha_{t,p})_{t \in \text{Top}, p \in \text{Pla}} \rangle$ where $\text{Pla}, \mathcal{S}, s_0, \text{Act}$, are the same as in concurrent games. Top is a finite set of *topologies*, and for every $t \in \text{Top}$ we have a transition function $\delta_t : \mathcal{S} \times \text{Act}^{\text{Pla}} \rightarrow \mathcal{S}$ and objective $\alpha_{t,p} \subseteq \mathcal{S}^{\omega}$ for every player $p \in \text{Pla}$. An MTG can be thought of as a tuple of games over the same states, players and actions. That is, for $t \in \text{Top}$, we can define $\mathcal{G}_t = \langle \text{Pla}, \mathcal{S}, s_0, \text{Act}, \delta_t, (\alpha_{t,p})_{p \in \text{Pla}} \rangle$ to be the concurrent parity game obtained by fixing the transition function to δ_t and the objective for Player p to $\alpha_{t,p}$.

Crucially, the players are assumed to have no a-priori information on which topology is selected when the game is played. This is captured in the definition of strategies: a strategy for Player p is identical to the setting of concurrent parity games, i.e., $\sigma_p : \mathcal{S}^+ \rightarrow \text{Act}$. This lifts to strategy profiles and outcomes, as per Section 2. In particular, a strategy σ in \mathcal{G}

can be applied to \mathcal{G}_t for every $t \in \text{Top}$. Consider a strategy profile $\sigma \in \Sigma^{\text{Pla}}$. The *winning topologies* of Player p is the set of topologies that Player p wins in when \mathcal{G} is played with strategy profile σ . Formally, $\text{WinTop}_{\mathcal{G}}^p(\sigma) = \{t \in \text{Top} \mid p \in \text{Win}_{\mathcal{G}_t}(\sigma)\}$.

3.1 Process Symmetry in Concurrent Games

As we discuss in Section 1, a central motivation for MTGs come from settings where players plug in to the system without knowing their identity. This setting is commonly referred to as *process symmetry* [12, 15, 18, 19, 1]. Symmetry in games was studied in [24, 23, 7, 17] for *strategic form games*, which are games with a single turn. In [5, 26], symmetry in concurrent games was studied by imposing restrictions on the game structure. We consider a different setting, where processes $1, \dots, k$ log into a system described as a concurrent game, but the index of the action controlled by each process is not revealed to the processes. This setting is naturally modelled as an MTG, as follows.

Consider a concurrent game $\mathcal{G} = \langle \text{Pla}, \mathcal{S}, s_0, \text{Act}, \delta, (\alpha_p)_{p \in \text{Pla}} \rangle$ with $k \geq 2$ players, and that $\text{Pla} = \{1, \dots, k\}$. We obtain from \mathcal{G} an MTG with $k!$ topologies by letting each topology correspond to a different permutation of the players. Formally, consider a permutation $\pi \in \mathcal{S}_k$, where \mathcal{S}_k is the set of permutations over $\{1, \dots, k\}$. For an action profile $\mathbf{a} \in \text{Act}^{\text{Pla}}$ we define $\pi(\mathbf{a}) = (a_{\pi^{-1}(1)}, \dots, a_{\pi^{-1}(k)})$. That is, the action performed by Player i is taken at index $\pi(i)$. We now obtain the MTG $\mathcal{G}' = \langle \text{Pla}, \mathcal{S}, s_0, \text{Act}, \mathcal{S}_k, (\delta_\pi)_{\pi \in \mathcal{S}_k}, (\alpha_{\pi,p})_{\pi \in \mathcal{S}_k, p \in \text{Pla}} \rangle$ where \mathcal{S}_k is the set of topologies, δ_π is obtained by applying π to the action profile of the players, that is, for $s \in \mathcal{S}$ and $\mathbf{a} \in \text{Act}^{\text{Pla}}$ we have $\delta_\pi(s, \mathbf{a}) = \delta(s, \pi(\mathbf{a}))$. Finally, the objective of Player p is $\alpha_{\pi,p} = \alpha_{\pi(p)}$. Figure 1 is an example of such game.

3.2 Solution Concepts

Recall that in NE, a beneficial deviation moves a player from losing to winning. In MTGs, however, winning is no longer binary. Indeed, a strategy profile associates with each player a set of winning topologies. Thus, the meaning of “beneficial deviation” becomes context dependent. We introduce and study two notions of equilibria for MTGs that lie on two “extremities”: in the *conservative* approach, a deviation is beneficial if it strictly increases (w.r.t. containment) the set of winning topologies. In the *greedy* approach, a deviation is beneficial if a previously-losing topology becomes winning. We now turn to formally define and demonstrate these notions.

Conservative NE. A *conservative NE (CNE)* is a strategy profile σ where no player can deviate from σ and have her winning topologies be a strict superset¹ of her winning topologies when obeying σ . Formally, $\sigma \in \Sigma^{\text{Pla}}$ is a CNE if the following holds:

$$\forall p \in \text{Pla} \forall \sigma'_p \in \Sigma_{\mathcal{G}}^p \left((\forall t \in \text{Top} \ p \in \text{Win}_{\mathcal{G}_t}(\sigma[p \mapsto \sigma'_p]) \rightarrow \text{Win}_{\mathcal{G}_t}(\sigma)) \vee \right. \\ \left. (\exists t \in \text{Top} \ p \notin \text{Win}_{\mathcal{G}_t}(\sigma[p \mapsto \sigma'_p]) \wedge p \in \text{Win}_{\mathcal{G}_t}(\sigma)) \right)$$

Equivalently, this condition can be written in terms of the set of winning topologies:

$$\forall p \in \text{Pla} \forall \sigma'_p \in \Sigma_{\mathcal{G}}^p \neg(\text{WinTop}_{\mathcal{G}}^p(\sigma) \subsetneq \text{WinTop}_{\mathcal{G}}^p(\sigma[p \mapsto \sigma'_p]))$$

We refer to this notion as *conservative* since a deviating player wants to conserve her existing winning strategies.

¹ we emphasize that the relation \subsetneq means “strictly contained”.

Greedy NE. A *greedy NE (GNE)* is a strategy profile σ where no player can unilaterally deviate and win in a previously-losing topology. Formally, $\sigma \in \Sigma^{\text{Pla}}$ is a GNE if the following holds:

$$\forall p \in \text{Pla} \forall \sigma'_p \in \Sigma^p_{\mathcal{G}} \forall t \in \text{Top} (p \in \text{Win}_{\mathcal{G}_t}(\sigma[p \mapsto \sigma'_p]) \rightarrow p \in \text{Win}_{\mathcal{G}_t}(\sigma))$$

Equivalently, this condition can also be written in terms of the set of winning topologies:

$$\forall p \in \text{Pla} \forall \sigma'_p \in \Sigma^p_{\mathcal{G}} (\text{WinTop}_{\mathcal{G}}^p(\sigma[p \mapsto \sigma'_p]) \subseteq \text{WinTop}_{\mathcal{G}}^p(\sigma))$$

The latter formulation shows that in a GNE, for every player and for every deviation, the player's winning topologies when deviating are a subset of the player's winning topologies when obeying σ . It refers to this notion as *greedy* since it assumes that a player deviates if she improves her outcome in a single topology, disregarding the outcome in other topologies.

► **Example 4 (CNE and GNE).** Recall the router game from Figure 1. The strategy profile where Player **blue** repeatedly plays $(0, 0, 1, 1)^\omega$ and **red** plays $(1, 1, 0, 0)^\omega$ is a CNE, since the set of winning topologies of this profile is $\{1, 2\}$ for both players. Thus, no deviation can win in strictly more topologies.

Note that the same strategy profile is also a GNE, since every set of winning topologies is a subset of $\{1, 2\}$.

► **Remark 5 (Additional notions of NE).** CNE and GNE are based on the \subseteq preorder on the sets of topologies, 2^{Top} . In Section 6 we discuss other notions of NE in MTGs.

3.3 Properties of CNE and GNE

We start by examining some properties and relationships between the notions of CNE and GNE, as well as their relation to standard NE.

Consider an MTG $\langle \text{Pla}, \mathcal{S}, s_0, \text{Act}, \text{Top}, (\delta_t)_{t \in \text{Top}}, (\alpha_{t,p})_{t \in \text{Top}, p \in \text{Pla}} \rangle$. The following observation is immediate from the definitions of GNE and CNE, since if there is only a single topology, the MTG collapses into a concurrent game.

► **Observation 6.** *If $\text{Top} = \{t\}$, i.e. there is only a single topology t , then the definitions of NE in \mathcal{G}_t coincides with that of CNE and of GNE in \mathcal{G} .*

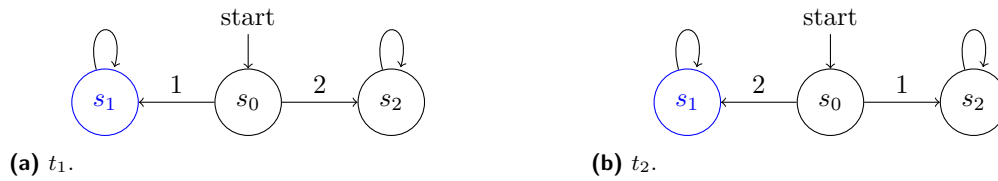
Next, we observe that GNE is a stricter notion than CNE. Indeed, a beneficial deviation in the conservative setting (namely increasing the set of winning topologies) implies a beneficial deviation in the greedy setting (namely winning in a previously-losing topology). Contrapositively, if there is no greedy beneficial deviation, there is also no conservative beneficial deviation. We thus have the following.

► **Observation 7.** *Let \mathcal{G} be an MTG. If σ is a GNE in \mathcal{G} then σ is a CNE in \mathcal{G} .*

The following example shows that the implication of Observation 7 is strict. That is, there are MTGs with a CNE but without a GNE.

► **Example 8 (CNE without GNE).** Consider the single-player game depicted in Figure 2. The outcome of the game depends only on the first action that the player takes and the topology that the game is played in. If the player takes action 1, then the set of winning topologies is $\{t_1\}$. If the player takes action 2, then the set of winning topologies is $\{t_2\}$. Since $\{t_1\} \not\subseteq \{t_2\}$ and $\{t_2\} \not\subseteq \{t_1\}$, there is no GNE in the game, as the player can switch strategies from t_1 to t_2 and vice versa to win in a previously-losing topology.

However, since there is no strategy for the player such that the set of winning topologies is $\{t_1, t_2\}$ (the only strict superset of $\{t_1\}$ and $\{t_2\}$), then every strategy is a CNE.



■ **Figure 2** A single player MTG with two topologies, t_1 and t_2 . In both topologies, the objective of the player is to reach s_1 (it is easy to capture this using a parity objective).

► **Remark 9** (Best-response dynamics in GNE). Example 8 demonstrates that, in stark contrast to NE, an MTG might not have a GNE even when there is only a single player. This has to do, in particular, with the notion of *best-response dynamics*: in standard games, one can approach an NE by starting from some profile, and repeatedly letting players deviate to their best-response strategy, until this process converges. While this does not always converge, it does so for a large class of games (e.g., *finite-potential games* [21]).

Thus, Example 8 shows that best-response does not converge even for a single player in MTGs, whereas it does converge for a single player both for standard NE, as well as in CNE for MTGs. Indeed, the best-response of a single player in the conservative setting will increase her set of winning topologies to the maximum, and from there she will no longer have incentive to deviate.

Remark 9 reflects the intuition that a GNE must be stable in each topology separately. That is, it captures the notion “NE on all topologies”, in the following sense.

► **Observation 10.** A GNE σ is also an NE in \mathcal{G}_t for every $t \in \text{Top}$.

Indeed, if σ was not an NE in \mathcal{G}_t for some $t \in \text{Top}$, then a player that deviates from σ in \mathcal{G}_t would similarly deviate from σ in \mathcal{G} , greedily winning in the previously-losing topology t .

In contrast, we now show that CNE is a more intricate notion, and might hold even when there is no NE in the separate topologies.



■ **Figure 3** Symmetric XOR game. The players are **blue** and **red**. In topology t_1 , the objective of **blue** is to reach s_1 , and the objective of **red** is to reach s_2 . In topology t_2 the objectives of the players are swapped. The game starts from s_0 . If both players take the same action, then the game transitions to state s_1 and gets stuck there. If the players take different actions then the game transitions to s_2 and gets stuck there.

► **Example 11** (CNE without NE). Consider the Symmetric XOR game \mathcal{G} depicted in Figure 3. Note that neither \mathcal{G}_{t_1} nor \mathcal{G}_{t_2} have a NE, since if a strategy for a single player is fixed, the other player can respond to it and win.

On the other hand, any strategy profile is a CNE, since every player always wins in exactly one topology. Thus, there is no way for a player to deviate and get strict superset of winning topologies.

There are MTGs without CNE. For example, every concurrent game \mathcal{G} without an NE can be viewed as an MTG with a single topology t_1 . Since there is no NE in \mathcal{G} , then for every profile σ there exists a player p that loses with σ , which corresponds to $\text{WinTop}_{\mathcal{G}}^p(\sigma) = \emptyset$ but p can deviate and win \mathcal{G} , which corresponds to $\text{WinTop}_{\mathcal{G}}^p(\sigma[p \mapsto \sigma'_p]) = \{t_1\}$. Since $\emptyset \subsetneq \{t_1\}$, then σ is not a CNE.

4 Existence of Conservative NE is Decidable

We now turn to our main technical contribution – showing that the existence of a CNE is a decidable property.

► **Theorem 12.** *The problem of deciding, given an MTG \mathcal{G} , whether there exists a CNE in \mathcal{G} is in 2-EXPTIME.*

The remainder of the section is devoted to proving Theorem 12. Our solution is based on a reduction to the problem of solving a restricted form of partial-information game. We then employ a result from [10], and obtain the complexity result by a careful analysis of the construction. The rest of the section is organized as follows. In Section 4.1 we present the model of partial-information games and the result of [10]. In Section 4.2 we give an overview of the reduction and in Section 4.3 we describe and analyze the reduction from our setting.

4.1 Partial-Information Games

Partial-information games (also known as *games with incomplete information*) are a ubiquitous model for settings where the players cannot fully observe the state of the game due to e.g., private/hidden variables, unknown parameters or abstractions of part of the system.

Formally, a *partial-information game* is a tuple $\mathcal{G} = \langle \text{Pla}, \mathcal{S}, s_0, \text{Act}, \delta, (\mathcal{O}_p)_{p \in \text{Pla}} \rangle$ where Pla , \mathcal{S} , s_0 , Act and δ are the same as in concurrent games. For every player $p \in \text{Pla}$, the set of *observations* $\mathcal{O}_p \subseteq 2^{\mathcal{S}}$ is a partition of \mathcal{S} . We omit the acceptance condition, and we will include it explicitly in Theorem 13 below.

Intuitively, when the play of \mathcal{G} is at state $s \in \mathcal{S}$, Player p can only observe $o \in \mathcal{O}_p$ such that $s \in o$, and needs to select an action according to o . Thus, we distinguish between *state histories*, \mathcal{S}^+ and *observation histories (of Player p)*, $(\mathcal{O}_p)^+$. For $s \in \mathcal{S}$ we define $\text{obs}_p(s) = o \in \mathcal{O}_p$ to be the unique observation of Player p such that $s \in o$. We extend obs_p to histories: let $h = s_0 s_1 \dots s_k \in \mathcal{S}^+$ be a state history, we define $\text{obs}_p(h) = \text{obs}_p(s_0) \text{obs}_p(s_1), \dots, \text{obs}_p(s_k) \in (\mathcal{O}_p)^+$ to be the corresponding observation history.

Strategies are *observation based*, that is, a *strategy* for Player p is a function $\sigma_p : \mathcal{O}_p^+ \rightarrow \text{Act}$. Since different players may have different observation sets, we denote by $\Sigma_{\mathcal{G}}^p$ the set of all strategies for Player p . We denote by $\Sigma_{\mathcal{G}}^{\text{Pla}}$ the set of all strategy profiles.

Similarly to concurrent games, a strategy profile σ can be thought of as a function that maps histories to action profiles $\sigma(h) = (\sigma_p(\text{obs}_p(h)))_{p \in \text{Pla}} \in \text{Act}^{\text{Pla}}$, and we define $\text{out}_{\mathcal{G}}(\sigma) \in \mathcal{S}^{\omega}$ similarly to concurrent games.

We say that Player $p \in \text{Pla}$ has *perfect information* if $\mathcal{O}_p = \{\{s\} \mid s \in \mathcal{S}\}$. That is, Player p can observe the exact state of the game. If all players have perfect information then the game is a *perfect information game*, and coincides with our definition of concurrent games. We say that *Player i is less informed than Player j* if \mathcal{O}_j is a refinement of \mathcal{O}_i . That is, for every $o_j \in \mathcal{O}_j$ there exists $o_i \in \mathcal{O}_i$ such that $o_j \subseteq o_i$.

Finally, consider an objective $\alpha \subseteq \mathcal{S}^{\omega}$, we say that α is *visible to Player p* if for every $\rho, \rho' \in \mathcal{S}^{\omega}$ such that $\text{obs}_p(\rho) = \text{obs}_p(\rho')$ we have that $\rho \in \alpha$ if and only if $\rho' \in \alpha$. That is, the objective can be defined according to observation sequences rather than plays.

The following theorem is a result from [10] that will serve as the target of our reduction.

► **Theorem 13.** *Let $\mathcal{G} = \langle \text{Pla}, S, s_0, \text{Act}, \delta, (\mathcal{O}_p)_{p \in \text{Pla}} \rangle$ be a partial information game, with $\text{Pla} = \{1, 2, 3\}$ where Player 1 less informed than Player 2. Let $\alpha \subseteq S^\omega$ be parity objective over S . The problem of deciding whether $\exists \sigma_1 \in \Sigma_{\mathcal{G}}^1 \forall \sigma_2 \in \Sigma_{\mathcal{G}}^2 \exists \sigma_3 \in \Sigma_{\mathcal{G}}^3 \text{out}_{\mathcal{G}}(\sigma_1, \sigma_2, \sigma_3) \in \alpha$ is 2-EXPTIME complete.*

4.2 Overview of the Reduction

We now turn to describe a reduction from the CNE existence problem to the setting of Theorem 13. We start with a high-level description. Consider an MTG \mathcal{G} . Instead of asking directly whether \mathcal{G} admits a CNE, we first fix a set of “intended” winning topologies $T_p \subseteq \text{Top}$ for each player $p \in \text{Pla}$. Then, we ask whether \mathcal{G} admits a CNE σ in which $\text{WinTop}_{\mathcal{G}}^p(\sigma) = T_p$ for every $p \in \text{Pla}$. If we are able to answer the latter problem, we can iterate over every possible tuple $(T_p)_{p \in \text{Pla}}$ (or nondeterministically guess a set) and conclude whether \mathcal{G} admits a CNE. We remark that this approach is reminiscent of the technique in [6], where the existence of an NE in a game is decided by first guessing a “witness” path.

Once the set of intended topologies is fixed, we construct a 3-player partial-information game whose players are **Eve**, **Adam** and **Snake**, with the following roles:

- **Eve** controls the coalition of all players, and suggests a strategy profile σ by selecting the actions for all the players at each step.
- **Adam** selects a deviating player p , and the deviating strategy σ'_p for that player. In addition, **Adam** selects a set $T \subseteq \text{Top}$ in which Player p tries to win when playing σ'_p .
- **Snake** helps² **Eve** by selecting a concrete topology t from the set T picked by **Adam**.

The game starts with **Adam** and **Snake** choosing p , T and $t \in T$. It then proceeds with **Eve** and **Adam** choosing σ and σ'_p , respectively, while playing on \mathcal{G}_t . The observation sets of the players are such that both **Eve** and **Adam** can only observe the current state of the game, so **Eve** is ignorant of p , T and t , and **Adam** is ignorant of t (except knowing that $t \in T$).

The objective of **Eve** and **Snake** is then composed of three conditions:

1. **Snake** must choose a topology $t \in T$.
2. If the strategy σ'_p proposed by **Adam** does not in fact deviate from the profile σ proposed by **Eve** (dubbed “**Adam obeys Eve**”), and if $t \in T_p$, i.e., p was intended to win in t , then the outcome must be winning for Player p .
3. If **Adam** selected T to contain a topology not in T_p (i.e., Player p potentially tries to win in a superset of T_p), then the outcome must be losing for Player p .

The overall idea is that if **Eve** can find a strategy for all the players, from which any deviation choice of **Adam** can be shown to be non-beneficial by an appropriate choice by **Snake**, then there is a CNE with the intended winning topologies, and vice-versa.

There are, however, some caveats: first, in order to allow **Adam** to choose any set of topologies, the size of the game would be exponential, which is undesirable. Second, it is not immediate that the conjunction of conditions above can be captured by a small parity objective (since the parity condition does not allow conjunction without a change of state space [4]). Third, we need to separate the cases where **Adam** obeys **Eve**. In the following we give the complete construction, which overcomes these caveats.

² It is arguable whether this matches the biblical interpretation. This work makes no theological claims.

4.3 Reduction to Partial Information Game

Consider an MTG $\mathcal{G} = \langle \text{Pla}, \text{S}, s_0, \text{Act}, \text{Top}, (\delta_t)_{t \in \text{Top}}, (\alpha_{t,p})_{t \in \text{Top}, p \in \text{Pla}} \rangle$. For every Player $p \in \text{Pla}$, fix $T_p \subseteq \text{Top}$ to be the intended set of winning topologies.

Game construction. We construct a 3-player partial-information game \mathcal{H} with the following components. The players are **Eve**, **Adam** and **Snake**. The states of \mathcal{H} are $Q_{\mathcal{H}} = \{q_0\} \cup Q$, where q_0 is a designated initial state and $Q \subseteq \text{S} \times \text{Pla} \times 2^{\text{Top}} \times \text{Top} \times \{\text{true}, \text{false}\}$ is described in the following. A state $(s, p, T, t, b) \in Q$ comprises $s \in \text{S}$ which tracks the state of \mathcal{G} , a player $p \in \text{Pla}$ that is controlled by **Adam**, a set $T \subseteq \text{Top}$ of topologies that **Adam** picks, $t \in \text{Top}$ is a topology picked by **Snake** and determines the topology \mathcal{G} is played in, and a bit $b \in \{\text{true}, \text{false}\}$ which tracks whether **Adam** obeys **Eve**.

In order to restrict the state space to a polynomial size in $|\mathcal{G}|$, i.e. reduce the 2^{Top} component, we define $\mathcal{T}_p = \{T_p \cup \{t\} \mid t \in \text{Top}\} \subseteq 2^{\text{Top}}$ and $\mathcal{T} = (\bigcup_{p \in \text{Pla}} \mathcal{T}_p) \cup \{\{t\} \mid t \in \text{Top}\}$. Note that $|\mathcal{T}| \leq (|\text{Pla}| + 1) \cdot |\text{Top}| \leq 2 \cdot |\text{Pla}| \cdot |\text{Top}|$. We now define $Q = \text{S} \times \text{Pla} \times \mathcal{T} \times \text{Top} \times \{\text{true}, \text{false}\}$. Intuitively, the restriction of 2^{Top} to \mathcal{T} is sound, since if a Player p is able to deviate and increase her winning topologies from T_p to some T , then she can also increase her winning topologies by just one topology, and thus we can assume $T \in \mathcal{T}_p$.

We now turn to define the transitions in \mathcal{H} . The actions are defined implicitly by the transitions.³ From q_0 , **Adam** selects a player $p \in \text{Pla}$ and a set of topologies $T \in \mathcal{T}_p$. As explained in Section 4.2, **Adam** controls Player p and attempts to show that p wins in T . Still in q_0 , **Snake** selects a topology $t \in \text{Top}$ that \mathcal{G} will be played in. Then, \mathcal{H} transitions to state $(s_0, p, T, t, \text{true}) \in Q$.

Henceforth, p, T and t remain fixed throughout the play, and **Snake** has no further effect on the play. From state $(s, p, T, t, b) \in Q$, **Eve** chooses an action profile $\mathbf{a} \in \text{Act}^{\text{Pla}}$ and **Adam** selects an action $a'_p \in \text{Act}$. Then, the game transitions to state $(s', p, T, t, b') \in Q$ such that $s' = \delta_t(s, \mathbf{a}[p \mapsto a'_p])$, and $b' = b \wedge a_p = a'_p$. That is, **Eve** chooses an action profile, **Adam** chooses a possible deviation, and the game proceeds according to \mathcal{G}_t . If **Adam** actually deviates, the bit b becomes **false** and remains so throughout the play. Adding $\{\{t\} \mid t \in \mathcal{T}\}$ to \mathcal{T} is to make sure that if Player p is supposed to win in topology t (that is, $t \in T_p$), then, the profile suggested by **Eve** must lead to player p winning in topology t . If not, **Adam** can choose $\{t\}$ and Player p at the start of the game, and obey **Eve**, falsifying one of **Eve**'s winning conditions (ψ_2).

Next, we define the observation sets of \mathcal{H} . For a state $q = (s, p, T, t, b) \in Q$ we define the *projection* of q on \mathcal{G} to be $\text{proj}(q) = s$. For every state $s \in \text{S}$ of \mathcal{G} , let $o_s = \{q \in Q \mid \text{proj}(q) = s\} \subseteq Q$. The observation sets in \mathcal{H} are $\mathcal{O}_{\text{Adam}} = \mathcal{O}_{\text{Eve}} = \mathcal{O} = \{\{q_0\}\} \cup \{o_s \mid s \in \text{S}\}$. That is, **Adam** and **Eve** can observe the initial state q_0 , and for every $q \in Q$ they can only observe $\text{proj}(q)$. **Snake** has perfect information.

This completes the construction of the game \mathcal{H} (recall that \mathcal{H} does not have an objective). We proceed to formalize the connection between \mathcal{G} and \mathcal{H} .

Correspondence between \mathcal{H} and \mathcal{G} . We lift the definition of projection to plays: for a play $\rho = q_0 q_1 q_2 \dots \in q_0 \cdot Q^\omega$ of \mathcal{H} define $\text{proj}(\rho) = \text{proj}(q_1)\text{proj}(q_2)\dots$ (note that we skip the initial state q_0). We also define the predicate $\text{obey}(\rho) = \bigwedge_{i \geq 1} b_i$, where b_i is the **true/false** bit of q_i . That is, $\text{obey}(\rho)$ is true if and only if **Adam** always takes the actions suggested by **Eve**. When $\text{obey}(\rho)$ is true, we say that **Adam** *obeys* **Eve**.

³ In the model we describe, actions are identical for all players. However, the model of [10] allows different actions as well as enabled and disabled actions in each state, so it is easy to accommodate our actions.

Since the observation of **Eve** and **Adam** correspond to states of \mathcal{G} , there is a correspondence between plays, observation-histories and strategies in \mathcal{H} to plays, histories and strategies in \mathcal{G} . We make this precise in the following. Consider the function $\gamma_{\text{obs}} : \{q_0\} \cdot \mathcal{O}^\omega \rightarrow \mathcal{S}^\omega$ defined $\gamma_{\text{obs}}(\{q_0\}, o_{s_0}, o_{s_1}, \dots) = s_0, s_1, \dots$. Since $o_s = \{q \mid \text{proj}(q) = s\}$ for every $s \in \mathcal{S}$, we have that γ_{obs} is a bijection between observation-plays of **Eve** and **Adam** in \mathcal{H} , and plays of \mathcal{G} . By looking at finite sequences, namely histories, we can refer to γ_{obs} as a bijection between observation-histories of **Adam** and **Eve** in \mathcal{H} , and histories in \mathcal{G} . Moreover, since strategies in \mathcal{H} are observation based, the following functions are also bijective:

- $\gamma_{\text{Eve}} : \Sigma_{\mathcal{H}}^{\text{Eve}} \rightarrow \Sigma_{\mathcal{G}}$ defined by $\gamma_{\text{Eve}}(\sigma_{\text{Eve}}) = \sigma_{\text{Eve}} \circ \gamma_{\text{obs}}^{-1}$.
- $\gamma_{\text{Adam}} : \Sigma_{\mathcal{H}}^{\text{Adam}} \rightarrow \bigcup_{p \in \text{Pla}} \{p\} \times \mathcal{T}_p \times \Sigma_{\mathcal{G}}^p$ defined $\gamma_{\text{Adam}}(\sigma_{\text{Adam}}) = (p, T, \sigma'_p)$ such that $\sigma_{\text{Adam}}(q_0) = (p, T)$ are the player and the set of topologies selected by **Adam** in state q_0 , and $\sigma'_p = \sigma_{\text{Adam}} \circ \gamma_{\text{obs}}^{-1}$ is the deviating strategy in \mathcal{G} induced by the deviation proposed in σ_{Adam} in \mathcal{H} .
- $\gamma_{\text{Snake}} : \Sigma_{\mathcal{H}}^{\text{Snake}} \rightarrow \text{Top}$ defined by $\gamma_{\text{Snake}}(\sigma_{\text{Snake}}) = \sigma_{\text{Snake}}(q_0)$ (recall that **Snake** only acts in q_0).

For readability, we omit the the subscript and write γ instead of $\gamma_{\text{obs}}, \gamma_{\text{Adam}}, \gamma_{\text{Eve}}, \gamma_{\text{Snake}}$. The correct subscript can be resolved from context. Intuitively, γ is the correspondence from strategies/histories/plays in \mathcal{H} to their counterpart in \mathcal{G} .

The connection between strategies and outcomes in \mathcal{H} and \mathcal{G} is formalized in the following lemma (see Appendix A.1 for the proof).

► **Lemma 14.** *Consider strategies $\sigma_{\text{Eve}} \in \Sigma_{\mathcal{H}}^{\text{Eve}}$, $\sigma_{\text{Adam}} \in \Sigma_{\mathcal{H}}^{\text{Adam}}$ and $\sigma_{\text{Snake}} \in \Sigma_{\mathcal{H}}^{\text{Snake}}$. Let $\sigma = \gamma(\sigma_{\text{Eve}})$, $(p, T, \sigma'_p) = \gamma(\sigma_{\text{Adam}})$ and $t = \gamma(\sigma_{\text{Snake}})$. Let $\rho = \text{out}_{\mathcal{H}}(\sigma_{\text{Eve}}, \sigma_{\text{Adam}}, \sigma_{\text{Snake}})$, $\pi' = \text{out}_{\mathcal{G}_t}(\sigma[p \mapsto \sigma'_p])$, and $\pi = \text{out}_{\mathcal{G}_t}(\sigma)$. Then $\text{proj}(\rho) = \pi'$. Furthermore, if **Adam** obeys **Eve** on ρ then $\text{proj}(\rho) = \pi = \pi'$.*

Objective for \mathcal{H} . As sketched in Section 4.2, the objective α in \mathcal{H} is constructed so that **Eve** and **Snake** can win if and only if there is a CNE in \mathcal{G} with winning topologies $(T_p)_{p \in \text{Pla}}$.

We define α as a conjunction of three conditions $\alpha = \{\rho \in q_0 \cdot Q^\omega \mid \psi_1(\rho) \wedge \psi_2(\rho) \wedge \psi_3(\rho)\}$, where the conditions are defined as follows. Consider a play $\rho = q_0, (s_0, p, T, t, b_0), (s_1, p, T, t, b_1), \dots$ of \mathcal{H} .

- $\psi_1(\rho) := t \in T$. That is, ψ_1 forces **Snake** to choose a topology from the set of topologies selected by **Adam**.
- $\psi_2(\rho) := (\text{obey}(\rho) \wedge t \in T_p) \rightarrow \text{proj}(\rho) \in \alpha_{t,p}$. That is, ψ_2 is satisfied if whenever **Adam** obeys **Eve** then Player p wins in any topology $t \in T_p$ selected by **Snake**.
- $\psi_3(\rho) := T_p \subsetneq T \rightarrow \text{proj}(\rho) \notin \alpha_{t,p}$. That is, ψ_3 is satisfied if whenever **Adam** tries to win in a strict superset of T_p , then Player p loses in the topology selected by **Snake**.

As mentioned in Section 4.2, it is not clear that α can be expressed as a single parity objective over $Q_{\mathcal{H}}$. Nonetheless, we prove that this is possible. The key observation is that the “postconditions” of ψ_2 and ψ_3 contradict, hence one of them must hold vacuously. This allows us to decouple the parity conditions for each of them and obtain a single parity objective that captures both, as follows.

For each objective $\alpha_{t,p}$ in \mathcal{G} we write $\alpha_{t,p} = \text{Parity}(\Omega_{t,p})$ such that $\Omega_{t,p} : \mathcal{S} \rightarrow \{0, \dots, d\}$ is the parity ranking function, where $d \in \mathbb{N}$. We define a new ranking function $\Omega : Q_{\mathcal{H}} \rightarrow \{0, \dots, d+1\}$, and show that $\alpha = \text{Parity}(\Omega)$.

First, observe that q_0 occurs only once in each play, so its parity rank has no effect. We arbitrarily set $\Omega(q_0) = 0$. Let $\rho \in q_0 \cdot Q^\omega$ be a play of \mathcal{H} and $(s, p, T, t, b), (s', p', T', t', b') \in \text{Inf}(\rho)$. It must be that $p = p'$, $T = T'$ and $t = t'$ since those are constant throughout the play, and $b = b'$ since it is either always **true** or from some point in ρ it turns into **false** and stays that way to the rest of the play.

34:12 Concurrent Games with Multiple Topologies

Let $q = (s, p, T, t, b) \in Q$. We define $\Omega(q)$ by cases according to p, T, t, b , and show that in each case, $\rho \in \alpha$ if and only if $\rho \in \text{Parity}(\Omega)$, concluding that $\alpha = \text{Parity}(\Omega)$. For a formula of the form $\psi = \varphi_1 \rightarrow \varphi_2$, we refer to φ_1 as the *precondition* of ψ , and φ_2 as the *postcondition* of ψ .

- $t \notin T$: In this case, if $q \in \text{Inf}(\rho)$ then ρ does not satisfy ψ_1 , thus, $\rho \notin \alpha$. We set $\Omega(q) = 1$ to get $\rho \notin \text{Parity}(\Omega)$.
- $t \in T$, $b = \mathbf{true}$, $t \in T_p$ and $T_p \subsetneq T$: In this case, if $q \in \text{Inf}(\rho)$ then ρ satisfies the preconditions of both ψ_2 and ψ_3 , but the postconditions of ψ_2 and ψ_3 contradict, thus, $\rho \notin \alpha$. We set $\Omega(q) = 1$ to get $\rho \notin \text{Parity}(\Omega)$.
- $t \in T$, $b = \mathbf{true} \wedge t \in T_p$ and $\neg(T_p \subsetneq T)$: In this case, if $q \in \text{Inf}(\rho)$, then $\rho \in \alpha \iff \text{proj}(\rho) \in \alpha_{t,p}$. So we set $\Omega(q) = \Omega_{t,p}(s)$, to apply the objective $\alpha_{t,p}$ over $\text{proj}(\rho)$.
- $t \in T$, $\neg(b = \mathbf{true} \wedge t \in T_p)$ and $T_p \subsetneq T$: In this case, if $q \in \text{Inf}(\rho)$, then $\rho \in \alpha \iff \text{proj}(\rho) \notin \alpha_{t,p}$. So we set $\Omega(q) = \Omega_{t,p}(s) + 1$, to apply the complement of the objective $\alpha_{t,p}$ over $\text{proj}(\rho)$.
- $t \in T$, $\neg(b = \mathbf{true} \wedge t \in T_p)$ and $\neg(T_p \subsetneq T)$: In this case, if $q \in \text{Inf}(\rho)$ then ψ_2 and ψ_3 are vacuously satisfied, and $\rho \in \alpha$. So we set $\Omega(q) = 0$ to get that $\rho \in \text{Parity}(\Omega)$.

We are now ready to characterize the existence of a CNE in \mathcal{G} by winning strategies in \mathcal{H} .

► **Lemma 15.** *Consider an MTG $\mathcal{G} = \langle \text{Pla}, \mathbf{S}, s_0, \text{Act}, \text{Top}, (\delta_t)_{t \in \text{Top}}, (\alpha_{t,p})_{t \in \text{Top}, p \in \text{Pla}} \rangle$. Let $(T_p)_{p \in \text{Pla}}$ be sets of topologies for each player and let \mathcal{H} be the corresponding partial-information game. There exists a strategy profile σ in \mathcal{G} such that σ is a CNE and for every $p \in \text{Pla}$ we have $\text{WinTop}_{\mathcal{G}}^p(\sigma) = T_p$ if and only if the following holds:*

$$\exists \sigma_{\text{Eve}} \in \Sigma_{\mathcal{H}}^{\text{Eve}} \forall \sigma_{\text{Adam}} \in \Sigma_{\mathcal{H}}^{\text{Adam}} \exists \sigma_{\text{Snake}} \in \Sigma_{\mathcal{H}}^{\text{Snake}} \text{out}_{\mathcal{H}}(\sigma_{\text{Eve}}, \sigma_{\text{Adam}}, \sigma_{\text{Snake}}) \in \alpha.$$

Proof. Assume σ is a CNE in \mathcal{G} such that for every $p \in \text{Pla}$, $\text{WinTop}_{\mathcal{G}}^p(\sigma) = T_p$, and fix $\sigma_{\text{Eve}} = \gamma^{-1}(\sigma)$ to be the corresponding strategy for **Eve** in \mathcal{H} . Consider a strategy $\sigma_{\text{Adam}} \in \Sigma_{\mathcal{H}}^{\text{Adam}}$ for **Adam**, and let $(p, T, \sigma'_p) = \gamma(\sigma_{\text{Adam}})$. We show that there exists a strategy $\sigma_{\text{Snake}} \in \Sigma_{\mathcal{H}}^{\text{Snake}}$ so that the outcome satisfies α . Recall that a strategy for **Snake** amounts to choosing a topology. We divide to cases according to the choice of T by **Adam**.

- If $\neg(T_p \subsetneq T)$, then ψ_3 is satisfied vacuously. Choose $t \in T$ for **Snake**, then ψ_1 is satisfied. If **Adam** does not obey **Eve** or $t \notin T_p$ then ψ_2 is vacuously satisfied. Otherwise, if **Adam** obeys **Eve** and $t \in T_p$, let $\rho = \text{out}_{\mathcal{H}}(\sigma_{\text{Eve}}, \sigma_{\text{Adam}}, \sigma_{\text{Snake}})$. In order to show that ψ_2 is satisfied we need to show that $\text{proj}(\rho) \in \alpha_{t,p}$. Let $\pi = \text{out}_{\mathcal{G}_t}(\sigma)$. Since $T_p = \text{WinTop}_{\mathcal{G}}^p(\sigma)$ and $t \in T_p$ we have that $\pi \in \alpha_{t,p}$. From Lemma 14 we have that $\text{proj}(\rho) = \pi$, so we get that $\text{proj}(\rho) \in \alpha_{t,p}$, as required.
- If $T_p \subsetneq T$, denote $T' = \text{WinTop}_{\mathcal{G}}^p(\sigma[p \mapsto \sigma'_p])$. Since σ is a CNE, we have that $\neg(T_p \subsetneq T')$, so $T \setminus T' \neq \emptyset$, as otherwise we would have that $T_p \subsetneq T \subseteq T'$. Choose $t \in T \setminus T'$ for **Snake**, then ψ_1 is satisfied. Let $\rho = \text{out}_{\mathcal{H}}(\sigma_{\text{Eve}}, \sigma_{\text{Adam}}, \sigma_{\text{Snake}})$, $\pi' = \text{out}_{\mathcal{G}_t}(\sigma[p \mapsto \sigma'_p])$ and $\pi = \text{out}_{\mathcal{G}_t}(\sigma)$. From Lemma 14 we have that $\text{proj}(\rho) = \pi'$ and if **Adam** obeys **Eve** then we have $\text{proj}(\rho) = \pi = \pi'$. Note that since $t \notin T' = \text{WinTop}_{\mathcal{G}}^p(\sigma[p \mapsto \sigma'_p])$ then $\pi' \notin \alpha_{t,p}$, so ψ_3 is satisfied. Finally, ψ_2 is satisfied vacuously since we cannot have $t \in T_p$ and that **Adam** obeys **Eve** simultaneously, as this would yield $T' = T_p = \text{WinTop}_{\mathcal{G}}^p(\sigma)$, but $t \notin T'$. We conclude that in all cases $\rho \in \alpha$, as required.

Conversely, assume that $\sigma_{\text{Eve}} \in \Sigma_{\mathcal{H}}^{\text{Eve}}$ is such that for every $\sigma_{\text{Adam}} \in \Sigma_{\mathcal{H}}^{\text{Adam}}$ there exists $\sigma_{\text{Snake}} \in \Sigma_{\mathcal{H}}^{\text{Snake}}$ such that $\text{out}_{\mathcal{H}}(\sigma_{\text{Eve}}, \sigma_{\text{Adam}}, \sigma_{\text{Snake}}) \in \alpha$. Let $\sigma = \gamma(\sigma_{\text{Eve}})$. We start by showing that for every $p \in \text{Pla}$ it holds that $\text{WinTop}_{\mathcal{G}}^p(\sigma) = T_p$. Indeed, let $p \in \text{Pla}$ and $t \in \text{Top}$.

If $t \in T_p$, take $\sigma_{\text{Adam}} \in \Sigma_{\mathcal{H}}^{\text{Adam}}$ that selects player p and $T = \{t\}$, and obeys Eve. The only strategy σ_{Snake} for Snake that satisfies ψ_1 is to select t . Let $\rho = \text{out}_{\mathcal{H}}(\sigma_{\text{Eve}}, \sigma_{\text{Adam}}, \sigma_{\text{Snake}})$. From ψ_2 we get that $\text{proj}(\rho) \in \alpha_{t,p}$, and by Lemma 14 we have $\text{proj}(\rho) = \text{out}_{\mathcal{G}_t}(\sigma)$. Thus, $t \in \text{WinTop}_{\mathcal{G}}^p(\sigma)$.

If $t \notin T_p$, take $\sigma_{\text{Adam}} \in \Sigma_{\mathcal{H}}^{\text{Adam}}$ that selects Player p and $T = T_p \cup \{t\}$, and obeys Eve. Since Adam obeys Eve, in order for ψ_1 , ψ_2 and ψ_3 to be satisfied, Snake must choose t , otherwise both preconditions of ψ_2 and ψ_3 hold, which means that in order to win we must have both $\text{proj}(\rho) \in \alpha_{t,p}$ (by ψ_2) and $\text{proj}(\rho) \notin \alpha_{t,p}$ (by ψ_3), which cannot hold. Thus, Snake chooses t , and from Lemma 14 we have $\text{proj}(\rho) = \text{out}_{\mathcal{G}_t}(\sigma)$. By ψ_3 we have $\text{proj}(\rho) \notin \alpha_{t,p}$, so $\text{out}_{\mathcal{G}_t}(\sigma) \notin \alpha_{t,p}$. Thus $t \notin \text{WinTop}_{\mathcal{G}}^p(\sigma)$. Therefore, $\text{WinTop}_{\mathcal{G}}^p(\sigma) = T_p$.

It remains to show that σ is a CNE. Assume by way of contradiction that there exists a player $p \in \text{Pla}$ with a beneficial deviation $\sigma'_p \in \Sigma_{\mathcal{G}}^p$. That is, $T' = \text{WinTop}_{\mathcal{G}}^p(\sigma[p \mapsto \sigma'_p])$ satisfies $T_p \subsetneq T'$. We will construct a strategy of Adam such that every strategy of Snake is losing, thereby reaching a contradiction. Let $T = T_p \cup \{t'\}$ for some $t' \in T \setminus T_p$ and fix $\sigma_{\text{Adam}} = \gamma^{-1}(p, T, \sigma'_p)$. Consider a strategy σ_{Snake} , denote $t = \gamma(\sigma_{\text{Snake}})$ and let $\rho = \text{out}_{\mathcal{H}}(\sigma_{\text{Eve}}, \sigma_{\text{Adam}}, \sigma_{\text{Snake}})$. By Lemma 14 we have $\text{proj}(\rho) = \text{out}_{\mathcal{G}_t}(\sigma[p \mapsto \sigma'_p])$, and because $t \in T \subseteq \text{WinTop}_{\mathcal{G}}^p(\sigma[p \mapsto \sigma'_p])$ it holds that $\text{proj}(\rho) \in \alpha_{t,p}$. However, $T_p \subsetneq T$, so ψ_3 is violated, and $\rho \notin \alpha$, which is a contradiction. We conclude that σ is a CNE. \blacktriangleleft

Using Lemma 15 we can decide whether a given MTG \mathcal{G} has a CNE, by iterating over all possible sets of candidate winning topologies $(T_p)_{p \in \text{Pla}}$, and repeatedly applying the reduction, and using the decision procedure of Theorem 13. It remains to analyze the complexity of this procedure.

To this end, observe that the size of \mathcal{H} is polynomial in the size of \mathcal{G} . Indeed, $|Q| \leq |S| \cdot |\text{Pla}| \cdot |\mathcal{T}| \cdot |\text{Top}| \cdot 2$ where $|\mathcal{T}| \leq 2|\text{Pla}||\text{Top}|$. and the description of the actions is also polynomial in that of \mathcal{G} (note that Eve has exponentially more actions than each player in \mathcal{G} , but the overall description of the transition table in \mathcal{G} is similarly exponential, cf. Remark 2).

Finally, by Theorem 13, solving \mathcal{H} takes double-exponential time in $|\mathcal{G}|$, and we have a single-exponential number of iterations, so the overall complexity remains double-exponential time in $|\mathcal{G}|$. This completes the proof of Theorem 12.

► Remark 16 (Lower bounds and improving the upper bound). We do not have a lower bound for the 2-EXPTIME complexity of Theorem 12. Indeed, we suspect that this bound can be lowered. This is due in part to the fact that game \mathcal{H} we construct does not utilize the full scope of Theorem 13 from [10]. Unfortunately, the decision procedure in [10] goes through three nontrivial reductions, one of which involves Safra's determinization, that is notoriously difficult to analyze: The first reduction [9, 10] transforms the objective to a visible objective for Adam which involves the determinization of a parity automaton. The second reduction [10] reduces the three-player partial-information game into a two-player partial-information game. The third reduction uses the results of [22] to reduce the two-player partial-information game to a two-player perfect-information game.

Therefore, it is likely that improving the bound (if indeed possible) will involve devising an ad-hoc procedure, possibly using some key ideas from [9, 10, 22].

5 Existence of Greedy NE is Decidable

We now turn our attention to Greedy NE (GNE). Recall that a greedy beneficial deviation is one that wins in a previously-losing topology, even at the cost of losing in previously-winning topologies. That is, given an MTG $\mathcal{G} = \langle \text{Pla}, S, s_0, \text{Act}, \text{Top}, (\delta_t)_{t \in \text{Top}}, (\alpha_{t,p})_{t \in \text{Top}, p \in \text{Pla}} \rangle$, a profile $\sigma \in \Sigma_{\mathcal{G}}^{\text{Pla}}$ is a GNE if for every $p \in \text{Pla}$, $\sigma'_p \in \Sigma_{\mathcal{G}}$ and $t \in \text{Top}$, if $p \in \text{Win}_{\mathcal{G}_t}(\sigma[p \mapsto \sigma'_p])$ then $p \in \text{Win}_{\mathcal{G}_t}(\sigma)$.

Intuitively, reasoning in the greedy approach is much less delicate than the conservative approach, since a deviating player need not concern itself with keeping the current winning topologies. As we show in the following, this allows for an exponentially faster solution.

► **Theorem 17.** *The problem of deciding, given an MTG \mathcal{G} , whether there exists a GNE in \mathcal{G} is in EXPTIME.*

Similarly to Section 4, our approach is to reduce the problem at hand to solving a partial-information game. In the greedy setting, however, it suffices to use two-player games. Specifically, we employ the following result from [9].

► **Theorem 18.** *Let $\mathcal{G} = \langle \text{Pla}, \text{S}, s_0, \text{Act}, \delta, (\mathcal{O}_p)_{p \in \text{Pla}} \rangle$ with $\text{Pla} = \{1, 2\}$. Let $\alpha \subseteq \text{S}^\omega$ be a parity objective. The problem of deciding whether $\exists \sigma_1 \in \Sigma_{\mathcal{G}}^1 \forall \sigma_2 \in \Sigma_{\mathcal{G}}^2 \text{out}_{\mathcal{G}}(\sigma_1, \sigma_2) \in \alpha$ is EXPTIME-complete.*

We sketch the proof of Theorem 17. The complete construction and analysis are detailed in Appendix B.

Proof sketch. As in Section 4.3, we first fix a set of “intended” winning topologies $T_p \subseteq \text{Top}$ for each player $p \in \text{Pla}$. Then, we ask whether \mathcal{G} admits a GNE σ in which $\text{WinTop}_{\mathcal{G}}^p(\sigma) = T_p$ for every $p \in \text{Pla}$. We then construct a 2-player partial-information game whose players are Eve, Adam, where Eve again controls the coalition of all players.

The behaviour of Adam is different than in the conservative setting. Here, Adam starts by choosing a deviating player $p \in \text{Pla}$ and a *single topology* $t \in \text{Top}$ where p attempts to win. The topology t is unobservable by Eve. The observations sets of Eve and Adam are again only the current state of \mathcal{G} . Then, the game is played on topology t with Eve suggesting an action profile, and Adam possibly deviating with Player p .

The objective for Eve now comprises two conditions:

- ψ_1 requires that whenever Adam obeys Eve and $t \in T_p$, the outcome is winning for Player p in \mathcal{G}_t .
- ψ_2 requires that if $t \notin T_p$, then Player p loses in \mathcal{G}_t .

Intuitively, Adam tries to cause Player p to win in a new topology t in which Player p is not intended to win, while Eve is trying to prevent Player p from achieving this, provided that Player p is actually deviating. Note that Eve must do this without knowing which topology is chosen, nor which player deviates (if at all). ◀

6 Discussion, Extensions and Future Work

We introduced MTGs and notions of NE pertaining to them, and showed that deciding whether an MTG admits either notion is decidable (in 2-EXPTIME for CNE and in EXPTIME for GNE). We have also explored the relationships and properties of these notions of NE. We now turn to explore several extensions, and remark about future research directions.

Social optimum. A standard solution concept for concurrent games, apart from NE, is *social optimum*, namely what is the maximum welfare the player can obtain by cooperating. Since in MTGs the winning sets of topologies may be incomparable, we formulate this as follows: given sets $(T_p)_{p \in \text{Pla}}$, is there a strategy profile σ such that $\text{WinTop}_{\mathcal{G}}^p(\sigma) = T_p$ for every $p \in \text{Pla}$?

Fortunately, the techniques we developed enable us to readily solve this problem. Indeed, we can modify the reduction used to decide the existence of GNE (Section 5) so that Adam chooses a player and a topology, but does not attempt to deviate and has no further effect on the game. Intuitively, Adam “challenges” Eve to show that the winning topologies for the players are exactly the intended ones. The complexity of this approach remains EXPTIME.

Lower bounds. As discussed in Remark 16, we do not provide lower bounds for our results. Trivial lower bounds on the existence of CNE and GNE can be obtained from those of NE existence in concurrent games, namely $P_{||}^{NP}$ -hardness [6]. This, however, is unlikely to be tight. A central open challenge is to determine the exact complexity of CNE and GNE existence in MTGs.

Additional notions of equilibria. The notions we propose, namely CNE and GNE, lie on two extremities: in the conservative setting a deviation is very strict, and in the greedy setting it is very lax. Generally, one can obtain a notion of equilibrium using any binary relation on 2^{Top} , which describes what the beneficial deviations are for each player. Moreover, different players can have different relations.

Of particular interest is a quantitative notion of NE, whereby a player deviates if she can increase the *number* of her winning topologies. This notion is fundamentally different from CNE and GNE, as it is not based on set containment, which is key to the correctness of our approach.

Succinct representation of topologies. A central motivation for MTGs, demonstrated in Example 1 and in Section 3.1 concerns process symmetry. There, from a game with k players, we construct an MTG with $k!$ topologies. However, these topologies can be succinctly represented by computing them on-the-fly. An interesting direction for future work is to determine whether we can devise a symbolic approach that is able to handle such MTGs without incurring an exponential blowup.

Logic for partial information games. Another approach to solve the CNE and GNE existence problems might be to formulate those problems with a logic for partial information games [3, 16, 20]. The most promising of those is [3], as it is the most expressive and the complexity of different fragments has been studied. From the complexity of the decision procedures for those logics, it not likely that this approach can be used to lower the EXPTIME complexity for GNE and the 2-EXPTIME complexity for CNE.

References

- 1 Shaull Almagor. Process symmetry in probabilistic transducers. In *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, 2020.
- 2 Shaull Almagor, Guy Avni, and Orna Kupferman. Repairing multi-player games. In *26th International Conference on Concurrency Theory (CONCUR 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
- 3 Raphaël Berthon, Bastien Maubert, Aniello Murano, Sasha Rubin, and Moshe Y Vardi. Strategy logic with imperfect information. *ACM Transactions on Computational Logic (TOCL)*, 22(1):1–51, 2021.
- 4 Udi Boker. Why these automata types? In *LPAR*, volume 18, pages 143–163, 2018.
- 5 Patricia Bouyer, Nicolas Markey, and Steen Vester. Nash equilibria in symmetric graph games with partial observation. *Information and Computation*, 254:238–258, 2017.

- 6 Patricia P Bouyer, Romain Brenguier, and Nicolas N Markey. Pure nash equilibria in concurrent games. *Logical methods in computer science*, 2015.
- 7 Felix Brandt, Felix Fischer, and Markus Holzer. Equilibria of graphical games with symmetries. *Theoretical Computer Science*, 412(8-10):675–685, 2011.
- 8 Romain Brenguier, Arno Pauly, Jean-François Raskin, and Ocan Sankur. Admissibility in games with imperfect information. In *CONCUR 2017-28th International Conference on Concurrency Theory*, volume 85, pages 2:1–2:23. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017.
- 9 Krishnendu Chatterjee and Laurent Doyen. The complexity of partial-observation parity games. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 1–14. Springer, 2010.
- 10 Krishnendu Chatterjee and Laurent Doyen. Games with a weak adversary. In *International Colloquium on Automata, Languages, and Programming*, pages 110–121. Springer, 2014.
- 11 Krishnendu Chatterjee, Thomas A Henzinger, and Nir Piterman. Strategy logic. *Information and Computation*, 208(6):677–693, 2010.
- 12 Edmund M. Clarke, Reinhard Enders, Thomas Filkorn, and Somesh Jha. Exploiting symmetry in temporal logic model checking. *Formal methods in system design*, 9(1):77–104, 1996.
- 13 Luca De Alfaro, Thomas A Henzinger, and Orna Kupferman. Concurrent reachability games. *Theoretical computer science*, 386(3):188–217, 2007.
- 14 Aldric Degorre, Laurent Doyen, Raffaella Gentilini, Jean-François Raskin, and Szymon Toruńczyk. Energy and mean-payoff games with imperfect information. In *International Workshop on Computer Science Logic*, pages 260–274. Springer, 2010.
- 15 E Allen Emerson and A Prasad Sistla. Symmetry and model checking. *Formal methods in system design*, 9(1):105–131, 1996.
- 16 Bernd Finkbeiner and Sven Schewe. Coordination logic. In *International Workshop on Computer Science Logic*, pages 305–319. Springer, 2010.
- 17 Nicholas Ham. Notions of anonymity, fairness and symmetry for finite strategic-form games. *arXiv preprint*, 2013. [arXiv:1311.4766](https://arxiv.org/abs/1311.4766).
- 18 C Norris Ip and David L Dill. Better verification through symmetry. In *Computer Hardware Description Languages and their Applications*, pages 97–111. Elsevier, 1993.
- 19 Anthony W Lin, Truong Khanh Nguyen, Philipp Rümmer, and Jun Sun. Regular symmetry patterns. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 455–475. Springer, 2016.
- 20 Bastien Maubert. *Logical foundations of games with imperfect information: uniform strategies*. PhD thesis, Université Rennes 1, 2014.
- 21 Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, 2007. doi:10.1017/CB09780511800481.
- 22 Jean-François Raskin, Thomas A Henzinger, Laurent Doyen, and Krishnendu Chatterjee. Algorithms for omega-regular games with imperfect information. *Logical Methods in Computer Science*, 3, 2007.
- 23 Noah Daniel Stein. *Exchangeable equilibria*. PhD thesis, Massachusetts Institute of Technology, 2011.
- 24 Fernando A Tohmé and Ignacio D Viglizzo. Structural relations of symmetry among players in strategic games. *International Journal of General Systems*, 48(4):443–461, 2019.
- 25 M Ummels and DK Wojtczak. The complexity of nash equilibria in stochastic multiplayer games. *Logical Methods in Computer Science*, 2010.
- 26 Steen Vester. *Symmetric Nash Equilibria*. PhD thesis, Master’s thesis, ENS Cachan, 2012.

A Proofs

A.1 Proof of Lemma 14

Proof. We prove by induction that for every $k \geq 1$, $\text{proj}(\rho_{\leq k+1}) = \pi'_{\leq k}$, and if **Adam** obeys **Eve** then $\text{proj}(\rho_{\leq k+1}) = \pi'_{\leq k} = \pi_{\leq k}$. For $k = 1$, $\rho_{\leq 2} = q_0, (s_0, p, t, T, b_0)$ and $\pi'_{\leq 1} = \pi_{\leq 1} = s_0$ and we have that $\text{proj}(\rho_{\leq k+1}) = \pi'_{\leq k}$. Assuming that $\text{proj}(\rho_{\leq k+1}) = \pi'_{\leq k}$ for $k \geq 1$, the next state of $\text{proj}(\rho)$ will depend on the transition function δ_t and action profile $\sigma[p \mapsto \sigma'_p](\pi'_{\leq k})$ from the way γ and the transitions of \mathcal{H} are defined, and the next state in π' will also depend on the same transition function and action profile. Thus, it holds that $\text{proj}(\rho_{\leq k+2}) = \pi'_{\leq k+1}$. Farther more, if **Adam** obeys **Eve** then in every step the action that **Adam** takes is identical to the action that **Eve** suggests for Player p , so we have that $\sigma[p \mapsto \sigma'_p](\pi'_{\leq k}) = \sigma(\pi'_{\leq k})$, and $\pi_{\leq k+1} = \pi'_{\leq k+1}$, thus, $\text{proj}(\rho_{\leq k+2}) = \pi_{\leq k+1} = \pi'_{\leq k+1}$. ◀

B Proof of Theorem 17

Consider an MTG $\mathcal{G} = \langle \text{Pla}, \text{S}, s_0, \text{Act}, \text{Top}, (\delta_t)_{t \in \text{Top}}, (\alpha_{t,p})_{t \in \text{Top}, p \in \text{Pla}} \rangle$. For every Player $p \in \text{Pla}$ fix $T_p \subseteq \text{Top}$ to be the intended set of winning topologies.

Game construction. We construct a two-player partial-information game \mathcal{H} with the following components. The players are **Eve** and **Adam**. The states of \mathcal{H} are $Q_{\mathcal{H}} = \{q_0\} \cup Q$ such that q_0 is a designated initial state and $Q = \text{S} \times \text{Pla} \times \text{Top} \times \{\text{true}, \text{false}\}$ is described in the following. A state $(s, p, t, b) \in Q$ comprises of $s \in \text{S}$ which tracks the state of \mathcal{G} , a player $p \in \text{Pla}$ that is controlled by **Adam**, a topology $t \in \text{Top}$ that **Adam** picks, and a bit $b \in \{\text{true}, \text{false}\}$ which tracks whether **Adam** obeys **Eve**.

We now turn to define the transitions of \mathcal{H} . The actions are defined implicitly by the transitions. From state q_0 , **Adam** selects a player $p \in \text{Pla}$ to control and a topology $t \in \text{Top}$ that \mathcal{G} will be played in. Then, \mathcal{H} transitions to state $(s_0, p, t, \text{true}) \in Q$. Henceforth, p and t remain fixed throughout the play. From state $(s, p, t, b) \in Q$, **Eve** chooses an action profile $\alpha \in \text{Act}^{\text{Pla}}$, and **Adam** selects an action $a'_p \in \text{Act}$ and \mathcal{H} transitions to state (s', p, t, b') such that $s' = \delta_t(s, \alpha[p \mapsto a'_p])$, and $b' = b \wedge (a'_p = a_p)$.

The observation sets for the players, proj and obey are defined similarly as Section 4.3. Correspondence between \mathcal{H} and \mathcal{G} , $\gamma_{\text{obs}}, \gamma_{\text{Eve}}$ is defined in the same way as in Section 4.3, and $\gamma_{\text{Adam}} : \Sigma_{\mathcal{H}}^{\text{Adam}} \rightarrow \bigcup_{p \in \text{Pla}} \{p\} \times \text{Top} \times \Sigma_{\mathcal{H}}^p$ is defined for $\gamma(\sigma_{\text{Adam}}) = (p, t, \sigma'_p)$ such that (p, t) are the player and topology selected by σ_{Adam} in state q_0 and $\sigma'_p = \sigma_{\text{Adam}} \circ \gamma_{\text{obs}}^{-1}$.

The connection between strategies and outcomes in \mathcal{H} and \mathcal{G} is formalized in the following lemma whose proof is similar to that of Lemma 14.

► **Lemma 19.** *Consider strategies $\sigma_{\text{Eve}} \in \Sigma_{\mathcal{H}}^{\text{Eve}}$ and $\sigma_{\text{Adam}} \in \Sigma_{\mathcal{H}}^{\text{Adam}}$. Let $\sigma = \gamma(\sigma_{\text{Eve}})$ and $(p, t, \sigma'_p) = \gamma(\sigma_{\text{Adam}})$. Let $\rho = \text{out}_{\mathcal{H}}(\sigma_{\text{Eve}}, \sigma_{\text{Adam}})$, $\pi' = \text{out}_{\mathcal{G}_t}(\sigma[p \mapsto \sigma'_p])$ and $\pi = \text{out}_{\mathcal{G}_t}(\sigma)$. Then, $\text{proj}(\rho) = \pi'$. Furthermore, if **Adam** obeys **Eve** on ρ then $\text{proj}(\rho) = \pi = \pi'$.*

Objective for \mathcal{H} . Let $\rho = q_0 \cdot (s_0, p, t, b_0) \cdot (s_1, p, t, b_1) \cdot \dots$ be a play in \mathcal{H} . The objective α is such that $\rho \in \alpha \iff \psi_1(\rho) \wedge \psi_2(\rho)$, where

- $\psi_1(\rho) := (\text{obey}(\rho) \wedge t \in T_p) \rightarrow \text{proj}(\rho) \in \alpha_{t,p}$.
- $\psi_2(\rho) := t \notin T_p \rightarrow \text{proj}(\rho) \notin \alpha_{t,p}$.

α can be expressed as a parity objective as follows. For every $t \in \text{Top}$, $p \in \text{Pla}$, let $\Omega_{t,p} : \text{S} \rightarrow \{0, \dots, d_{t,p}\}$ be the priority function for the parity objective $\alpha_{t,p}$ in \mathcal{G} . We construct a priority function $\Omega : Q_{\mathcal{H}} \rightarrow \{0, \dots, d\}$ such that $d = \max\{d_{t,p} + 1 \mid t \in \text{Top}, p \in \text{Pla}\}$. We set $\Omega(q_0) = 0$ and for state $q = (s, p, t, b) \in Q$ we have

$$\Omega(q) = \begin{cases} \Omega_{t,p}(s) + 1 & t \notin T_p \\ \Omega_{t,p}(s) & b \wedge t \in T_p \\ \Omega(q) = 0 & \neg b \wedge t \in T_p \end{cases}$$

If $t \notin T_p$, then, according to α , $\rho \in \alpha$ if and only if $\text{proj}(\rho) \notin \alpha_{t,p}$. This is achieved by adding 1 to $\Omega_{t,p}$ which gives us the complement of $\alpha_{t,p}$. The case where **Adam** obeys **Eve** and $t \in T_p$ is captured in the second case, where $\rho \in \alpha$ if and only if $\text{proj}(\rho) \in \alpha_{t,p}$. This is achieved by setting Ω to be the same as $\Omega_{t,p}$. In the last case, none of the preconditions of ψ_1 and ψ_2 hold, so $\rho \in \alpha$. This is achieved by setting Ω to 0, such that every such play will satisfy the objective.

► **Lemma 20.** *There exists a GNE $\sigma \in \Sigma_{\mathcal{G}}$ in \mathcal{G} with $\text{WinTop}_{\mathcal{G}}^p(\sigma) = T_p$ for every $p \in \text{Pla}$, if and only if $\exists \sigma_{\text{Eve}} \in \Sigma_{\mathcal{H}}^{\text{Eve}} \forall \sigma_{\text{Adam}} \in \Sigma_{\mathcal{H}}^{\text{Adam}} \text{out}_{\mathcal{H}}(\sigma_{\text{Eve}}, \sigma_{\text{Adam}}) \in \alpha$.*

Proof. Let $\sigma \in \Sigma_{\mathcal{G}}$ be a GNE with $\text{WinTop}_{\mathcal{G}}^p(\sigma) = T_p$ for every $p \in \text{Pla}$. Let $\sigma_{\text{Eve}} \in \Sigma_{\mathcal{H}}^{\text{Eve}}$ be the corresponding strategy for σ , and let $\sigma_{\text{Adam}} \in \Sigma_{\mathcal{H}}^{\text{Adam}}$ be some strategy for **Adam** that corresponds to (p, t, σ'_p) . Let $\rho = \text{out}_{\mathcal{H}}(\sigma_{\text{Eve}}, \sigma_{\text{Adam}})$. If $\text{obey}(\rho) \wedge t \in T_p$, then from Lemma 19 we have that $\text{proj}(\rho) = \text{out}_{\mathcal{G}_t}(\sigma)$, and since $t \in T_p = \text{WinTop}_{\mathcal{G}}^p(\sigma)$ then $\text{out}_{\mathcal{G}_t}(\sigma) \in \alpha_{t,p}$. Thus, ψ_1 is satisfied by ρ . If $t \notin T_p$ then from Lemma 19 we have that $\text{proj}(\rho) = \text{out}_{\mathcal{G}_t}(\sigma[p \mapsto \sigma'_p])$ and since **Player** p is losing in t when \mathcal{G} is played with σ and σ is a GNE, then $\text{out}_{\mathcal{G}_t}(\sigma[p \mapsto \sigma'_p]) \notin \alpha_{t,p}$. Thus, ψ_2 is satisfied and $\rho \in \alpha$.

Conversely, let $\sigma_{\text{Eve}} \in \Sigma_{\mathcal{H}}^{\text{Eve}}$ be such that for any $\sigma_{\text{Adam}} \in \Sigma_{\mathcal{H}}^{\text{Adam}}$ we have $\text{out}_{\mathcal{H}}(\sigma_{\text{Eve}}, \sigma_{\text{Adam}}) \in \alpha$. Let $\sigma \in \Sigma_{\mathcal{G}}$ correspond to σ_{Eve} . We show that σ is a GNE. First, we show that for every $p \in \text{Pla}$, $\text{WinTop}_{\mathcal{G}}^p(\sigma) = T_p$. Let $t \in \text{Top}$ and $p \in \text{Pla}$. Take $\sigma_{\text{Adam}} \in \Sigma_{\mathcal{H}}^{\text{Adam}}$ that corresponds to (p, t, σ_p) where σ_p is the strategy assigned to p in σ . Let $\rho_t = \text{out}_{\mathcal{G}_t}(\sigma)$ and $\rho = \text{out}_{\mathcal{H}}(\sigma_{\text{Eve}}, \sigma_{\text{Adam}})$. We have that $\rho \in \alpha$. Since **Adam** obeys **Eve** on ρ , from Lemma 19 we have that $\text{proj}(\rho) = \rho_t$. If $t \in T_p$ then from ψ_1 we get that $\rho_t = \text{proj}(\rho) \in \alpha_{t,p}$, thus, $t \in \text{WinTop}_{\mathcal{G}}^p(\sigma)$. If $t \notin T_p$ then from ψ_2 we get that $\rho_t = \text{proj}(\rho) \notin \alpha_{t,p}$, thus, $t \notin \text{WinTop}_{\mathcal{G}}^p(\sigma)$. So we get that $\text{WinTop}_{\mathcal{G}}^p(\sigma) = T_p$. Now, we show that σ is a GNE. Let $p \in \text{Pla}$, $\sigma'_p \in \Sigma_{\mathcal{G}}^p$ and $t \in \text{Top}$ such that $t \notin T_p$. Let $\sigma_{\text{Adam}} \in \Sigma_{\mathcal{H}}^{\text{Adam}}$ correspond to (p, t, σ'_p) , and let $\rho = \text{out}_{\mathcal{H}}(\sigma_{\text{Eve}}, \sigma_{\text{Adam}})$. We have that $\rho \in \alpha$, thus, since $t \notin T_p$ then $\text{proj}(\rho) \notin \alpha_{t,p}$. From Lemma 19 we have that $\rho'_t = \text{out}_{\mathcal{G}_t}(\sigma[p \mapsto \sigma'_p]) = \text{proj}(\rho) \notin \alpha_{t,p}$, thus, $t \notin \text{WinTop}_{\mathcal{G}_t}^p(\sigma[p \mapsto \sigma'_p]) = T_p$, so σ is a GNE. ◀

The algorithm for solving the GNE existence problem is, for each $(T_p)_{p \in \text{Pla}} \in (2^{\text{Top}})^{\text{Pla}}$ we construct \mathcal{H} from \mathcal{G} and $(T_p)_{p \in \text{Pla}}$, and check if there exists $\sigma_{\text{Eve}} \in \Sigma_{\mathcal{H}}^{\text{Eve}}$ such that for every $\sigma_{\text{Adam}} \in \Sigma_{\mathcal{H}}^{\text{Adam}}$, $\text{out}_{\mathcal{H}}(\sigma_{\text{Eve}}, \sigma_{\text{Adam}}) \in \alpha$, if there exists such σ_{Eve} , then according to Lemma 20 is corresponding strategy profile is a GNE, then we return it. If we went through all $(T_p)_{p \in \text{Pla}} \in (2^{\text{Top}})^{\text{Pla}}$, then return that there does not exist a GNE in \mathcal{G} .

The size of \mathcal{H} is polynomial in the size of \mathcal{G} . We copy each $s \in \mathcal{S}$ for every combination of $p \in \text{Pla}$, $t \in \text{Top}$, $b \in \{\text{true}, \text{false}\}$, so we get $|\mathcal{Q}_{\mathcal{H}}| = 2 \cdot |\mathcal{S}| \cdot |\text{Pla}| \cdot |\text{Top}| + 1$, which is polynomial in the size of \mathcal{G} . The number of actions in \mathcal{H} is also polynomial in the number of enabled actions in \mathcal{G} (similarly to the analysis in Section 4.3).

The algorithm performs at most $2^{|\text{Top}| \cdot |\text{Pla}|}$ iterations, which is exponential in $|\mathcal{G}|$. In each iteration we solve \mathcal{H} with size that is polynomial in $|\mathcal{G}|$, so according to Theorem 18 this takes exponential time in $|\mathcal{G}|$, so the GNE existence problem is in EXPTIME.

Generalised Multiparty Session Types with Crash-Stop Failures

Adam D. Barwell  


Imperial College London, UK

Alceste Scalas  

DTU Compute, Technical University of Denmark, Lyngby, Denmark

Nobuko Yoshida  

Imperial College London, UK

Fangyi Zhou  

Imperial College London, UK

Abstract

Session types enable the specification and verification of communicating systems. However, their theory often assumes that processes never fail. To address this limitation, we present a generalised multiparty session type (MPST) theory with *crash-stop failures*, where processes can crash arbitrarily.

Our new theory validates more protocols and processes w.r.t. previous work. We apply minimal syntactic changes to standard session π -calculus and types: we model crashes and their handling semantically, with a generalised MPST typing system parametric on a behavioural safety property. We cover the spectrum between fully reliable and fully unreliable sessions, via *optional reliability assumptions*, and prove type safety and protocol conformance in the presence of crash-stop failures.

Introducing crash-stop failures has non-trivial consequences: writing correct processes that handle all crash scenarios can be difficult. Yet, our generalised MPST theory allows us to tame this complexity, via model checking, to validate whether a multiparty session satisfies desired behavioural properties, e.g. deadlock-freedom or liveness, even in presence of crashes. We implement our approach using the mCRL2 model checker, and evaluate it with examples extended from the literature.

2012 ACM Subject Classification Theory of computation \rightarrow Distributed computing models; Theory of computation \rightarrow Process calculi; Software and its engineering \rightarrow Model checking

Keywords and phrases Session Types, Concurrency, Failure Handling, Model Checking

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.35

Related Version *Full Version*: <https://doi.org/10.48550/arXiv.2207.02015>

Supplementary Material *Software (Source Code)*: <https://github.com/alcestes/mpstk-crash-stop>, archived at `swh:1:dir:dd8b3c8c6f5f16e5405c0a697f1acd72e3868514`

Funding Work supported by: EU Horizon 2020 project 830929; EPSRC grants EP/K011715/1, EP/K034413/1, EP/L00058X/1, EP/N027833/1, EP/N028201/1, EP/T006544/1, EP/T014709/1, EP/V000462/1, and NCSS/EPSRC VeTSS; Danmarks Industriens Fond 2020-0489.

1 Introduction

Multiparty session types (MPST) [20] provide a typing discipline for message-passing processes. The theory ensures well-typed processes enjoy desirable properties, a.k.a. *the Session Theorems*: type safety (processes communicate without errors), protocol conformance (a.k.a. *session fidelity*, processes behave according to their types), deadlock-freedom (processes do not get stuck), and liveness (input/output actions eventually succeed). Researchers devote significant effort into integrating session types in programming languages and tools [16].



© Adam D. Barwell, Alceste Scalas, Nobuko Yoshida, and Fangyi Zhou;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 35; pp. 35:1–35:25

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A common assumption in session type theory is that everything is reliable and there are no failures, which is often unrealistic in real-world systems. So, we pose a question: how can we better model systems *with failures*, and make session types less idealistic?

In this paper, we take steps towards bridging the gap between theory and practice with a new *generalised* multiparty session type theory that models failures with *crash-stop* semantics [5, §2.2]: processes may crash, and crashed processes stop interacting with the world. This model is standard in distributed systems, and is used in related work on session types with error-handling capabilities [33,34]. However, unlike previous work, we allow *any* process to crash arbitrarily, and support optional assumptions on non-crashing processes.

In our new theory, we add crashing and crash handling semantics to processes and session types. With minimal changes to the standard surface syntax, we model a variety of subtle, complex behaviours arising from unreliable communicating processes. An active process P may crash arbitrarily, and a process Q interacting with P might need to be prepared to handle possible crashes. Messages sent from Q to a crashed P are lost – but if Q tries to receive from P , then Q can detect that P has crashed, and take a crash handling branch. Meanwhile, another process R may (or may not) have detected P 's crash, and may be handling it – and in either case, any interaction between Q and R should remain correct.

Our MPST theory is generalised in two aspects: (1) we introduce *optional reliability assumptions*, so we can model a mixture of reliable and unreliable communicating peers; and (2) our type system is parametric on a type-level behavioural property φ which can be instantiated as safety, deadlock freedom, liveness, *etc.* (in the style of [29]), while accounting for potential crashes. We prove *session fidelity*, showing how type-level properties transfer to well-typed processes; we also prove that our new theory satisfies other Session Theorems of MPST, while (unlike previous work) being resilient to arbitrary crash-stop failures.

With *optional reliability assumptions*, one may declare that some peers will never crash for the duration of the protocol. Such optional assumptions allow for simplifying protocols and programs: if a peer is assumed reliable, the other peers can interact with it without needing to handle its crashes. By making such assumptions explicit and customisable, our theory supports a *spectrum* of scenarios ranging from only having sessions with reliable peers (thus subsuming classic MPST works [20,29]), to having no reliable peers at all.

As in the real world, a system with crash-stop failures can have subtle complex behaviours; hence, writing protocols and processes where all possible crash scenarios are correctly handled can be hard. This highlights a further benefit of our generalised theory: we formalise our behavioural properties as modal μ -calculus formulæ, and verify them with a model checker. To show the feasibility of our approach, we present an accompanying tool, utilising the mCRL2 model checker [4], for verifying session properties under optional reliability assumptions.

Overview. Session typing systems assign *session types* (a.k.a. local types) to communication channels, used by processes to send and receive messages. In essence, a session type describes a *protocol*: how a *role* is expected to interact with other roles in a multiparty session. The type system checks whether a process implements desired protocols.

As an example, consider a simple Domain Name System (DNS) scenario: a client p queries a server q for an IP address of a host name. With classic session types (without crashes), we use the type $T_p = q \oplus \text{req}.q \& \text{res}$ to represent the client p 's behaviour: first sending (\oplus) a **request** message to server q , and then receiving ($\&$) a **response** from q . The server implements a dual type $T_q = p \& \text{req}.p \oplus \text{res}$, who receives a **request** from client p , and then sends a **response** to p . We can write a process $Q = s[q][p] \& \text{req}.s[q][p] \oplus \text{res}.\mathbf{0}$ for the server. Using T_q , we type-check the channel (a.k.a. session endpoint) $s[q]$, where Q plays the role q on session s . Here, Q type-checks – it uses channel $s[q]$ correctly, according to type T_q .



■ **Figure 1** Transition systems (based on Def. 7, with labels omitted) generated from the DNS examples. Left: without crashes/handling. Right: with crashes (for q) and crash handling.

In this work, we augment the classic session types theory by introducing process failures with *crash-stop* semantics [5, §2.2]. We adopt the following failure model: (1) processes have *crash-stop* failures, i.e. they may crash and do not recover; (2) communication channels deliver messages in order, without losses (unless the recipient has crashed); (3) each process has a failure detector [11], so a process trying to receive from a crashed peer accurately detects the crash. The combination of (1), (2), and (3) is called the *crash-fail* model in [5, §2.6.2].

We now revise our DNS example in the presence of failures. Let us assume that the server q may crash, whereas the client p remains reliable. The client p may now send its **request** to a new *failover server* r (assumed reliable for simplicity). We represent this scenario by a type for the new failover server T'_r , and a new branch in T'_p for handling q 's crash:

$$T'_p = q \oplus \text{req}.q \& \left\{ \begin{array}{l} \text{res} \\ \text{crash}.r \oplus \text{req}.r \& \text{res} \end{array} \right\} \quad \begin{array}{l} T'_q = p \& \text{req}.p \oplus \text{res} \\ T'_r = q \& \text{crash}.p \& \text{req}.p \oplus \text{res} \end{array}$$

Here, T'_p states that client p first sends a message to the unreliable server q ; then, p expects a **response** from q . If q crashes, the client p detects the crash and handles it (via the new **crash** handling branch) by **requesting** from the failover server r . Meanwhile, r *also* detects whether q has crashed. If so, r activates its **crash** handling branch and handles p 's **request**.

In our model, crash detection and handling is done on the receiving side, e.g. T'_p detects whether q has **crashed** when waiting for a **response**, while T'_r monitors whether q has crashed. Handling crashes when receiving messages from a reliable role is unnecessary, e.g. the server q does not need crash handling when it receives from the (reliable) client p ; similarly, the (reliable) roles p and r interact without crash handling. This failure model is reflected in the semantics of both processes and session types in our work. Unlike classic MPST works, we allow processes to crash *arbitrarily* while attempting inputs or outputs (§2). When a process crashes, the channel endpoints held by the process also crash, and are assigned the new type **stop** (§3). E.g. when the server process Q crashes, the endpoint $s[q]$ held by Q becomes a crashed endpoint $s[q]^\dagger$; accordingly, the server type T'_q advances to **stop** to reflect the crash.

To ensure that communicating processes are type-safe even in the presence of crashes, we require their session types to satisfy a *safety property* accounting for possible crashes (Def. 8), which can be refined, e.g. as deadlock-freedom or liveness (Def. 18). We prove subject reduction, session fidelity, and various process properties (deadlock-freedom, liveness, *etc.*) even in the presence of crashes and optional reliability assumptions (Thms. 11, 15, 20).

Despite minimal changes to the surface syntax of session types and processes, the semantics surrounding crashes introduce subtle behaviours and increase complexity. Taking the DNS examples above, we compare the sizes of their (labelled) transition systems in Fig. 1 (based on Def. 7): the original system (left, two roles p and q , no crashes) has 10 states and 15 transitions; and the revised system (right, q may crash, with a new role r) has 101 states and 427 transitions. We discuss another, more complex example in §4. Checking whether a given combination of session types with possible crashes is safe, deadlock-free, or live, can be

challenging due to non-trivial behaviours and increased model size arising from crashes and crash handling. To tackle this, we show how to automatically verify such type-level properties by representing them as modal μ -calculus formulæ via the mCRL2 model checker [4].

Contributions and Structure. In §2 we introduce a multiparty session π -calculus (with minimal changes to the standard syntax) giving crash and crash handling semantics modelling *crash-stop* failures. In §3 we present *multiparty session types with crashes*: they describe how communication channels should be used to send/receive messages, and handle crashes. We formalise the semantics of collections of local types under *optional* reliability assumptions; we introduce a type system, and prove the Session Theorems: type safety, protocol conformance, and process properties (deadlock-freedom, termination, liveness, *etc.*) in Thms. 11, 15, and 20. In §4 we show how model checking can be incorporated to verify our behavioural properties, by expressing them as modal μ -calculus formulæ. We discuss related work and conclude in §5. Appendices §C and §D include additional examples and more details about the tool implementing our theory using the mCRL2 model checker. Further appendices with proofs are available in a separate technical report [3].

2 Multiparty Session Calculus with Crash-Stop Semantics

In this section, we formalise the syntax and operational semantics of our multiparty session π -calculus, where a process can fail arbitrarily, and crashes can be detected and handled by receiving processes. For clarity of presentation, we formalise a synchronous semantics.

Syntax of Processes. Our multiparty session π -calculus models processes that interact via multiparty channels, and may arbitrarily crash. For simplicity of presentation, our calculus is streamlined to focus on communication; standard extensions, e.g. with expressions and “if..then..else” statements, are routine and orthogonal to our formulation.

► **Definition 1** (Syntax of Multiparty Session π -Calculus). *Let $\mathbf{p}, \mathbf{q}, \dots$ denote roles belonging to a set \mathfrak{R} ; let s, s', \dots denote sessions; let x, y, \dots denote variables; let $\mathbf{m}, \mathbf{m}', \dots$ denote message labels; let X, Y, \dots denote process variables. The multiparty session π -calculus syntax is:*

$c ::= x \mid s[\mathbf{p}]$	<i>(variable or channel for session s with role \mathbf{p})</i>
$d ::= v \mid c$	<i>(basic value, variable, or channel with role)</i>
$w ::= v \mid s[\mathbf{p}]$	<i>(basic value or channel with role)</i>
$P, Q ::= \mathbf{0} \mid (\nu s)P \mid P \mid Q$	<i>(inaction, restriction, parallel composition)</i>
$c[\mathbf{q}] \oplus \mathbf{m}(d).P$	<i>(where $\mathbf{m} \neq \text{crash}$) (selection towards role \mathbf{q})</i>
$c[\mathbf{q}] \& \{ \mathbf{m}_i(x_i).P_i \}_{i \in I}$	<i>(branching from role \mathbf{q} with an index set $I \neq \emptyset$)</i>
$\text{def } D \text{ in } P \mid X(\tilde{d})$	<i>(process definition, process call)</i>
err $\mid s[\mathbf{p}] \zeta$	<i>(error, crashed channel endpoint)</i>
$D ::= X(\tilde{x}) = P$	<i>(declaration of process variable X)</i>

We write $\Pi_{i \in I} P_i$ for the parallel composition of processes P_i . Restriction, branching, and process definitions and declarations act as binders, as expected; $\text{fc}(P)$ is the set of free channels with roles in P (including $s[\mathbf{p}]$ in $s[\mathbf{p}] \zeta$), and $\text{fv}(P)$ is the set of free variables in P . Noticeable changes w.r.t. standard session calculi are *highlighted*.

Our calculus (Def. 1) includes basic values v (e.g. unit $()$, integers, strings), channels with roles (a.k.a. session endpoints) $s[\mathbf{p}]$, session scope restriction $(\nu s)P$, inaction $\mathbf{0}$, parallel composition $P \mid Q$, process definition $\text{def } D \text{ in } P$, process call $X(\tilde{d})$, and error **err**. *Selection*

$$\begin{array}{l}
\text{[R-}\oplus\&] \quad s[\mathbf{p}][\mathbf{q}]\&\{\mathbf{m}_i(x_i).P_i\}_{i \in I} \mid s[\mathbf{q}][\mathbf{p}]\oplus\mathbf{m}_k\langle w \rangle.Q \rightarrow P_k\{w/x_k\} \mid Q \quad \text{if } k \in I \\
\text{[R-ERR]} \quad s[\mathbf{p}][\mathbf{q}]\&\{\mathbf{m}_i(x_i).P_i\}_{i \in I} \mid s[\mathbf{q}][\mathbf{p}]\oplus\mathbf{m}\langle w \rangle.Q \rightarrow \mathbf{err} \quad \text{if } \forall i \in I : \mathbf{m}_i \neq \mathbf{m} \\
\text{[R-X]} \quad \text{def } X(x_1, \dots, x_n) = P \text{ in } (X\langle w_1, \dots, w_n \rangle \mid Q) \\
\quad \rightarrow \text{def } X(x_1, \dots, x_n) = P \text{ in } (P\{w_1/x_1\} \cdots \{w_n/x_n\} \mid Q) \\
\text{[R-CTX]} \quad P \rightarrow P' \text{ implies } \mathbb{C}[P] \rightarrow \mathbb{C}[P'] \\
\text{[R-}\equiv] \quad P' \equiv P \text{ and } P \rightarrow Q \text{ and } Q \equiv Q' \text{ implies } P' \rightarrow Q' \\
\text{[R-}\zeta\oplus] \quad P = s[\mathbf{p}][\mathbf{q}]\oplus\mathbf{m}\langle w \rangle.P' \rightarrow \prod_{j \in J} s_j[\mathbf{p}_j]\zeta \quad \text{where } \{s_j[\mathbf{p}_j]\}_{j \in J} = \text{fc}(P) \\
\text{[R-}\zeta\&] \quad P = s[\mathbf{p}][\mathbf{q}]\&\{\mathbf{m}_i(x_i).P_i\}_{i \in I} \rightarrow \prod_{j \in J} s_j[\mathbf{p}_j]\zeta \quad \text{where } \{s_j[\mathbf{p}_j]\}_{j \in J} = \text{fc}(P) \\
\text{[R-}\zeta\mathbf{mB}] \quad s[\mathbf{p}]\zeta \mid s[\mathbf{q}][\mathbf{p}]\oplus\mathbf{m}\langle v \rangle.Q' \rightarrow s[\mathbf{p}]\zeta \mid Q' \\
\text{[R-}\zeta\mathbf{m}] \quad s[\mathbf{p}]\zeta \mid s[\mathbf{q}][\mathbf{p}]\oplus\mathbf{m}\langle s'[\mathbf{r}] \rangle.Q' \rightarrow s[\mathbf{p}]\zeta \mid s'[\mathbf{r}]\zeta \mid Q' \\
\text{[R-}\odot] \quad s[\mathbf{p}][\mathbf{q}]\&\{\mathbf{m}_i(x_i).P_i, \mathbf{crash}.P'\}_{i \in I} \mid s[\mathbf{q}]\zeta \rightarrow P' \mid s[\mathbf{q}]\zeta
\end{array}$$

■ **Figure 2** Semantics of our session π -calculus. Rule [R- \equiv] uses the congruence \equiv defined in § A.

(a.k.a. internal choice) $c[\mathbf{q}]\oplus\mathbf{m}\langle d \rangle.P$ sends a message \mathbf{m} with payload d to role \mathbf{q} via endpoint c , where c may be a variable or channel with role, while d may also be a basic value. *Branching* (a.k.a. external choice) $c[\mathbf{q}]\&\{\mathbf{m}_i(x_i).P_i\}_{i \in I}$ expects to receive a message \mathbf{m}_i (for some $i \in I$) from role \mathbf{q} via endpoint c , and then continues as P_i . Importantly, a process implements crash detection by “receiving” the special message label `crash` in an external choice; such special message *cannot* be sent by any process (side condition $\mathbf{m} \neq \mathbf{crash}$ in selection). For example, $s[\mathbf{p}][\mathbf{q}]\&\{\mathbf{m}(x).P, \mathbf{crash}.P'\}$ is a process that uses the session endpoint $s[\mathbf{p}]$ to receive message \mathbf{m} from \mathbf{q} , but if \mathbf{q} has crashed, then the process continues as P' . Finally, our calculus includes *crashed session endpoints* $s[\mathbf{p}]\zeta$, denoting that the endpoint for role \mathbf{p} in session s has crashed.

Operational Semantics. We give the operational semantics of our session π -calculus in Def. 2, using a standard *structural congruence* extended with a new *crash elimination rule* which garbage-collects sessions where all endpoints are crashed: (full congruence rules in § A)

$$(\nu s)(s[\mathbf{p}_1]\zeta \mid \cdots \mid s[\mathbf{p}_n]\zeta) \equiv \mathbf{0} \quad \text{[C-CRASHELIM]}$$

► **Definition 2.** A reduction context \mathbb{C} is defined as: $\mathbb{C} ::= \mathbb{C} \mid P \mid (\nu s)\mathbb{C} \mid \text{def } D \text{ in } \mathbb{C} \mid []$. The reduction \rightarrow is defined in Fig. 2; we write $\rightarrow^+ / \rightarrow^*$ for its transitive / reflexive-transitive closure. We write $P \not\rightarrow$ iff $\nexists P'$ such that $P \rightarrow P'$ is derivable without rules [R- $\zeta\oplus$] and [R- $\zeta\&$] (i.e. P is stuck, unless a crash occurs). We say P has an error iff $\exists \mathbb{C}$ with $P = \mathbb{C}[\mathbf{err}]$.

Part of our operational semantics rules in Fig. 2 are standard. Rule [R- $\oplus\&$] describes a communication on session s between receiver \mathbf{p} and sender \mathbf{q} , if the sent message \mathbf{m}_k can be handled by the receiver ($k \in I$); otherwise, a message label mismatch causes an **error** via rule [R-ERR]. Rule [R-X] expands process definitions when called. Rules [R-CTX] and [R- \equiv] allow processes to reduce under reduction contexts and modulo structural congruence.

The remaining rules in Fig. 2 (highlighted) are novel: they model crashes, and crash handling. Rules [R- $\zeta\oplus$] and [R- $\zeta\&$] state that a process P may crash while attempting any selection or branching operation, respectively; when P crashes, it reduces to a parallel composition where all the channel endpoints held by P are crashed. The *lost message rules* [R- $\zeta\mathbf{mB}$] and [R- $\zeta\mathbf{m}$] state that if a process sends a message to a crashed endpoint, then the message is lost; if the message payload is a session endpoint $s'[\mathbf{r}]$, then it becomes crashed. Finally, the *crash handling rule* [R- \odot] states that if a process attempts to receive a message from a crashed endpoint, then the process detects the crash and follows its crash handling branch P' . We now show an example of rule [R- $\zeta\oplus$]; more examples can be found in § C.

► **Example 3.** Processes

$$P = s[\mathbf{p}][\mathbf{q}] \oplus \mathbf{m}' \langle s[\mathbf{r}] \rangle . s[\mathbf{p}][\mathbf{r}] \& \mathbf{m}(x) \quad \text{and} \quad Q = s[\mathbf{q}][\mathbf{p}] \& \mathbf{m}'(x) . x[\mathbf{p}] \oplus \mathbf{m} \langle 42 \rangle$$

communicate on a session s ; P uses $s[\mathbf{p}]$ to send $s[\mathbf{r}]$ to role \mathbf{q} ; Q uses $s[\mathbf{q}]$ to receive it, then sends a message to role \mathbf{p} via $s[\mathbf{r}]$. Suppose that P crashes before sending: this gives rise to the reduction (by rule $[\mathbf{R}\text{-}\mathcal{I} \oplus]$) $(\nu s)(P \mid Q) \rightarrow (\nu s)(s[\mathbf{p}] \dagger \mid s[\mathbf{r}] \dagger \mid Q)$. Observe that $s[\mathbf{p}]$ and $s[\mathbf{r}]$, which were held by P , are now crashed.

3 Multiparty Session Types with Crashes

In this section, we present a generalised type system for our multiparty session π -calculus (introduced in Def. 1). As in standard MPST, we assign session types to channel endpoints; we show the syntax of our types in §3.1, where our key additions are *crash handling branches*, and a new type **stop** for crashed endpoints. In §3.2, we give a labelled transition system (LTS) semantics to typing contexts, to represent the behaviour of a collection of types.

Unlike classic MPST, our type system is generalised in the style of [29], hence it has *no* global types; rather, it uses a *safety property* formalising the *minimum* requirement for a typing context to ensure *subject reduction* (and thus, type safety). In this paper, such a safety property is defined in §3.3: unlike previous work, the property accounts for potential crashes, and supports explicit (and optional) reliability assumptions. We show typing rules in §3.4, and the main properties of the typing system: subject reduction (Thm. 11) and session fidelity (Thm. 15) in §3.5. Finally, we demonstrate how we can infer runtime process properties from typing contexts in §3.6.

3.1 Types

A *session type* describes how a process is expected to use a communication channel to send/receive messages to/from other roles involved in a multiparty session. We formalise the syntax of session types in Def. 4, where we add the **stop** type to their standard syntax [29].

► **Definition 4 (Types).** *Our types include both basic types and session types:*

$$\begin{aligned} B &::= \text{int} \mid \text{bool} \mid \text{real} \mid \text{unit} \mid \dots && (\text{basic types}) \\ S &::= B \mid T && (\text{basic type or session type}) \\ T &::= \mathbf{p} \& \{ \mathbf{m}_i(S_i).T_i \}_{i \in I} \mid \mathbf{p} \oplus \{ \mathbf{m}_i(S_i).T_i \}_{i \in I} && (\text{external or internal choice, with } I \neq \emptyset) \\ &\quad \mid \mu \mathbf{t}.T \mid \mathbf{t} \mid \text{end} && (\text{recursion, type variable, or termination}) \\ U &::= T \mid \text{stop} && (\text{session type or crash type}) \end{aligned}$$

*In internal and external choices, the index set I must be non-empty, and labels \mathbf{m}_i must be pairwise distinct. Types are always closed (i.e. each recursion variable \mathbf{t} is bound under a $\mu \mathbf{t} \dots$) and recursion variables are guarded, i.e. they can only appear under an internal/external choice (e.g. $\mu \mathbf{t}. \mu \mathbf{t}'. \mathbf{t}$ is not a valid type). For brevity, we may omit the payload type **unit** and the trailing **end**: e.g. $\mathbf{p} \oplus \mathbf{m}_1.\mathbf{r} \& \mathbf{m}_2$ is shorthand for $\mathbf{p} \oplus \mathbf{m}_1(\text{unit}).\mathbf{r} \& \mathbf{m}_2(\text{unit}).\text{end}$.*

The internal choice (selection) type $\mathbf{p} \oplus \{ \mathbf{m}_i(S_i).T_i \}_{i \in I}$ denotes *sending* a message \mathbf{m}_i (by picking some $i \in I$) with a payload of type S_i to role \mathbf{p} , and then continue the protocol as T_i . Dually, the external choice (branching) type $\mathbf{p} \& \{ \mathbf{m}_i(S_i).T_i \}_{i \in I}$ denotes *receiving* a message \mathbf{m}_i (for any $i \in I$) with a payload of type S_i from role \mathbf{p} , and then continue as T_i . The type **end** indicates that a session endpoint should not be used for further communications.

$$\begin{array}{c}
\frac{k \in I}{s[p]:q\oplus\{m_i(S_i).T_i\}_{i \in I} \xrightarrow{s[p]:q\oplus m_k(S_k)} s[p]:T_k} \quad [\Gamma\text{-}\oplus] \qquad \frac{k \in I}{s[p]:q\&\{m_i(S_i).T_i\}_{i \in I} \xrightarrow{s[p]:q\& m_k(S_k)} s[p]:T_k} \quad [\Gamma\text{-}\&] \\
\frac{\Gamma_1 \xrightarrow{s[p]:q\oplus m(S)} \Gamma'_1 \quad \Gamma_2 \xrightarrow{s[q]:p\& m(S')} \Gamma'_2 \quad S \leq S'}{\Gamma_1, \Gamma_2 \xrightarrow{s[p]:[q]m} \Gamma'_1, \Gamma'_2} \quad [\Gamma\text{-}\oplus\&] \qquad \frac{s[p]:T\{\mu t.T/t\} \xrightarrow{\alpha} \Gamma' \quad \Gamma \xrightarrow{\alpha} \Gamma'}{s[p]:\mu t.T \xrightarrow{\alpha} \Gamma'} \quad [\Gamma\text{-}\mu] \quad \frac{\Gamma \xrightarrow{\alpha} \Gamma'}{\Gamma, c:U \xrightarrow{\alpha} \Gamma', c:U} \quad [\Gamma\text{-},]} \\
\frac{T \not\leq \text{end}}{s[p]:T \xrightarrow{s[p]\not\leq} s[p]:\text{stop}} \quad [\Gamma\text{-}\not\leq] \qquad \frac{}{s[p]:\text{stop} \xrightarrow{s[p]\text{stop}} s[p]:\text{stop}} \quad [\Gamma\text{-}\text{stop}] \qquad \frac{\Gamma \xrightarrow{\alpha} \Gamma'}{\Gamma, x:B \xrightarrow{\alpha} \Gamma', x:B} \quad [\Gamma\text{-},B] \\
\frac{\Gamma_1 \xrightarrow{s[q]:p\& \text{crash}} \Gamma'_1 \quad \Gamma_2 \xrightarrow{s[p]\text{stop}} \Gamma'_2}{\Gamma_1, \Gamma_2 \xrightarrow{s[q]\odot p} \Gamma'_1, \Gamma'_2} \quad [\Gamma\text{-}\odot] \qquad \frac{\Gamma_1 \xrightarrow{s[p]:q\oplus m(S)} \Gamma'_1 \quad \Gamma_2 \xrightarrow{s[q]\text{stop}} \Gamma'_2}{\Gamma_1, \Gamma_2 \xrightarrow{s[p]:[q]m} \Gamma'_1, \Gamma'_2} \quad [\Gamma\text{-}\not\leq m]
\end{array}$$

■ **Figure 3** Typing context semantics.

Crashes and Crash Detection. The key novelty of Def. 4 is the new type `stop` describing a crashed session endpoint. Similarly to Def. 1, we also introduce a distinguished message label `crash` for crash handling in external choices. For example, recall the types in § 1:

- the type $q\&\{\text{res}.T, \text{crash}.T'\}$ means that we expect a `res` response message from role q , but if we detect that q has crashed, then the protocol continues along the handling branch T' ;
- the type $q\&\text{crash}.T$ denotes a “pure” crash recovery behaviour: we are not communicating with q , but the recovery protocol T is activated whenever we detect that q has crashed.

Since `crash` messages *cannot* be crafted by any role in a session (see Def. 1), we postulate that the `crash` message label cannot appear in internal choice types.

Session Subtyping. We use a subtyping relation \leq that is mostly standard: a subtype can have wider internal choices and narrower external choices w.r.t. a supertype. To correctly support crash handling, we apply two changes: (1) we add the relation `stop` \leq `stop`, and (2) we treat external choices with a singleton `crash` branch in a special way: they represent a “pure” crash recovery protocol (as outlined above), hence we do not allow the supertype to have more input branches. This way, a “pure” crash recovery type can only be implemented by a “pure” crash recovery process (with a singleton `crash` detection branch); such processes are treated specially by the properties in § 3.6. For the complete definition of \leq , see § B.

3.2 Typing Contexts and their Semantics

Before introducing the typing rules for our calculus (in § 3.4), we first formalise typing contexts (Def. 5) and their semantics (Def. 7).

► **Definition 5** (Typing Contexts). Θ denotes a partial mapping from process variables to n -tuples of types, and Γ denotes a partial mapping from channels to types. Their syntax is:

$$\Theta ::= \emptyset \mid \Theta, X:S_1, \dots, S_n \qquad \Gamma ::= \emptyset \mid \Gamma, x:S \mid \Gamma, s[p]:U$$

The context composition Γ_1, Γ_2 is defined iff $\text{dom}(\Gamma_1) \cap \text{dom}(\Gamma_2) = \emptyset$. We write $s \notin \Gamma$ iff $\forall p : s[p] \notin \text{dom}(\Gamma)$ (i.e. session s does not occur in Γ). We write $\Gamma \leq \Gamma'$ iff $\text{dom}(\Gamma) = \text{dom}(\Gamma')$ and $\forall c \in \text{dom}(\Gamma) : \Gamma(c) \leq \Gamma'(c)$.

Unlike typical session typing systems, our Def. 5 allows a session endpoint $s[p]$ to have either a session type T , or the crash type `stop`. We equip our typing contexts with a labelled transition system (LTS) semantics (in Def. 7) using the labels in Def. 6.

► **Definition 6** (Transition Labels). Let α denote a transition label having the form:

$$\alpha ::= \begin{array}{l} s[\mathbf{p}]:\mathbf{q}\&\mathbf{m}(S) \quad (\text{in session } s, \mathbf{p} \text{ receives message } \mathbf{m}(S) \text{ from } \mathbf{q}; \text{ we omit } S \text{ if } S = \text{unit}) \\ s[\mathbf{p}]:\mathbf{q}\oplus\mathbf{m}(S) \quad (\text{in session } s, \mathbf{p} \text{ sends message } \mathbf{m}(S) \text{ to } \mathbf{q}; \text{ we omit } S \text{ if } S = \text{unit}) \\ s[\mathbf{p}][\mathbf{q}]\mathbf{m} \quad (\text{in session } s, \text{ message } \mathbf{m} \text{ is transmitted from } \mathbf{p} \text{ to } \mathbf{q}) \\ s[\mathbf{p}]\downarrow \quad (\text{in session } s, \mathbf{p} \text{ crashes}) \\ s[\mathbf{p}]\odot\mathbf{q} \quad (\text{in session } s, \mathbf{p} \text{ has detected that } \mathbf{q} \text{ has crashed}) \\ s[\mathbf{p}]\text{stop} \quad (\text{in session } s, \mathbf{p} \text{ has stopped due to a crash}) \end{array}$$

► **Definition 7** (Typing Context Semantics). The typing context transition $\xrightarrow{\alpha}$ is defined in Fig. 3. We write $\Gamma \xrightarrow{\alpha}$ iff $\Gamma \xrightarrow{\alpha} \Gamma'$ for some Γ' . We define the two reductions \rightarrow and $\rightarrow_{\downarrow} \setminus s; \mathcal{R}$ (where s is a session, and \mathcal{R} is a set of roles) as follows:

- $\Gamma \rightarrow \Gamma'$ holds iff $\Gamma \xrightarrow{s[\mathbf{p}][\mathbf{q}]\mathbf{m}} \Gamma'$ or $\Gamma \xrightarrow{s[\mathbf{q}]\odot\mathbf{p}} \Gamma'$ (for some $s, \mathbf{p}, \mathbf{q}, \mathbf{m}$). This means that Γ can advance via message transmission or crash detection, but it cannot advance by crashing one of its entries. We write $\Gamma \rightarrow$ iff $\Gamma \rightarrow \Gamma'$ for some Γ' , and $\Gamma \not\rightarrow$ for its negation (i.e. there is no Γ' such that $\Gamma \rightarrow \Gamma'$), and \rightarrow^* for the reflexive and transitive closure of \rightarrow ;
- $\Gamma \rightarrow_{\downarrow} \setminus s; \mathcal{R} \Gamma'$ holds iff $\Gamma \xrightarrow{\alpha} \Gamma'$ with $\alpha \in \{s[\mathbf{q}][\mathbf{r}]\mathbf{m}, s[\mathbf{q}]\odot\mathbf{r}, s[\mathbf{p}]\downarrow \mid \mathbf{p}, \mathbf{q}, \mathbf{r} \in \mathcal{R}, \mathbf{p} \notin \mathcal{R}\}$. This means that Γ can advance via message transmission or crash detection on session s , involving any roles \mathbf{q} and \mathbf{r} . (Recall that \mathcal{R} is the set of all roles.) Moreover, Γ can advance by crashing one of its entries $s[\mathbf{p}]$ – unless $\mathbf{p} \in \mathcal{R}$, which means that \mathbf{p} is assumed to be reliable. We write $\Gamma \rightarrow_{\downarrow} \setminus s; \mathcal{R} \Gamma'$ iff $\Gamma \rightarrow_{\downarrow} \setminus s; \mathcal{R} \Gamma'$ for some Γ' , and $\Gamma \not\rightarrow_{\downarrow} \setminus s; \mathcal{R}$ for its negation, and $\rightarrow_{\downarrow}^* \setminus s; \mathcal{R}$ as the reflexive and transitive closure of $\rightarrow_{\downarrow} \setminus s; \mathcal{R}$. We write $\Gamma \rightarrow_{\downarrow} \setminus s; \emptyset \Gamma'$ iff $\Gamma \rightarrow_{\downarrow} \setminus s; \emptyset \Gamma'$ for some s (i.e. Γ may advance by crashing any role on any session).

Def. 7 subsumes the standard typing context reductions [29, Def. 2.8]. Rule $[\Gamma\&]$ (resp. $[\Gamma\&]$) says that an entry can perform an output (resp. input) transition. Rule $[\Gamma\&\&]$ synchronises matching input/output transitions, provided that the payloads are compatible by subtyping; as a result, the context advances via a message transmission label $s[\mathbf{p}][\mathbf{q}]\mathbf{m}$. Other standard rules are $[\Gamma\text{-}\mu]$ for recursion, and $[\Gamma\text{-}\cdot]$ and $[\Gamma\text{-}\mathcal{B}]$ for reductions in a larger context.

The key innovations are the (highlighted) rules modelling crashes and crash detection. By rule $[\Gamma\text{-}\downarrow]$, an entry can crash and become **stop** at any time (unless it is already **ended** or **stopped**); then, by rule $[\Gamma\text{-}\text{stop}]$, it keeps signalling that it is crashed, with label $s[\mathbf{p}]\text{stop}$.

Rule $[\Gamma\text{-}\odot]$ models crash detection and handling: if $s[\mathbf{p}]$ signals that it has crashed and stopped, another entry $s[\mathbf{q}]$ can then take its **crash** handling branch (part of an external choice from \mathbf{p}). This corresponds to the process reduction rule $[\mathbf{R}\text{-}\odot]$ for crash detection.

Finally, rule $[\Gamma\text{-}\downarrow\mathbf{m}]$ models the case where the entry $s[\mathbf{p}]$ is sending a message $\mathbf{m}(S)$ to a crashed $s[\mathbf{q}]$: this yields a transmission label $s[\mathbf{p}][\mathbf{q}]\mathbf{m}$, and \mathbf{p} continues – although the sent message is not actually received by crashed \mathbf{q} . This corresponds to the process reduction rule $[\mathbf{R}\text{-}\downarrow\mathbf{m}]$ where a process sends a message to a crashed endpoint, and cannot detect its crash.

3.3 Typing Context Safety

To ensure type safety (Cor. 12), i.e. well-typed processes do not result in **errors**, we define a safety property $\varphi(\cdot)$ (Def. 8) as a predicate on typing contexts Γ . The safety property φ is the key feature of generalised MPST systems [29, Def. 4.1]; in this work, we extend its definition in two crucial ways: (1) we support crashes and crash detection, and (2) we make the property parametric upon a (possibly empty) set of *reliable* roles \mathcal{R} , thus introducing *optional reliability assumptions* about roles in a session that never fail.

$$\begin{array}{c}
\frac{\Theta(X) = S_1, \dots, S_n}{\Theta \vdash X:S_1, \dots, S_n} \text{ [T-X]} \quad \frac{v \in B}{\emptyset \vdash v:B} \text{ [T-B]} \quad \frac{\forall i \in 1..n \ S_i \text{ is basic or } c_i:S_i \vdash c_i:\text{end}}{\text{end}(c_1:S_1, \dots, c_n:S_n)} \text{ [T-end]} \\
\frac{\Theta \vdash X:S_1, \dots, S_n \quad \text{end}(\Gamma_0) \quad \forall i \in 1..n \ \Gamma_i \vdash d_i:S_i \quad S_i \not\leq \text{end}}{\Theta \cdot \Gamma_0, \Gamma_1, \dots, \Gamma_n \vdash X(d_1, \dots, d_n)} \text{ [T-CALL]} \\
\frac{\text{end}(\Gamma)}{\Theta \cdot \Gamma \vdash \mathbf{0}} \text{ [T-0]} \quad \frac{\Theta, X:S_1, \dots, S_n \cdot x_1:S_1, \dots, x_n:S_n \vdash P \quad \Theta, X:S_1, \dots, S_n \cdot \Gamma \vdash Q}{\Theta \cdot \Gamma \vdash \text{def } X(x_1:S_1, \dots, x_n:S_n) = P \text{ in } Q} \text{ [T-def]} \\
\frac{\Gamma_1 \vdash c:\mathbf{q}\&\{\mathbf{m}_i(S_i).T_i\}_{i \in I} \quad \forall i \in I \ \Theta \cdot \Gamma, y_i:S_i, c:T_i \vdash P_i}{\Theta \cdot \Gamma, \Gamma_1 \vdash c[\mathbf{q}]\&\{\mathbf{m}_i(y_i).P_i\}_{i \in I}} \text{ [T-\&]} \quad \frac{\Theta \cdot \Gamma_1 \vdash P_1 \quad \Theta \cdot \Gamma_2 \vdash P_2}{\Theta \cdot \Gamma_1, \Gamma_2 \vdash P_1 \mid P_2} \text{ [T-|]} \\
\frac{\Gamma_1 \vdash c:\mathbf{q}\oplus\{\mathbf{m}(S).T\} \quad \Gamma_2 \vdash d:S \quad S \not\leq \text{end} \quad \Theta \cdot \Gamma, c:T \vdash P}{\Theta \cdot \Gamma, \Gamma_1, \Gamma_2 \vdash c[\mathbf{q}]\oplus\mathbf{m}(d).P} \text{ [T-\oplus]} \quad \frac{S \leq S'}{c:S \vdash c:S'} \text{ [T-SUB]} \\
\frac{\text{end}(\Gamma)}{\Theta \cdot \Gamma, s[\mathbf{p}]:\text{stop} \vdash s[\mathbf{p}]\downarrow} \text{ [T-\downarrow]} \quad \frac{\Gamma' = \{s[\mathbf{p}]:T_p\}_{p \in I} \quad \varphi(\Gamma') \quad s \notin \Gamma \quad \Theta \cdot \Gamma, \Gamma' \vdash P}{\Theta \cdot \Gamma \vdash (\nu s:\Gamma') P} \text{ [T-\nu]}
\end{array}$$

■ **Figure 4** Typing rules for processes; φ in [T- ν] is an $(s; \mathcal{R})$ -safety property, for some \mathcal{R} .

► **Definition 8** (Typing Context Safety). *Given a set of reliable roles \mathcal{R} and a session s , we say that φ is an $(s; \mathcal{R})$ -safety property of typing contexts iff, whenever $\varphi(\Gamma)$, we have:*

$$\begin{array}{l}
\text{[S-\oplus\&]} \ \Gamma \xrightarrow{s[\mathbf{p}]:\mathbf{q}\oplus\mathbf{m}(S)} \text{ and } \Gamma \xrightarrow{s[\mathbf{q}]:\mathbf{p}\&\mathbf{m}'(S')} \text{ implies } \Gamma \xrightarrow{s[\mathbf{p}]:\mathbf{q}\mathbf{m}}; \\
\text{[S-\downarrow\&]} \ \Gamma \xrightarrow{s[\mathbf{p}]:\text{stop}} \text{ and } \Gamma \xrightarrow{s[\mathbf{q}]:\mathbf{p}\&\mathbf{m}(S)} \text{ implies } \Gamma \xrightarrow{s[\mathbf{q}]\oplus\mathbf{p}}; \\
\text{[S-\rightarrow_i]} \ \Gamma \xrightarrow{i} \setminus s; \mathcal{R} \Gamma' \text{ implies } \varphi(\Gamma').
\end{array}$$

We say Γ is $(s; \mathcal{R})$ -safe, written $\text{safe}(s; \mathcal{R}, \Gamma)$, if $\varphi(\Gamma)$ holds for some $(s; \mathcal{R})$ -safety property φ . We say Γ is safe, written $\text{safe}(\Gamma)$, if $\varphi(\Gamma)$ holds for some property φ which is an $(s; \emptyset)$ -safety property for all sessions s occurring in $\text{dom}(\Gamma)$.

By Def. 8, safety is a *coinductive* property [28]: fix s and \mathcal{R} , $(s; \mathcal{R})$ -safe is the largest $(s; \mathcal{R})$ -safety property, i.e. the union of all $(s; \mathcal{R})$ -safety properties; to prove that some Γ is $(s; \mathcal{R})$ -safe, we must find a property φ such that $\Gamma \in \varphi$, and prove that φ is an $(s; \mathcal{R})$ -safety property. Intuitively, we can construct such φ (if it exists) as the set containing Γ and all its reductums (via transition $\xrightarrow{i}^* \setminus s; \mathcal{R}$), and checking whether all elements of φ satisfy all clauses of Def. 8. By clause [S-\oplus\&], whenever two roles \mathbf{p} and \mathbf{q} attempt to communicate, the communication must be possible, i.e. the receiver \mathbf{q} must support all output messages of sender \mathbf{p} , with compatible payload types (by rule [T-\oplus\&] in Fig. 3). For “pure” crash recovery types (with a singleton **crash** handling branch) there would not be corresponding sender, so this clause holds trivially. Clause [S-\downarrow\&] states that if a role \mathbf{q} receives from a crashed role \mathbf{p} , then \mathbf{q} must have a **crash** handling branch. Clause [S-\rightarrow_i] states that any typing context Γ' that Γ transitions to (on session s) must also be in φ (hence, Γ' must also be $(s; \mathcal{R})$ -safe); notice that, by using transition $\xrightarrow{i} \setminus s; \mathcal{R}$, we ignore crashes $s[\mathbf{p}]\downarrow$ of any reliable role $\mathbf{p} \in \mathcal{R}$.

► **Example 9.** Consider the simple DNS scenario from §1, its types T'_p , T'_q and T'_r , and the typing context $\Gamma = s[\mathbf{p}]:T'_p, s[\mathbf{q}]:T'_q, s[\mathbf{r}]:T'_r$. We know, and can verify, that Γ is $(s; \{\mathbf{p}, \mathbf{r}\})$ -safe by checking its reductions. For example, for the case where \mathbf{q} crashes immediately, we have: $\Gamma \xrightarrow{i} \setminus s; \{\mathbf{p}, \mathbf{r}\} \ s[\mathbf{p}]:T'_p, \ s[\mathbf{q}]:\text{stop}, \ s[\mathbf{r}]:T'_r \xrightarrow{i}^* \setminus s; \{\mathbf{p}, \mathbf{r}\} \ s[\mathbf{p}]:\text{end}, \ s[\mathbf{q}]:\text{stop}, \ s[\mathbf{r}]:\text{end}$ and each reductum satisfies all clauses of Def. 8. Full reductions are available in §C, Ex. 23.

3.4 Typing Rules

Our type system uses two kinds of typing contexts (introduced in Def. 5): Θ to assign an n -tuple of types to each process variable X (one type per argument), and Γ to map variables to payload types (basic types or session types), and channels with roles to session types or the **stop** type. Together, they are used in judgements of the form:

$$\Theta \cdot \Gamma \vdash P \quad (\text{with } \Theta \text{ omitted when empty})$$

which reads, “given the process types in Θ , P uses its variables and channels *linearly* according to Γ .” This *typing judgement* is defined by the rules in Fig. 4, where, for convenience, we type-annotate channels bound by process definitions and restrictions.

The main innovations in Fig. 4 are rules $[\text{T-}\nu]$ and $[\text{T-}\zeta]$ (**highlighted**). Rule $[\text{T-}\nu]$ utilises a safety property φ (Def. 8) to validate session restrictions, taking into account crashes and crash handling, and any reliable role assumption in the (possibly empty) set \mathcal{R} . The rule can be instantiated by choosing a set \mathcal{R} and safety property φ (e.g. among the stronger properties presented in Def. 18 later on). Rule $[\text{T-}\zeta]$ types crashed session endpoints as **stop**.

The rest of the rules in Fig. 4 are mostly standard. $[\text{T-X}]$ looks up process variables. $[\text{T-B}]$ types a value v if it belongs to a basic type B . $[\text{T-SUB}]$ holds for a singleton typing context $c:S$, and applies subtyping when assigning a type S' to a variable or channel c . $[\text{T-end}]$ defines a predicate $\text{end}(\cdot)$ on typing contexts, indicating all endpoints are terminated – it is used in $[\text{T-0}]$ for typing an inactive process $\mathbf{0}$, and in $[\text{T-}\zeta]$ for crashed endpoints. $[\text{T-}\oplus]$ and $[\text{T-}\&]$ assign selection and branching types to channels used by selection and branching processes. Minor changes w.r.t. standard session types are the clauses “ $S \not\leq \text{end}$ ” in rules $[\text{T-}\oplus]$ and $[\text{T-CALL}]$: they forbid sending or passing **end**-typed channels, while allowing sending/passing channels and data of any other type.¹ Rules $[\text{T-def}]$ and $[\text{T-CALL}]$ handle recursive processes declarations and calls. $[\text{T-}]$ *linearly* splits the typing context into two, one for typing each sub-process.

3.5 Subject Reduction and Session Fidelity

We present our key results on typed processes: *subject reduction* and *session fidelity* (Thms. 11 and 15). A main feature of our theory is that our results explicitly account for the *spectrum* of optional reliability assumptions used during typing.

- On one end of the spectrum, our results hold without any reliability assumption: any process and session endpoint may crash at any time. This is obtained if, for each Γ used during typing, we assume $\text{safe}(\Gamma)$ (Def. 8), with no reliable roles.
- At the other end of the spectrum, we recover the classic MPST results by assuming that all roles in all sessions are reliable – i.e. if for each Γ used during typing, and for all $s \in \Gamma$, we assume $\text{safe}(s; \mathcal{R}_s, \Gamma)$ with $\mathcal{R}_s = \{\mathbf{p} \mid s[\mathbf{p}] \in \text{dom}(\Gamma)\}$.

Subject reduction (Thm. 11 below) states that if a well-typed process P reduces to P' , then the reduction is simulated by its typing context Γ , provided that the reliability assumptions embedded in Γ hold when P reduces. In other words, if a channel endpoint $s[\mathbf{p}]$ occurring in P is assumed reliable in Γ , then P should *not* crash $s[\mathbf{p}]$ while reducing; any other reduction of P (including those that crash other session endpoints) are type-safe. To formalise this idea, we define *reliable process reduction* $\rightarrow_{\zeta \setminus s, \mathcal{R}}$ as a subset of P 's reductions. We also define *assumption-abiding reduction* \checkmark to enforce reliable process reductions across nested sessions.

¹ This restriction is needed for Thms. 11 and 20. It does not limit the expressiveness of our typed calculus, since sending an **end**-typed channel (not usable for communication) amounts to sending a basic value.

► **Definition 10** (Reliable Process Reductions and Assumption-Abiding Reductions). *The reliable process reduction $\rightarrow_{\not\downarrow s; \mathcal{R}}$ is defined as follows:*

$$\frac{P \rightarrow P' \quad \forall \mathbf{p} \in \mathcal{R} : \nexists R : P' \equiv R \mid s[\mathbf{p}] \not\downarrow}{P \rightarrow_{\not\downarrow s; \mathcal{R}} P'}$$

Assume $\Theta \cdot \Gamma \vdash P$ where, for each $s \in \Gamma$, there is a set of reliable roles \mathcal{R}_s such that $\text{safe}(s; \mathcal{R}_s, \Gamma)$. We define the assumption-abiding reduction \checkmark such that $P \checkmark P'$ holds when: (1) $P \rightarrow_{\not\downarrow s; \mathcal{R}_s} P'$ for all $s \in \Gamma$; and (2) if $P \equiv (\nu s' : \Gamma_{s'}) Q$ (for some $s', \Gamma_{s'}, Q$) and $P' \equiv (\nu s') Q'$ and $Q \rightarrow Q'$, then $\exists \mathcal{R}'$ such that $\text{safe}(s'; \mathcal{R}', \Gamma_{s'})$ and $Q \rightarrow_{\not\downarrow s'; \mathcal{R}'} Q'$. We write $\checkmark^+ / \checkmark^*$ for the transitive / reflexive-transitive closure of \checkmark .

Hence, when $P \rightarrow_{\not\downarrow s; \mathcal{R}} P'$ holds, none of the session endpoints $s[\mathbf{p}]$ (where \mathbf{p} is a reliable role in set \mathcal{R}) are crashed in P' . When P is well-typed, the reduction $P \checkmark P'$ covers all (and only) the reductions of P that do not violate any reliability assumption used for deriving $\Theta \cdot \Gamma \vdash P$; notice that we use congruence \equiv to quantify over all restricted sessions in P and ensure their reductions respect all reliability assumptions in their typing, by $[\text{T-}\nu]$ in Fig. 4.

We can now use \checkmark to state our subject reduction result. Its proof is available in [3].

► **Theorem 11** (Subject Reduction). *Assume $\Theta \cdot \Gamma \vdash P$ where $\forall s \in \Gamma : \exists \mathcal{R}_s : \text{safe}(s; \mathcal{R}_s, \Gamma)$. If $P \checkmark P'$, then $\exists \Gamma'$ such that $\Gamma \rightarrow_{\not\downarrow}^* \Gamma'$, and $\forall s \in \Gamma' : \text{safe}(s; \mathcal{R}_s, \Gamma')$, and $\Theta \cdot \Gamma' \vdash P'$.*

► **Corollary 12** (Type Safety). *Assume $\emptyset \cdot \emptyset \vdash P$. If $P \checkmark^* P'$, then P' has no error.*

► **Example 13** (Subject reduction). Take the DNS example (§1) and consider the process acting as the (unreliable) role \mathbf{q} : $P_{\mathbf{q}} = s[\mathbf{q}][\mathbf{p}] \& \text{req}.s[\mathbf{q}][\mathbf{p}] \oplus \text{res}.\mathbf{0}$. Using type $T'_{\mathbf{q}}$ from the same example, can type $P_{\mathbf{q}}$ with the typing context $\Gamma_{\mathbf{q}} = s[\mathbf{q}] : T'_{\mathbf{q}}$. Following a crash reduction via $[\text{R-}\not\downarrow \&]$, the process evolves as $P_{\mathbf{q}} \rightarrow P'_{\mathbf{q}} = s[\mathbf{q}] \not\downarrow$. Observe that the typing context $\Gamma_{\mathbf{q}}$ can reduce to $\Gamma'_{\mathbf{q}} = s[\mathbf{p}] : \text{stop}$, via $[\text{T-}\not\downarrow]$; and by typing rule $[\text{T-}\not\downarrow]$, we can type $P'_{\mathbf{q}}$ with $\Gamma'_{\mathbf{q}}$.

Session fidelity states the opposite implication w.r.t. subject reduction: if a process P is typed by Γ , and Γ can reduce along session s (possibly by crashing some endpoint of s), then P can reproduce at least one of the reductions of Γ (but maybe not all such reductions, because Γ over-approximates the behaviour of P). As a consequence, we can infer P 's behaviour from Γ 's behaviour, as shown in Thm. 20. This result does *not* hold for all well-typed processes: a well-typed process can loop in a recursion like $\text{def } X(\dots) = X \text{ in } X$, or deadlock by suitably interleaving its communications across multiple sessions [13]. Thus, similarly to [29] and most session type works, we prove session fidelity for processes with guarded recursion, and implementing a single multiparty session as a parallel composition of one sub-process per role. Session fidelity is given in Thm. 15 below, by leveraging Def. 14.

► **Definition 14** (from [29]). *Assume $\emptyset \cdot \Gamma \vdash P$. We say that P :*

- (1) *has guarded definitions* iff in each process definition in P of the form $\text{def } X(x_1 : S_1, \dots, x_n : S_n) = Q$ in P' , for all $i \in 1..n$, if S_i is a session type, then a call $Y(\dots, x_i, \dots)$ can only occur in Q as a subterm of $x_i[\mathbf{q}] \& \{m_j(y_j).P_j\}_{j \in J}$ or $x_i[\mathbf{q}] \oplus m(d).P''$ (i.e. after using x_i for input or output);
- (2) *only plays role \mathbf{p} in s , by Γ* iff: (i) P has guarded definitions; (ii) $\text{fv}(P) = \emptyset$; (iii) $\Gamma = \Gamma_0, s[\mathbf{p}] : S$ with $S \not\prec \text{end}$ and $\text{end}(\Gamma_0)$; (iv) for all subterms $(\nu s' : \Gamma') P'$ in P , $\text{end}(\Gamma')$.

We say “ P only plays role \mathbf{p} in s ” iff $\exists \Gamma : \emptyset \cdot \Gamma \vdash P$, and item 2 holds.

Item 1 of Def. 14 formalises guarded recursion for processes. Item 2 identifies a process that plays exactly *one* role on *one* session; clearly, an ensemble of such processes cannot deadlock by waiting for each other on multiple sessions. All our examples satisfy Def. 14(2).

We can now formalise our session fidelity result (Thm. 15). The statement is superficially similar to Thm. 5.4 in [29], but it now includes explicit reliability assumptions for Γ ; it also covers more cases, since our typing contexts and processes can reduce by crashing, handling crashes, or losing messages sent to crashed session endpoints. The proof is available in [3].

► **Theorem 15** (Session Fidelity). *Assume $\emptyset \cdot \Gamma \vdash P$, with $\text{safe}(s; \mathcal{R}, \Gamma)$, $P \equiv \Pi_{p \in I} P_p$, and $\Gamma = \bigcup_{p \in I} \Gamma_p$ such that for each P_p : (1) $\emptyset \cdot \Gamma_p \vdash P_p$, and (2) either $P_p \equiv \mathbf{0}$, or P_p only plays p in s , by Γ_p . Then, $\Gamma \rightarrow_{\downarrow}^s \mathcal{R}$ implies $\exists \Gamma', P'$ such that $\Gamma \rightarrow_{\downarrow}^s \mathcal{R} \Gamma'$, $P \xrightarrow{\downarrow}^* P'$ and $\emptyset \cdot \Gamma' \vdash P'$, with $\text{safe}(s; \mathcal{R}, \Gamma')$, $P' \equiv \Pi_{p \in I} P'_p$, and $\Gamma' = \bigcup_{p \in I} \Gamma'_p$ such that for each P'_p : (1) $\emptyset \cdot \Gamma'_p \vdash P'_p$, and (2) either $P'_p \equiv \mathbf{0}$, or P'_p only plays p in s , by Γ'_p .*

3.6 Statically Verifying Run-Time Properties of Processes with Crashes

We conclude this section by showing how to infer run-time process properties from typing contexts, even in the presence of arbitrary process crashes. The formulations are based on [29, Def. 5.1 & Fig. 5(1)], but (1) we cater for optional assumptions on reliable roles; (2) a successfully-terminated process or typing context may include crashed session endpoints and failover types/processes (like DNS server \mathbf{r} in §1) that only run after detecting a crash; and (3) non-failover reliable roles terminate by reaching $\mathbf{0}$ (in processes) or end (in types).

Def. 16 formalises several desirable process properties, using the assumption-abiding reduction $\xrightarrow{\downarrow}$ (Def. 10) to embed any assumptions on reliable roles used for typing. The properties are mostly self-explanatory: *deadlock-freedom* means that if a process cannot reduce, then it only contains inactive or crashed sub-processes, or recovery processes attempting to detect others' crashes; *liveness* means that if a process is trying to perform an input or output, then it eventually succeeds (unless it is only attempting to detect others' crashes).

► **Definition 16** (Runtime Process Properties). *Assume $\emptyset \cdot \Gamma \vdash P$ where, $\forall s \in \Gamma$, there is a set of roles \mathcal{R}_s such that $\text{safe}(s; \mathcal{R}_s, \Gamma)$. We say P is:*

(1) **deadlock-free** iff $P \xrightarrow{\downarrow}^* P' \not\vdash$ implies

$$P' \equiv \mathbf{0} \mid \Pi_{i \in I} s_i[\mathbf{p}_i] \downarrow \mid \Pi_{j \in J} (\text{def } D_{j,1} \text{ in } \dots \text{def } D_{j,n_j} \text{ in } s_j[\mathbf{p}_j][\mathbf{q}_j] \& \text{crash}.Q'_j);$$

(2) **terminating** iff it is deadlock-free, and $\exists j$ finite such that $\forall n \geq j: P = P_0 \xrightarrow{\downarrow} P_1 \xrightarrow{\downarrow} \dots \xrightarrow{\downarrow} P_n$ implies $P_n \not\vdash$;

(3) **never-terminating** iff $P \xrightarrow{\downarrow}^* P'$ implies $P' \rightarrow$;

(4) **live** iff $P \xrightarrow{\downarrow}^* P' \equiv \mathbb{C}[Q]$ implies:

(i) if $Q = c[\mathbf{q}] \oplus \mathbb{m}(w).Q'$ then $\exists \mathbb{C}': P' \rightarrow^* \mathbb{C}'[Q']$;

(ii) if $Q = c[\mathbf{q}] \& \{\mathbb{m}_i(x_i).Q'_i\}_{i \in I}$ where $\{\mathbb{m}_i \mid i \in I\} \neq \{\text{crash}\}$, then $\exists \mathbb{C}', k \in I, w: P' \rightarrow^* \mathbb{C}'[Q'_k \{w/x_k\}]$.

In Def. 18 we formalise the type-level properties corresponding to Def. 16. Type-level liveness means that all pending internal/external choices are eventually fired (via a message transmission or crash detection) – assuming *fairness* (Def. 17, based on *strong fairness of components* [32, Fact 2]) so all enabled message transmissions are eventually performed.

► **Definition 17** (Non-crashing, Fair, Live Paths (adapted from [17, Def. 4.4])). *A non-crashing path is a possibly infinite sequence of typing contexts $(\Gamma_n)_{n \in N}$, where $N = \{0, 1, 2, \dots\}$ is a set of consecutive natural numbers, and, $\forall n \in N$, $\Gamma_n \rightarrow \Gamma_{n+1}$.*

We say that a non-crashing path $(\Gamma_n)_{n \in N}$ is fair for session s iff, $\forall n \in N: \Gamma_n \xrightarrow{s[\mathbf{p}][\mathbf{q}]\mathbb{m}} \text{implies } \exists k, \mathbf{m}'$ such that $N \ni k \geq n$, and $\Gamma_k \xrightarrow{s[\mathbf{p}][\mathbf{q}]\mathbf{m}'} \Gamma_{k+1}$.

We say that a non-crashing path $(\Gamma_n)_{n \in \mathbb{N}}$ is **live for session** s iff, $\forall n \in \mathbb{N}$:

- (1) $\Gamma_n \xrightarrow{s[p]:q \oplus m(S)} \Gamma_{k+1}$ implies $\exists k, m'$ such that $N \ni k \geq n$ and $\Gamma_k \xrightarrow{s[p][q]m'} \Gamma_{k+1}$;
- (2) $\Gamma_n \xrightarrow{s[q]:p \& m(S)} \Gamma_{k+1}$ and $m \neq \text{crash}$ implies $\exists k, m'$ such that $N \ni k \geq n$ and $\Gamma_k \xrightarrow{s[p][q]m'} \Gamma_{k+1}$ or $\Gamma_k \xrightarrow{s[q] \circ p} \Gamma_{k+1}$.

► **Definition 18** (Typing Context Properties). Given a session s and a set of reliable roles \mathcal{R} , we say Γ is:

- (1) $(s; \mathcal{R})$ -**deadlock-free** iff $\Gamma \rightarrow_{\downarrow \setminus s; \mathcal{R}}^* \Gamma' \not\rightarrow$ implies $\forall s[p] \in \Gamma : \Gamma(s[p]) \leq \text{end}$ or $\Gamma(s[p]) = \text{stop}$ or $\exists q : \Gamma(s[p]) \leq q \& \text{crash}.T'$;
- (2) $(s; \mathcal{R})$ -**terminating** iff it is deadlock-free, and $\exists j$ finite such that $\forall n \geq j : \Gamma = \Gamma_0 \rightarrow_{\downarrow \setminus s; \mathcal{R}} \Gamma_1 \rightarrow_{\downarrow \setminus s; \mathcal{R}} \dots \rightarrow_{\downarrow \setminus s; \mathcal{R}} \Gamma_n$ implies $\Gamma_n \not\rightarrow$;
- (3) $(s; \mathcal{R})$ -**never-terminating** iff $\Gamma \rightarrow_{\downarrow \setminus s; \mathcal{R}}^* \Gamma'$ implies $\Gamma' \rightarrow$;
- (4) $(s; \mathcal{R})$ -**live** iff $\Gamma \rightarrow_{\downarrow \setminus s; \mathcal{R}}^* \Gamma'$ implies all non-crashing paths starting with Γ' which are fair for session s are also live for s .

► **Example 19.** Reliability assumptions \mathcal{R} can affect typing context properties, e.g. consider:

$$\Gamma = s[p]:\mu t_p.q \oplus \text{ok}.t_p, s[q]:\mu t_q.p \& \{ \text{ok}.t_q, \text{crash}.\mu t'_q.r \& \{ \text{ok}.t'_q, \text{crash}.\text{end} \} \}, s[r]:\mu t_r.q \oplus \text{ok}.t_r$$

If $\mathcal{R} = \emptyset$, Γ is safe and deadlock-free but *not* live: if p does *not* crash, r 's **ok** message is never received by q . If we have $\mathcal{R} = \{r\}$, Γ satisfies never-termination. Here, neither liveness nor termination can be satisfied by adding reliability assumptions. More examples in §C, Ex. 24.

We conclude by showing how the type-level properties in Def. 18 allow us to infer the corresponding process properties in Def. 16. The proof is available in [3].

► **Theorem 20** (Verification of Process Properties). Assume $\emptyset \cdot \Gamma \vdash P$, where Γ is $(s; \mathcal{R})$ -safe, $P \equiv \Pi_{p \in I} P_p$, and $\Gamma = \bigcup_{p \in I} \Gamma_p$ such that for each P_p , we have $\emptyset \cdot \Gamma_p \vdash P_p$. Further, assume that each P_p is either $\mathbf{0}$ (up to \equiv), or only plays p in s , by Γ_p . Then, for all $\varphi \in \{\text{deadlock-free, terminating, never-terminating, live}\}$, if Γ is $(s; \mathcal{R})$ - φ , then P is φ .

4 Verifying Type-Level Properties via Model Checking

In our generalised typing system, we prove subject reduction when a typing context satisfies a safety property (Def. 8); we then give examples of more refined typing context properties (Def. 18) and show how they are inherited by typed processes (Thm. 20). In this section, we highlight a major benefit of our theory: we show how such typing context behavioural properties can be verified using model checkers. We use our typing contexts and their semantics (including crashes and crash handling) as models, and we express our behavioural properties as modal μ -calculus formulæ; we then use a model checker (mCRL2 [4]) to verify whether a typing context enjoys a desired property.

Contexts as Models. We encode our typing contexts as mCRL2 processes, with LTS semantics that match Def. 7. To embed our optional reliability assumptions, the context encoding reflects the transition relation $\rightarrow_{\downarrow \setminus s; \mathcal{R}}$, so it never crashes any reliable role in \mathcal{R} .

Properties as Formulæ. A modal μ -calculus formula ϕ accepts or rejects a typing context Γ depending on the transition labels Γ can fire while reducing. We write $\Gamma \models \phi$ when a typing context Γ satisfies ϕ . Actions α range over transition labels in Def. 6; d (for **d**ata) ranges over sessions, roles, message labels, and types. Our formulæ ϕ follow a standard syntax:

$$\begin{array}{l}
 \text{[}\mu\text{-SAFE] } \Gamma \models \nu Z. \left(\forall s, p, q, m, m', S, S'. \left(\langle s[p]\text{stop} \rangle \top \wedge \langle s[q]:p\&m'(S') \rangle \top \Rightarrow \langle s[q]\odot p \rangle \top \right) \right. \\
 \left. \wedge \left(\langle s[p]:q\oplus m(S) \rangle \top \wedge \langle s[q]:p\&m'(S') \rangle \top \vee \langle s[q]\text{stop} \rangle \top \right) \Rightarrow \langle s[p][q]m \rangle \top \right) \wedge \phi_{\rightarrow}(Z) \\
 \text{[}\mu\text{-DF] } \Gamma \models \nu Z. \left(\left(\forall s, p, q, m. [s[p][q]m] \perp \wedge [s[p]\odot q] \perp \right) \Rightarrow \right. \\
 \left. \forall s, p, q, m, S. [s[p]:q\&m(S)] \perp \wedge [s[p]:q\oplus m(S)] \perp \right) \wedge \phi_{\rightarrow}(Z) \\
 \text{[}\mu\text{-TERM] } \Gamma \models \mu Z. \left(\left(\forall s, p, q, m. [s[p][q]m] \perp \wedge [s[p]\odot q] \perp \right) \Rightarrow \right. \\
 \left. \forall s, p, q, m, S. [s[p]:q\&m(S)] \perp \wedge [s[p]:q\oplus m(S)] \perp \right) \wedge \phi_{\rightarrow}(Z) \\
 \text{[}\mu\text{-NTERM] } \Gamma \models \nu Z. \left(\exists s, p, q, m. \langle s[p][q]m \rangle \top \vee \langle s[p]\odot q \rangle \top \right) \wedge \phi_{\rightarrow}(Z) \\
 \text{[}\mu\text{-LIVE] } \Gamma \models \nu Z. \left(\forall s, p, q. \left(\begin{array}{l} \phi_{in} = \left(\begin{array}{l} (\exists m, S. \langle s[q]:p\&m(S) \rangle \top) \Rightarrow \\ \mu Z'. \left(\begin{array}{l} \exists m. \langle s[p][q]m \rangle \top \vee \langle s[q]\odot p \rangle \top \\ \vee \exists p', q'. \left(\begin{array}{l} \exists m'. \langle s[p'] [q'] m' \rangle \top \vee \langle s[q'] \odot p' \rangle \top \\ \wedge \phi'_{\rightarrow}(s, p', q', Z') \end{array} \right) \end{array} \right) \end{array} \right) \right) \\ \wedge \phi_{out} = \forall m. \left(\begin{array}{l} (\exists S. \langle s[p]:q\oplus m(S) \rangle \top) \Rightarrow \\ \mu Z'. \left(\begin{array}{l} \langle s[p][q]m \rangle \top \\ \vee \exists p', q'. \left(\begin{array}{l} \exists m'. \langle s[p'] [q'] m' \rangle \top \vee \langle s[q'] \odot p' \rangle \top \\ \wedge \phi'_{\rightarrow}(s, p', q', Z') \end{array} \right) \end{array} \right) \end{array} \right) \right) \\ \wedge \phi_{\rightarrow}(Z) \end{array} \right) \right)
 \end{array}
 \end{array}$$

■ **Figure 5** Modal μ -Calculus Formulæ corresponding to Properties in Defs. 8 and 18, where $\phi_{\rightarrow}(Z) = \forall s, p, q. \phi'_{\rightarrow}(s, p, q, Z)$, and $\phi'_{\rightarrow}(s, p, q, Z) = \forall m. [s[p][q]m]Z \wedge [s[p]\not\downarrow]Z \wedge [s[p]\odot q]Z$.

$$\phi ::= \top \mid \perp \mid [\alpha]\phi \mid \langle \alpha \rangle \phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \Rightarrow \phi_2 \mid \mu Z. \phi \mid \nu Z. \phi \mid Z \mid \forall d. \phi \mid \exists d. \phi$$

Truth (\top) accepts any Γ ; falsity (\perp) accepts no Γ . The box (resp. diamond) modality, $[\alpha]\phi$ (resp. $\langle \alpha \rangle \phi$), requires that ϕ is satisfied in all cases (resp. some cases) after action α is fired. The least (resp. greatest) fixed point $\mu Z. \phi$ (resp. $\nu Z. \phi$) allows one to iterate ϕ for a finite (resp. infinite) number of times, where Z denotes a variable for iteration. Lastly, the forms $\phi_1 \Rightarrow \phi_2$, $\forall d. \phi$, and $\exists d. \phi$ denote implication, and universal and existential quantification.

In Fig. 5 we show the μ -calculus formulæ corresponding to our properties in Defs. 8 and 18. Compared to [29], such properties are more complex, since they cater for crashes and crash handling transitions. Recall Def. 8, and take a safety property φ : for $\varphi(\Gamma)$ to hold, clause $[S \rightarrow_i]$ requires that whenever Γ can transition to some Γ' (via $\rightarrow_i \setminus s; \mathcal{R}$), then $\varphi(\Gamma')$ also holds. To represent this clause in modal μ -calculus, we use fixed points for possibly infinite paths; in Fig. 5 we write $\phi_{\rightarrow}(Z)$ for following a fixed point Z via any transmission, crash,² or crash handling actions, and we define it as $\phi_{\rightarrow}(Z) = \forall s, p, q, m. [s[p][q]m]Z \wedge [s[p]\not\downarrow]Z \wedge [s[p]\odot q]Z$.

Safety ($[\mu\text{-SAFE}]$) requires (in its second implication) that whenever Γ can fire an input action, and either an output or $s[q]\text{stop}$ action, then Γ can also fire a message transmission, $s[p][q]m$. The first implication requires that, if Γ can fire a $s[p]\text{stop}$ action and an input action $s[q]:p\&m'(S')$, then Γ must be capable of firing a crash handling action, $s[q]\odot p$.

Deadlock-Freedom ($[\mu\text{-DF}]$) requires that, if Γ is unable to reduce further without crashing (via \rightarrow), then Γ can hold only **ended** or **stopped** endpoints. The antecedent of \Rightarrow characterises a context that is unable to reduce (since \rightarrow only allows for transmissions $s[p][q]m$ and crash detection $s[p]\odot q$); the consequent forbids the presence of any input $s[p]:q\&m(S)$ or output $s[p]:q\oplus m(S)$ transitions. By Def. 7, this means all session endpoints in Γ are **ended** or **stopped**.

Terminating ($[\mu\text{-TERM}]$) holds when Γ can reach a terminal configuration (i.e. cannot further reduce via \rightarrow) within a *finite* number of steps. Hence, the formula is similar to deadlock-freedom, except that it uses the *least* fixed point ($\mu Z \dots$) to ensure finiteness.

² Since our typing contexts encoded in mCRL2 produce $\rightarrow_i \setminus s; \mathcal{R}$ -transitions that never crash reliable roles in \mathcal{R} , our μ -calculus formulæ can follow *all* crash transitions; hence, the formulæ do not depend on \mathcal{R} .

$ \begin{aligned} & s[\mathbf{b1}]: s \oplus \text{req}(\text{Str}).s \& \{ \text{quote}(\text{Int}).\mathbf{b2} \oplus \text{split}(\text{Int}).\mathbf{b2} \& \{ \text{crash}.T_1 \}, \text{crash}.\mathbf{b2} \oplus T_{\text{ko}} \} \\ & \quad T_1 = s \& \{ \text{rp1}.s \oplus \{ \text{ok}.T_2, T_{\text{ko}} \}, \text{rp2}.T_2, \text{rp3}.s \& \{ \text{date}(\text{Str}).\text{end}, T_4 \}, T_4 \} \\ & \quad T_2 = s \oplus \text{addr}(\text{Str}).s \& \{ \text{date}(\text{Str}).\text{end}, T_4 \} \\ & s[\mathbf{b2}]: s \& \{ \text{quote}(\text{Int}).T_1, T_{\text{ko}}, \text{crash}.T_1 \} \\ & \quad T_1 = \mathbf{b1} \& \{ \text{split}(\text{Int}).s \oplus \{ \text{ok}.s \oplus \text{addr}(\text{Str}).s \& \{ \text{date}(\text{Str}).\text{end}, T_4 \}, T_{\text{ko}} \}, T_{\text{ko}}, \text{crash}.s \oplus T_{\text{ko}} \} \\ & s[\mathbf{s}]: \mathbf{b1} \& \{ \text{req}(\text{Str}).\mathbf{b1} \oplus \text{quote}(\text{Int}).\mathbf{b2} \oplus \text{quote}(\text{Int}).\mathbf{b2} \& \{ \text{ok}.T_1, T_{\text{ko}}, \text{crash}.\mathbf{b1} \oplus \text{rp1}.T_2 \}, \text{crash}.\mathbf{b2} \oplus T_{\text{ko}} \} \\ & \quad T_1 = \mathbf{b2} \& \{ \text{addr}(\text{Str}).\mathbf{b2} \oplus \text{date}(\text{Str}).\mathbf{b2} \& \{ \text{crash}.\mathbf{b1} \oplus \text{rp3}.T_4 \}, \text{crash}.\mathbf{b1} \oplus \text{rp2}.T_3 \} \\ & \quad T_2 = \mathbf{b1} \& \{ \text{ok}.T_3, T_{\text{ko}}, T_4 \} \quad T_3 = \mathbf{b1} \& \{ \text{addr}(\text{Str}).T_4, T_4 \} \quad T_4 = \mathbf{b1} \oplus \text{date}(\text{Str}).\text{end} \\ & \text{where: } T_4 = \text{crash}.\text{end} \quad T_{\text{ok}} = \text{ok}.\text{end} \quad T_{\text{ko}} = \text{ko}.\text{end} \end{aligned} $

■ **Figure 6** Two-Buyers protocol extended with crash-handling.

Never-Terminating ($[\mu\text{-NTERM}]$) requires that Γ can always keep reducing via \rightarrow transitions. Therefore, we require some transmission $s[\mathbf{p}][\mathbf{q}]\mathbf{m}$ or crash detection action $s[\mathbf{p}]\odot\mathbf{q}$ to be always fireable, even after some of the non-reliable roles crash.

Liveness ($[\mu\text{-LIVE}]$) requires that any enabled input/output action is triggered by a corresponding message transmission or crash detection, within a finite number of steps. For input actions (sub-formula ϕ_{in}): if an input $s[\mathbf{q}]:\mathbf{p}\&\mathbf{m}(S)$ is enabled (left of \Rightarrow), then, in a finite number of steps ($\mu Z' \dots$) involving *other* roles \mathbf{p}', \mathbf{q}' , a transmission $s[\mathbf{p}][\mathbf{q}]\mathbf{m}'$ or a crash detection $s[\mathbf{q}]\odot\mathbf{p}$ can be fired. For output actions, the sub-formula ϕ_{out} is similar. The μ -calculus formula embeds fairness (Def. 17) by finding *some* roles \mathbf{p}', \mathbf{q}' that, no matter how they interact (sub-formula ϕ'_{\rightarrow}), lead to the desired transmission or crash detection.

Tool Implementation and Example. To verify the properties in Fig. 5, we implement a prototype tool that extends `mpstk` [30] (based on the `mCRL2` model checker [4]) with support for our crash-stop semantics. The updated tool is available at:

<https://github.com/alcestes/mpstk-crash-stop>

We now illustrate how this new tool helps in writing correct session protocols with crash handling, and briefly discuss its performance.

In the *two-buyers protocol* from MPST literature [19], buyers $\mathbf{b1}$ and $\mathbf{b2}$ agree on splitting the cost of buying a book from seller \mathbf{s} . We tackle this protocol with crashes and *no reliability assumptions*: all roles may crash, and survivors must end the session correctly. The resulting *crash-tolerant two-buyers protocol* (Fig. 6) is much more complex than the one in the literature. In fact, the possibility of crashes introduces a variety of scenarios where different roles may be crashed (or not), hence the protocol needs many `crash` branches. The protocol exhibits two crash-handling patterns: *i*) exiting gracefully, and *ii*) recovery behaviour. The former occurs either when \mathbf{s} crashes or when $\mathbf{b1}$ crashes prior to the agreed split. The latter occurs should $\mathbf{b2}$ crash after the agreed split, whereupon $\mathbf{b2}$ concludes the transaction if both $\mathbf{b2}$ and \mathbf{s} do not crash. This behaviour is activated via a recovery type in $s[\mathbf{b1}]$, where the labels `rpn` represent the point at which $\mathbf{b2}$ crashed: `rp1` represents $\mathbf{b2}$ failing prior to confirmation with \mathbf{s} ; `rp2` corresponds to before the sending of `addr`; and `rp3` prior to receiving the `date`. Overlooking or mishandling some cases is easy; our tool spots such errors, so the protocol can be tweaked until all desired properties hold. We used our tool to verify the protocol: it has 1409 states and 10248 transitions; it is safe, deadlock-free, live, and it is terminating; it is *not* never-terminating. All properties verify within 100ms on a 4.20 GHz Intel Core i7-7700K CPU with 16 GB RAM. More experimental results can be found in §D.

5 Related Work, Conclusions, and Future Work

Previous Work on Failure Handling in Session Types. can be generally classified under two main approaches: *affine* and *coordinator model*. The former adapts session types to allow session endpoints to cease prematurely (e.g. by throwing an exception); the latter assumes reliable process coordination to handle failures.

Affine failure handling is first proposed in [24] for a π -calculus with binary sessions (i.e. two roles), and [14] presents a concurrent λ -calculus with binary sessions and exception handling; exceptions are also found in [7, 8]. These works model failures at the application level, via throw/catch constructs. Our key innovations are: (1) we model arbitrary failures (e.g. hardware failures); (2) we specify what to do when a failure is detected *at the type level*; (3) we support multiparty sessions; and (4) we seamlessly support handling the crash of a role while handling another role’s crash, whereas the *do-catch* constructs cannot be nested.

Coordinator model approaches include [1], which extends MPST with *optional blocks* where *default values* are used when communications fail; and [12], which uses synchronisation points to detect and handle failures. Both need processes to coordinate to handle failures. [33] extends MPST with a *try-handle* construct: a reliable coordinator detects and broadcasts failures, and the remaining processes proceed with failure handling. Unlike these works, we do *not* assume reliable processes, failure broadcasts, or coordination/synchronisation points.

Other papers address failures with different approaches. The recent work [26] annotates global and local types to specify which interactions may fail, and how (process crash, message loss). Their failure model is different from ours; and unlike us, they handle failures by continuing the protocol via *default branches and values*. Instead, our types include **crash** branches defining recovery behaviours that are only executed upon crash detection; further, by nesting such **crash** branches, we can specify different behaviours depending on which roles have crashed. [25] uses an MPST specification to build a dependency graph among running processes, supervise them, and restart them in case of failure. [34] utilises MPST to specify fault-tolerant, event-driven distributed systems, where processes are monitored and restarted if they fail; unlike our work, they require certain reliable roles, but their model tolerates false crash suspicions. More on the theory side, [6] presents a Curry-Howard interpretation of a language with binary session types and internal non-determinism, which is used to model failures (that are propagated to all relevant sessions, similarly to [14, 24]). Process calculi with *localities* have been proposed to model distributed systems with failures [2, 9, 27]; unlike our work, they do not have a typing system to verify failure handling.

Generalised Multiparty Session Type Systems. (introduced in [29]) depart from “classic” MPST [20] by not requiring top-down syntactic notions of protocol correctness (global types, projection, *etc.*); rather, they check behavioural predicates (safety, liveness, *etc.*) against (local) session types. [18] adopts the approach to model actor systems with explicit connections in their types [21]. By adopting this general framework, we support protocols not representable as global types in classic MPST (e.g. DNS in §1, two-buyers in §4, and all examples in §D, excepting Adder).

Model Checking Behavioural Types. [10] develops a behavioural type system for the π -calculus, and check LTL formulæ against such types. In [22], the type system combines typing and local analyses, with liveness properties verified via model checking. A similar approach is introduced in [29] for MPST. Regarding applications, [15, 23] verify behavioural types extracted from Go source code; and in [31], the **Effpi** Scala library assigns behavioural types to communicating programs. These works use a model checker to validate e.g. liveness through type-level properties, but do not support crashes or crash handling.

Conclusions and Future Work. We presented a multiparty session typing system for verifying processes with crash-stop failures. We model crashes and crash handling in a session π -calculus and its typing contexts, and prove type safety, protocol conformance, deadlock freedom and liveness. Our system is generalised in two ways: (1) it supports *optional* reliability assumptions, ranging from fully reliable (as in classic MPST), to fully unreliable (every process may crash); and (2) it is parametric on a behavioural property φ (validated by model checking) which can ensure deadlock-freedom, liveness, *etc. even in presence of crashes*. We also present a prototype implementation of our approach. As future work, we plan to study more crash models (e.g. crash-recover) and types of failure (e.g. link failures). We also plan to study the use of *asynchronous* global types for specifying protocols with failure handling – but unlike [26], we plan to support the type-level specification of dedicated recovery behaviours that are only executed upon crash detection.

References

- 1 Manuel Adameit, Kirstin Peters, and Uwe Nestmann. Session types for link failures. In Ahmed Bouajjani and Alexandra Silva, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 37th IFIP WG 6.1 International Conference, FORTE 2017, Held as Part of the 12th International Federated Conference on Distributed Computing Techniques, DisCoTec 2017, Neuchâtel, Switzerland, June 19-22, 2017, Proceedings*, volume 10321 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2017. doi:10.1007/978-3-319-60225-7_1.
- 2 Roberto M. Amadio. An asynchronous model of locality, failure and process mobility. In David Garlan and Daniel Le Métayer, editors, *Coordination Languages and Models, Second International Conference, COORDINATION '97, Berlin, Germany, September 1-3, 1997, Proceedings*, volume 1282 of *Lecture Notes in Computer Science*, pages 374–391. Springer, 1997. doi:10.1007/3-540-63383-9_92.
- 3 Adam D. Barwell, Alceste Scalas, Nobuko Yoshida, and Fangyi Zhou. Generalised Multiparty Session Types with Crash-Stop Failures, 2022. doi:10.48550/arXiv.2207.02015.
- 4 Olav Bunte, Jan Friso Groote, Jeroen J. A. Keiren, Maurice Laveaux, Thomas Neele, Erik P. de Vink, Wieger Wesselink, Anton Wijs, and Tim A. C. Willemse. The mCRL2 Toolset for Analysing Concurrent Systems. In Tomáš Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 21–39, Cham, 2019. Springer International Publishing.
- 5 Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011. doi:10.1007/978-3-642-15260-3.
- 6 Luís Caires and Jorge A. Pérez. Linearity, Control Effects, and Behavioral Types. In Hongseok Yang, editor, *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 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 10201 of *Lecture Notes in Computer Science*, pages 229–259. Springer, 2017. doi:10.1007/978-3-662-54434-1_9.
- 7 Sara Capecchi, Elena Giachino, and Nobuko Yoshida. Global escape in multiparty sessions. *Math. Struct. Comput. Sci.*, 26(2):156–205, 2016. doi:10.1017/S0960129514000164.
- 8 Marco Carbone, Kohei Honda, and Nobuko Yoshida. Structured Interactional Exceptions in Session Types. In Franck van Breugel and Marsha Chechik, editors, *CONCUR 2008 - Concurrency Theory, 19th International Conference, CONCUR 2008, Toronto, Canada, August 19-22, 2008. Proceedings*, volume 5201 of *Lecture Notes in Computer Science*, pages 402–417. Springer, 2008. doi:10.1007/978-3-540-85361-9_32.
- 9 Ilaria Castellani. Process algebras with localities. In Jan A. Bergstra, Alban Ponse, and Scott A. Smolka, editors, *Handbook of Process Algebra*, pages 945–1045. North-Holland / Elsevier, 2001. doi:10.1016/b978-044482830-9/50033-3.

- 10 Sagar Chaki, Sriram K. Rajamani, and Jakob Rehof. Types as models: Model checking message-passing programs. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '02, pages 45–57, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/503272.503278.
- 11 Tushar Deepak Chandra and Sam Toueg. Unreliable Failure Detectors for Reliable Distributed Systems. *J. ACM*, 43(2):225–267, March 1996. doi:10.1145/226643.226647.
- 12 Tzu-Chun Chen, Malte Viering, Andi Bejleri, Lukasz Ziarek, and Patrick Eugster. A Type Theory for Robust Failure Handling in Distributed Systems. In Elvira Albert and Ivan Lanese, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 36th IFIP WG 6.1 International Conference, FORTE 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings*, volume 9688 of *Lecture Notes in Computer Science*, pages 96–113. Springer, 2016. doi:10.1007/978-3-319-39570-8_7.
- 13 Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. Global progress for dynamically interleaved multiparty sessions. *MSCS*, 760, 2015. doi:10.1017/S0960129514000188.
- 14 Simon Fowler, Sam Lindley, J. Garrett Morris, and Sára Decova. Exceptional Asynchronous Session Types: Session Types without Tiers. *Proc. ACM Program. Lang.*, 3(POPL):28:1–28:29, 2019. doi:10.1145/3290341.
- 15 Julia Gabet and Nobuko Yoshida. Static Race Detection and Mutex Safety and Liveness for Go Programs. In Robert Hirschfeld and Tobias Pape, editors, *34th European Conference on Object-Oriented Programming (ECOOP 2020)*, volume 166 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:30, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ECOOP.2020.4.
- 16 Simon Gay and António Ravara. *Behavioural Types: From Theory to Tools*. River Publishers, Series in Automation, Control and Robotics, 2017. doi:10.13052/rp-9788793519817.
- 17 Silvia Ghilezan, Jovanka Pantović, Ivan Prokić, Alceste Scalas, and Nobuko Yoshida. Precise Subtyping for Asynchronous Multiparty Sessions. *Proc. ACM Program. Lang.*, 5(POPL), January 2021. doi:10.1145/3434297.
- 18 Paul Harvey, Simon Fowler, Ornela Dardha, and Simon J. Gay. Multiparty Session Types for Safe Runtime Adaptation in an Actor Language. In Anders Møller and Manu Sridharan, editors, *35th European Conference on Object-Oriented Programming (ECOOP 2021)*, volume 194 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:30, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ECOOP.2021.10.
- 19 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *POPL*, 2008. Full version in [20]. doi:10.1145/1328438.1328472.
- 20 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty Asynchronous Session Types. *J. ACM*, 63(1), 2016. doi:10.1145/2827695.
- 21 Raymond Hu and Nobuko Yoshida. Explicit Connection Actions in Multiparty Session Types. In *FASE*, 2017. doi:10.1007/978-3-662-54494-5_7.
- 22 Naoki Kobayashi and Davide Sangiorgi. A Hybrid Type System for Lock-Freedom of Mobile Processes. *TOPLAS*, 32(5), 2010. doi:10.1145/1745312.1745313.
- 23 Julien Lange, Nicholas Ng, Bernardo Toninho, and Nobuko Yoshida. A Static Verification Framework for Message Passing in Go Using Behavioural Types. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 1137–1148, 2018. doi:10.1145/3180155.3180157.
- 24 Dimitris Mostrous and Vasco T. Vasconcelos. Affine Sessions. *Logical Methods in Computer Science*, Volume 14, Issue 4, November 2018. doi:10.23638/LMCS-14(4:14)2018.
- 25 Rumyana Neykova and Nobuko Yoshida. Let It Recover: Multiparty Protocol-Induced Recovery. In *CC*, 2017. doi:10.1145/3033019.3033031.

- 26 Kirstin Peters, Uwe Nestmann, and Christoph Wagner. Fault-tolerant multiparty session types. In Mohammad Reza Mousavi and Anna Philippou, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 42nd IFIP WG 6.1 International Conference, FORTE 2022, Held as Part of the 17th International Federated Conference on Distributed Computing Techniques, DisCoTec 2022, Lucca, Italy, June 13-17, 2022, Proceedings*, volume 13273 of *Lecture Notes in Computer Science*, pages 93–113. Springer, 2022. doi:10.1007/978-3-031-08679-3_7.
- 27 James Riely and Matthew Hennessy. Distributed processes and location failures (extended abstract). In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *Automata, Languages and Programming, 24th International Colloquium, ICALP'97, Bologna, Italy, 7-11 July 1997, Proceedings*, volume 1256 of *Lecture Notes in Computer Science*, pages 471–481. Springer, 1997. doi:10.1007/3-540-63165-8_203.
- 28 Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2011. doi:10.1017/CB09780511777110.
- 29 Alceste Scalas and Nobuko Yoshida. Less is More: Multiparty Session Types Revisited. *Proc. ACM Program. Lang.*, 3(POPL):30:1–30:29, January 2019. doi:10.1145/3290343.
- 30 Alceste Scalas and Nobuko Yoshida. mpstk: the Multiparty Session Types ToolKit, 2019. Peer-reviewed artifact of [29]. (Latest version available at: <https://alcestes.github.io/mpstk>). doi:10.1145/3291638.
- 31 Alceste Scalas, Nobuko Yoshida, and Elias Benussi. Verifying Message-Passing Programs with Dependent Behavioural Types. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019*, pages 502–516, New York, NY, USA, 2019. Association for Computing Machinery. doi:10.1145/3314221.3322484.
- 32 Rob van Glabbeek, Peter Höfner, and Ross Horne. Assuming Just Enough Fairness to make Session Types Complete for Lock-freedom. In *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470531.
- 33 Malte Viering, Tzu-Chun Chen, Patrick Eugster, Raymond Hu, and Lukasz Ziarek. A Typing Discipline for Statically Verified Crash Failure Handling in Distributed Systems. In Amal Ahmed, editor, *Programming Languages and Systems*, pages 799–826, Cham, 2018. Springer International Publishing.
- 34 Malte Viering, Raymond Hu, Patrick Eugster, and Lukasz Ziarek. A Multiparty Session Typing Discipline for Fault-Tolerant Event-Driven Distributed Programming. *Proc. ACM Program. Lang.*, 5(OOPSLA), October 2021. doi:10.1145/3485501.
- 35 Fangyi Zhou, Francisco Ferreira, Raymond Hu, Romyana Neykova, and Nobuko Yoshida. Statically Verified Refinements for Multiparty Protocols. *Proc. ACM Program. Lang.*, 4(OOPSLA):148:1–148:30, 2020. doi:10.1145/3428216.

A Structural Congruence

The structural congruence relation of our MPST π -calculus, mentioned in Fig. 2, is formalised below. These rules are standard, and taken from [29]; the only extension is rule [C-CRASHELIM]. Here, $\text{fpv}(D)$ is the set of *free process variables* in D , and $\text{dpv}(D)$ is the set of *declared process variables* in D .

$$\begin{aligned}
P \mid Q &\equiv Q \mid P \quad [\text{C-PAR}] & (P \mid Q) \mid R &\equiv P \mid (Q \mid R) \quad [\text{C-ASSOC}] & P \mid \mathbf{0} &\equiv P \quad [\text{C-PARID}] \\
(\nu s) \mathbf{0} &\equiv \mathbf{0} \quad [\text{C-RESELIM}] & (\nu s) (\nu s') P &\equiv (\nu s') (\nu s) P \quad [\text{C-RESVAR}] \\
(\nu s) (P \mid Q) &\equiv P \mid (\nu s) Q \quad \text{if } s \notin \text{fc}(P) \quad [\text{C-RESLIFT}] & (\nu s) (s[\mathbf{p}_0] \dot{\downarrow} \mid \cdots \mid s[\mathbf{p}_n] \dot{\downarrow}) &\equiv \mathbf{0} \quad [\text{C-CRASHELIM}] \\
\text{def } D \text{ in } \mathbf{0} &\equiv \mathbf{0} \quad [\text{C-DEFELIM}] & \text{def } D \text{ in } (\nu s) P &\equiv (\nu s) (\text{def } D \text{ in } P) \quad \text{if } s \notin \text{fc}(D) \quad [\text{C-DEFLIFT}] \\
\text{def } D \text{ in } (P \mid Q) &\equiv (\text{def } D \text{ in } P) \mid Q \quad \text{if } \text{dpv}(D) \cap \text{fpv}(Q) = \emptyset \quad [\text{C-DEFPARLIFT}] \\
\text{def } D \text{ in } (\text{def } D' \text{ in } P) &\equiv \text{def } D' \text{ in } (\text{def } D \text{ in } P) \quad [\text{C-DEFORD}] \\
\text{if } (\text{dpv}(D) \cup \text{fpv}(D)) \cap \text{dpv}(D') &= (\text{dpv}(D') \cup \text{fpv}(D')) \cap \text{dpv}(D) = \emptyset
\end{aligned}$$

B Session Subtyping

We formalise our *subtyping* relation \leq in Def. 21 below. The relation is mostly standard [29, Def. 2.5], except for the new rule [SUB-stop], and the new (**highlighted**) side condition “ $|I| = 1 \implies \dots$ ” in rule [SUB-&]: this condition prevents the supertype from adding input branches to “pure” crash recovery external choices.

► **Definition 21** (Subtyping). *Given a standard subtyping $<$: for basic types (e.g. including `int` $<$: `real`), the session subtyping relation \leq is coinductively defined:*

$$\begin{aligned}
\frac{B <: B'}{B \leq B'} \quad [\text{SUB-B}] & \quad \frac{}{\text{end} \leq \text{end}} \quad [\text{SUB-end}] & \quad \frac{\forall i \in I \quad S'_i \leq S_i \quad T_i \leq T'_i}{\mathbf{p} \oplus \{\mathbf{m}_i(S_i).T_i\}_{i \in I \cup J} \leq \mathbf{p} \oplus \{\mathbf{m}_i(S'_i).T'_i\}_{i \in I}} \quad [\text{SUB-}\oplus] \\
\frac{}{\text{stop} \leq \text{stop}} \quad [\text{SUB-stop}] & \quad \frac{\forall i \in I \quad S_i \leq S'_i \quad T_i \leq T'_i \quad |I| = 1 \implies (\mathbf{m}_i \neq \text{crash} \text{ or } J = \emptyset)}{\mathbf{p} \& \{\mathbf{m}_i(S_i).T_i\}_{i \in I} \leq \mathbf{p} \& \{\mathbf{m}_i(S'_i).T'_i\}_{i \in I \cup J}} \quad [\text{SUB-}\&] \\
\frac{T[\mu\mathbf{t}.T/\mathbf{t}] \leq T'}{\mu\mathbf{t}.T \leq T'} \quad [\text{SUB-}\mu\text{L}] & \quad \frac{T \leq T'[\mu\mathbf{t}.T'/\mathbf{t}]}{T \leq \mu\mathbf{t}.T'} \quad [\text{SUB-}\mu\text{R}]
\end{aligned}$$

Rule [SUB-B] lifts \leq to basic types. The rest of the rules say that a subtype describes a more permissive session protocol w.r.t. its supertype. By rule [SUB- \oplus], the subtype of an internal choice allows for selecting from a wider set of message labels, and sending more generic payloads. By rule [SUB-&], the subtype of an external choice can support a smaller set of input message labels, and less generic payloads; the side condition “ $|I| = 1 \dots$ ” ensures that if the subtype only has a singleton **crash** branch, then the same applies to the supertype – hence, both subtype and supertype describe a “pure” crash recovery behaviour, and do not expect to receive any other input.³ By rules [SUB-end] and [SUB-stop], the types `end` and `stop` are only subtypes of themselves. Finally, rules [SUB- μ L] and [SUB- μ R] say that recursive types are related up to their unfolding.

C Additional Examples

► **Example 22.** We show an example of our crashing semantics. Processes P and Q below communicate on a session s ; P uses the endpoint $s[\mathbf{p}]$ to send an endpoint $s[\mathbf{r}]$ to role \mathbf{q} ; Q uses the endpoint $s[\mathbf{q}]$ to receive an endpoint x , then sends a message to role \mathbf{p} via x .

$$P = s[\mathbf{p}][\mathbf{q}] \oplus \mathbf{m}' \langle s[\mathbf{r}] \rangle . s[\mathbf{p}][\mathbf{r}] \& \mathbf{m}(x) \quad Q = s[\mathbf{q}][\mathbf{p}] \& \mathbf{m}'(x) . x[\mathbf{p}] \oplus \mathbf{m} \langle 42 \rangle$$

³ Notice, however, that rule [SUB-&] allows a supertype to have a **crash**-handling branch even when the subtype does not have one.

On a successful reduction (without crashes), we have:

$$\begin{aligned}
& (\nu s) (P \mid Q) \\
&= (\nu s) (s[\mathbf{p}][\mathbf{q}] \oplus m' \langle s[\mathbf{r}] \rangle . s[\mathbf{p}][\mathbf{r}] \& m(x) \mid s[\mathbf{q}][\mathbf{p}] \& m'(x) . x[\mathbf{p}] \oplus m \langle 42 \rangle) \\
&\rightarrow (\nu s) (s[\mathbf{p}][\mathbf{r}] \& m(x) \mid s[\mathbf{r}][\mathbf{p}] \oplus m \langle 42 \rangle) \\
&\rightarrow \mathbf{0}
\end{aligned}$$

Now, suppose that P crashes before sending; this gives rise to the reduction:

$$\begin{aligned}
& (\nu s) (P \mid Q) \\
&= (\nu s) (s[\mathbf{p}][\mathbf{q}] \oplus m' \langle s[\mathbf{r}] \rangle . s[\mathbf{p}][\mathbf{r}] \& m(x) \mid s[\mathbf{q}][\mathbf{p}] \& m'(x) . x[\mathbf{p}] \oplus m \langle 42 \rangle) \\
&\rightarrow (\nu s) (s[\mathbf{p}] \dagger \mid s[\mathbf{r}] \dagger \mid s[\mathbf{q}][\mathbf{p}] \& m'(x) . x[\mathbf{p}] \oplus m \langle 42 \rangle)
\end{aligned}$$

We can observe that when the sending process P crashes (by $[\mathbf{R}\text{-}\dagger\oplus]$), all endpoints in P (i.e. both $s[\mathbf{p}]$ and $s[\mathbf{r}]$) crash. If Q has a crash handling branch, it can be triggered via $[\mathbf{R}\text{-}\odot]$, suppose instead we have

$$Q' = s[\mathbf{q}][\mathbf{p}] \& \{m'(x) . x[\mathbf{p}] \oplus m \langle 42 \rangle, \text{crash.}\mathbf{0}\}$$

A crash handling reduction can trigger when P crashes:

$$\begin{aligned}
& (\nu s) (P \mid Q') \\
&= (\nu s) (s[\mathbf{p}][\mathbf{q}] \oplus m' \langle s[\mathbf{r}] \rangle . s[\mathbf{p}][\mathbf{r}] \& m(x) \mid s[\mathbf{q}][\mathbf{p}] \& \{m'(x) . x[\mathbf{p}] \oplus m \langle 42 \rangle, \text{crash.}\mathbf{0}\}) \\
&\rightarrow (\nu s) (s[\mathbf{p}] \dagger \mid s[\mathbf{r}] \dagger \mid s[\mathbf{q}][\mathbf{p}] \& \{m'(x) . x[\mathbf{p}] \oplus m \langle 42 \rangle, \text{crash.}\mathbf{0}\}) \\
&\rightarrow (\nu s) (s[\mathbf{p}] \dagger \mid s[\mathbf{r}] \dagger \mid \mathbf{0})
\end{aligned}$$

► **Example 23.** Recall the types of the DNS example in § 1:

$$\begin{aligned}
T'_p &= \mathbf{q} \oplus \text{req.}\mathbf{q} \& \left\{ \begin{array}{l} \text{res.end} \\ \text{crash.r} \oplus \text{req.r} \& \text{res.end} \end{array} \right\} & \begin{array}{l} T'_q = \mathbf{p} \& \text{req.p} \oplus \text{res.end} \\ T'_r = \mathbf{q} \& \text{crash.p} \& \text{req.p} \oplus \text{res.end} \end{array}
\end{aligned}$$

Now, consider the following typing context, containing such types:

$$\Gamma = s[\mathbf{p}]:T'_p, s[\mathbf{q}]:T'_q, s[\mathbf{r}]:T'_r$$

Such Γ is $(s; \{\mathbf{p}, \mathbf{r}\})$ -safe. We can verify it by checking its reductions. When no crashes occur, we have the following two reductions, where each reductum satisfies Def. 8:

$$\begin{aligned}
\Gamma &\rightarrow_{\dagger \setminus s; \{\mathbf{p}, \mathbf{r}\}} s[\mathbf{p}]:\mathbf{q} \& \left\{ \begin{array}{l} \text{res} \\ \text{crash.r} \oplus \text{req.r} \& \text{res} \end{array} \right\}, s[\mathbf{q}]:\mathbf{p} \oplus \text{res}, s[\mathbf{r}]:T'_r \\
&\rightarrow_{\dagger \setminus s; \{\mathbf{p}, \mathbf{r}\}} s[\mathbf{p}]:\text{end}, s[\mathbf{q}]:\text{end}, s[\mathbf{r}]:T'_r
\end{aligned}$$

In the case where \mathbf{q} crashes immediately, we have:

$$\begin{aligned}
\Gamma &\rightarrow_{\dagger \setminus s; \{\mathbf{p}, \mathbf{r}\}} s[\mathbf{p}]:T'_p, s[\mathbf{q}]:\text{stop}, s[\mathbf{r}]:T'_r \\
&\rightarrow_{\dagger \setminus s; \{\mathbf{p}, \mathbf{r}\}} s[\mathbf{p}]:\mathbf{q} \& \{\text{res}, \text{crash.r} \oplus \text{req.r} \& \text{res}\}, s[\mathbf{q}]:\text{stop}, s[\mathbf{r}]:T'_r \\
&\rightarrow_{\dagger \setminus s; \{\mathbf{p}, \mathbf{r}\}} s[\mathbf{p}]:\mathbf{r} \oplus \text{req.r} \& \text{res}, s[\mathbf{q}]:\text{stop}, s[\mathbf{r}]:T'_r \\
&\rightarrow_{\dagger \setminus s; \{\mathbf{p}, \mathbf{r}\}} s[\mathbf{p}]:\mathbf{r} \oplus \text{req.r} \& \text{res}, s[\mathbf{q}]:\text{stop}, s[\mathbf{r}]:\mathbf{p} \& \text{req.p} \oplus \text{res} \\
&\rightarrow_{\dagger \setminus s; \{\mathbf{p}, \mathbf{r}\}} s[\mathbf{p}]:\mathbf{r} \& \text{res}, s[\mathbf{q}]:\text{stop}, s[\mathbf{r}]:\mathbf{p} \oplus \text{res} \\
&\rightarrow_{\dagger \setminus s; \{\mathbf{p}, \mathbf{r}\}} s[\mathbf{p}]:\text{end}, s[\mathbf{q}]:\text{stop}, s[\mathbf{r}]:\text{end}
\end{aligned}$$

and each reductum satisfies Def. 8. The case where \mathbf{q} crashes after receiving the **request** is similar. There are no other crash reductions to consider, since \mathbf{p} and \mathbf{r} are reliable.

► **Example 24.** We illustrate safety, deadlock-freedom, liveness, termination, and never-termination over typing contexts via a series of small examples. We first consider the typing context $\Gamma_A = \Gamma_{Ap}, \Gamma_{Aq}, \Gamma_{Ar}$ where:

$$\begin{aligned}\Gamma_{Ap} &= s[p]:\mu t_p.q \oplus \{ok.q \& \{ok.t_p, ko.end, crash.end\}, ko.end\} \\ \Gamma_{Aq} &= s[q]:\mu t_q.p \& \{ok.p \oplus \{ok.t_q, ko.end\}, ko.end, crash.r \oplus ok.end\} \\ \Gamma_{Ar} &= s[r]:p \& \{crash.q \& \{ok.end, crash.end\}\}\end{aligned}$$

If we assume that all roles in Γ_A are unreliable, Γ_A is safe since its inputs/outputs are dual. However, Γ_A is *neither* deadlock-free *nor* live since it is possible for p to crash immediately before q sends ko to p . In such cases, q will *not* detect that p has crashed (since we only detect crashes on receive actions) and terminate *without* sending a message to the backup process r . This results in a deadlock because r *will* detect that p has crashed, and *will* expect a message from q .

We observe that changing the reliability assumptions, without changing the typing context, may influence whether a typing context property holds. For example, consider the typing context $\Gamma_B = \Gamma_{Bp}, \Gamma_{Bq}, \Gamma_{Br}$ where:

$$\begin{aligned}\Gamma_{Bp} &= s[p]:\mu t_p.q \oplus ok.t_p \\ \Gamma_{Bq} &= s[q]:\mu t_q.p \& \{ok.t_q, crash.\mu t'_q.r \& \{ok.t'_q, crash.end\}\} \\ \Gamma_{Br} &= s[r]:\mu t_r.q \oplus ok.t_r\end{aligned}$$

If we assume that all roles are unreliable, Γ_B is safe and deadlock-free but *not* live – because p may never crash, and in this case, r 's outputs are never received by q . Notably, Γ_B is *not* never-terminating because if both p and r crash, then the surviving q can reach end ; however, if we assume that just r is reliable (i.e. $\mathcal{R} = \{r\}$), then Γ_B becomes also never-terminating – because even if both p and q crash, role r can keep running by sending forever ok messages that are lost (by rule $[\Gamma \not\vdash m]$ in Fig. 3).

Notice that, in the case of Γ_B , we are unable to make liveness hold purely via combinations of reliable roles: this is because (unless p crashes) r 's output will never be received by q , irrespective of reliability assumptions. The typing context itself must instead be adapted; for example, by only permitting r to send once it has detected that p has crashed.

Instead, in the case of Γ_A , we *can* obtain liveness by adjusting the reliability assumptions: in fact, if we assume $r \in \mathcal{R}$, then Γ_A is both deadlock-free and live.

Finally, consider the typing context $\Gamma_C = \Gamma_{Cp}, \Gamma_{Cq}, \Gamma_{Cr}$ where:

$$\begin{aligned}\Gamma_{Cp} &= s[p]:q \oplus m_1.q \& \{m_2.end, crash.\mu t_p.r \oplus ok.t_p\} \\ \Gamma_{Cq} &= s[q]:p \& \{m_1.p \oplus m_2.end\} \\ \Gamma_{Cr} &= s[r]:p \& \{crash.\mu t_q.p \& \{ok.t_q\}\}\end{aligned}$$

Γ_C satisfies safety, deadlock-freedom, and termination when *all* roles are assumed to be reliable. However, should we instead assume that only p is reliable, then Γ_C does not satisfy termination. Since external choices in Γ_C do not feature a crash-handling branch when receiving from p , should no roles be assumed reliable, Γ_C satisfies only safety.

D Tool Evaluation

To verify the properties in Fig. 5, we extend the Multiparty Session Types toolKit (mpstk) [30], which uses the mCRL2 model checker [4]. Our extended tool is available at:

<https://github.com/alcestes/mpstk-crash-stop>

(α)	$s[p]:q \oplus req.q \& \{res.end, crash.r \oplus req.r \& res.end\}$ $s[q]:p \& req.p \oplus res.end$ $s[r]:q \& crash.p \& req.p \oplus res.end$
(β)	$s[p]:\mu t.q \oplus add(Int).q \oplus \{add(Int).q \& \{res(Int).t, T_i\}, ko.q \& \{T_{ko}, T_i\}\}$ $s[q]:\mu t.p \& \{add(Int).p \& \{add(Int).p \oplus res(Int).t, ko.p \oplus T_{ko}, T_i\}, T_i\}$
(γ)	$s[b1]:s \oplus r(Str).s \& \{q(Int).b2 \oplus s(Int).b2 \& \{crash.T_1\}, crash.b2 \oplus T_{ko}\}$ $T_1 = s \& \{rp1.s \oplus \{ok.T_2, T_{ko}\}, rp2.T_2, rp3.s \& \{d(Str).end, T_i\}, T_i\}$ $T_2 = s \oplus a(Str).s \& \{d(Str).end, T_i\}$ $s[b2]:s \& \{q(Int).T_1, T_{ko}, crash.T_1\}$ $T_1 = b1 \& \{s(Int).s \oplus \{ok.s \oplus a(Str).s \& \{d(Str).end, T_i\}, T_{ko}\}, T_{ko}, crash.s \oplus T_{ko}\}$ $s[s]:b1 \& \{r(Str).b1 \oplus q(Int).b2 \oplus q(Int).b2 \& \{ok.T_1, T_{ko}, crash.b1 \oplus rp1.T_2\}, crash.b2 \oplus T_{ko}\}$ $T_1 = b2 \& \{a(Str).b2 \oplus d(Str).b2 \& \{crash.b1 \oplus rp3.T_4\}, crash.b1 \oplus rp2.T_3\}$ $T_2 = b1 \& \{ok.T_3, T_{ko}, T_i\} \quad T_3 = b1 \& \{a(Str).T_4, T_i\} \quad T_4 = b1 \oplus d(Str).end$
(δ)	$s[n]:c \& \{o(Int).\mu t.c \oplus g.c \oplus \{o(Int).c \& \{o(Int).t, ok.c \oplus T_{ok}, T_{ko}, T_i\}, ok.c \& \{T_{ok}, T_i\}, T_{ko}\}, T_i\}$ $s[c]:n \oplus o(Int).\mu t.o.n \& \{g.n \& \{o(Int).T_1, ok.n \& \{crash.b \oplus T_{ok}\}, T_{ko}, crash.T_2\}, crash.T_2\}$ $T_1 = n \oplus \{o(Int).t.o, ok.n \& \{T_{ok}, crash.T_2\}, ko.n \& \{crash.b \oplus T_{ko}\}\}$ $T_2 = b \oplus o(Int).\mu t1.b \& \{g.b \& \{o(Int).b \oplus \{o(Int).t1, ok.b \& \{T_{ok}\}, T_{ko}\}, ok.b \& \{T_{ok}\}, T_{ko}\}\}$ $s[b]:n \& \{crash.c \& \{o(Int).\mu t.c \oplus g.c \oplus \{o(Int).T_1, ok.T_2, T_{ko}\}, T_{ok}, T_{ko}, T_i\}\}$ $T_1 = c \& \{o(Int).t, ok.c \oplus T_{ok}, T_{ko}\} \quad T_2 = c \& \{ok.c \oplus T_{ok}, T_i\}$
(ε)	$s[p]:q \oplus data(Str).r \oplus data(Str).end$ $s[q]:p \& \{data(Str).p \& \{crash.r \& \{h.r \oplus data(Str).end, T_i\}\}, crash.r \& \{req.r \oplus T_{ko}, T_i\}\}$ $s[r]:p \& \{data(Str).end, crash.q \oplus req.q \& \{data(Str).end, T_{ko}, T_i\}\}$

■ **Figure 7** Typing contexts for (α) DNS, (β) Adder, (γ) TwoBuyers, (δ) Negotiate, and (ε) Broadcast. Roles **p** and **q** of DNS, and **b** of Negotiate are reliable; all other roles are unreliable. Let $T_i = crash.end$, $T_{ko} = ko.end$, and $T_{ok} = ok.end$.

We evaluate our approach with 5 examples: DNS, from § 1; Adder, TwoBuyers, and Negotiate, extended from the session type literature [35] with crashes and crash handling behaviour; and Broadcast, inspired by the reliable broadcast algorithms in [5, Ch. 3]. The full typing contexts for each example are given in Fig. 7. We model and verify both fully reliable and (partially) unreliable versions of each example. In all examples, we show how the introduction of unreliability leads to an increase of model sizes and verification times. The increased model size reflects how the addition of crash handling can complicate even simple protocols, and motivates the use of automatic model checking. Still, we show that the verification of our examples always completes in less than 100 ms.

D.1 Description of the Examples in Figure 7

DNS is the example described in § 1. The example demonstrates both backup processes and optional reliability assumptions.

Adder demonstrates a minimal extension of the fully reliable protocol, in which **q** receives two numbers from **p**, sums them, and communicates the result to **p**. In our extension, both roles are unreliable and the protocol ends when a crash is detected. It satisfies safety, deadlock-freedom, and liveness.

TwoBuyers is the example described in § 4. It assumes that both the seller and buyers **b1** and **b2** are unreliable. In cases where the split has been agreed upon, and **b2** has crashed, **b1** concludes the sale. It satisfies safety, deadlock-freedom, liveness, and terminating. This form of TwoBuyers is not projectable from a global type, since **b1** would need to be informed on conclusion of a sale. TwoBuyers uses recovery behaviour in order to satisfy deadlock-freedom. Finally, TwoBuyers demonstrates the flexibility of crash-handling that our approach permits: **b2** does not alter its behaviour having detected that **s** has crashed (i.e. continues as T_1), instead leaving **b1** to instigate crash-handling behaviour.

■ **Table 1** Average times (in milliseconds \pm std. dev.) for the verification of DNS (α), Adder (β), TwoBuyers (γ), Negotiate (δ), and Broadcast (ε) in Fig. 7 over safety (safe), deadlock-freedom (df), liveness (live), never-terminating (nterm) and terminating (term). Each example has two rows of measurements, varying the sets of reliable roles \mathcal{R} : either zero/one/two reliable roles (first row), or all reliable roles (second row). (Benchmarking specs: Intel Core i7-7700K CPU, 4.20 GHz, 16 GB RAM, mCRL2 202106.0 invoked 30 times with: `pbes2bool --solve-strategy=2.`)

	\mathcal{R}	states	transitions	safe	df	live	nterm	term
(α)	$\{\mathbf{p}, \mathbf{r}\}$	101	427	$12.28 \pm 1\%$	$17.14 \pm 1\%$	$11.24 \pm 1\%$	$15.47 \pm 0\%$	$12.33 \pm 0\%$
	\mathfrak{R}	10	15	$7.61 \pm 1\%$	$8.23 \pm 1\%$	$7.46 \pm 1\%$	$7.78 \pm 1\%$	$7.6 \pm 1\%$
(β)	\emptyset	37	159	$12.43 \pm 0\%$	$15.74 \pm 0\%$	$12.24 \pm 1\%$	$14.46 \pm 0\%$	$12.06 \pm 1\%$
	\mathfrak{R}	26	56	$8.92 \pm 2\%$	$10.06 \pm 0\%$	$8.71 \pm 1\%$	$9.42 \pm 0\%$	$8.79 \pm 0\%$
(γ)	\emptyset	1409	10248	$45.6 \pm 0\%$	$88.26 \pm 0\%$	$31.33 \pm 0\%$	$77.2 \pm 0\%$	$45.65 \pm 0\%$
	\mathfrak{R}	169	510	$11.12 \pm 1\%$	$15.94 \pm 0\%$	$10.9 \pm 1\%$	$12.19 \pm 0\%$	$11.06 \pm 1\%$
(δ)	$\{\mathbf{b}\}$	1089	8106	$34.61 \pm 0\%$	$55.07 \pm 0\%$	$25.69 \pm 0\%$	$47.46 \pm 0\%$	$26.04 \pm 0\%$
	\mathfrak{R}	50	157	$10.17 \pm 0\%$	$12.7 \pm 0\%$	$9.93 \pm 0\%$	$11.33 \pm 0\%$	$9.72 \pm 0\%$
(ε)	\emptyset	161	925	$17.99 \pm 1\%$	$28.13 \pm 0\%$	$14.08 \pm 0\%$	$25.72 \pm 1\%$	$17.74 \pm 0\%$
	\mathfrak{R}	13	25	$7.85 \pm 3\%$	$8.65 \pm 1\%$	$7.7 \pm 0\%$	$8.12 \pm 1\%$	$7.85 \pm 0\%$

Negotiate introduces a (reliable) backup negotiator to the version found in the literature.

During normal operation, a **c**lient will send an opening offer to a **n**egotiator. Both **c** and **n** can then choose to repeatedly exchange counter offers until the other accepts the offer, or rejects it outright, bringing the protocol to an end. In our extension, should the **c**ustomer detect that the original **n**egotiator crashes, the **b**ackup negotiator activates and continues the negotiation with **c**. The example satisfies safety, deadlock-freedom, and liveness. Recovery actions are necessary for **c** in two locations in order to avoid deadlocks: it is otherwise possible for an **o**ffer to be declined or agreed upon, then for **n** to crash without **c** noticing; this results in **b** activating, and expecting a message from the terminated **c**.

Broadcast contains an unreliable broadcaster **p** attempting to send **d**ata to two receivers **q** and **r**. In cases where **p** crashes, **r** requests the data from **q**, who responds with the data it received before **p** crashed, or with **ko** when **p** crashed immediately. The example is not projectable from a global type, since **q** would otherwise require a message from **r** even when **p** had not crashed. **Broadcast** satisfies safety, deadlock-freedom, liveness, and termination. As in **Negotiate**, recovery behaviour is necessary for **Broadcast** to satisfy deadlock-freedom.

Notably, **Adder**, **TwoBuyers** and **Broadcast** have *no* reliability assumptions: any role may crash at any point. Barring **Adder**, our examples cannot be written using *global types* in the session types literature. This demonstrates the flexibility of our generalised MPST system over the classic one. Moreover, the examples include the use of failover processes (**DNS** and **Negotiate**) and complex recovery behaviour (**TwoBuyers**, **Negotiate**, and **Broadcast**), thus showcasing the expressivity of our approach.

D.2 Experimental Results

We applied our extended implementation of `mpstk` to the examples in Fig. 7. Table 1 gives the full set of verification times, reported in milliseconds with standard deviations, where each time is an average of 30 runs.

For each example, we give verification times for both the typing contexts in Fig. 7 and a corresponding fully reliable version (i.e. where all roles in the protocol are reliable; $\mathcal{R} = \mathfrak{R}$). For **Adder**, **TwoBuyers**, and **Negotiate**, we use the standard protocol definitions from the literature. For **DNS** and **Broadcast**, we omit crash-handling branches. For **DNS**, this has the consequence of removing the backup role \mathbf{r} entirely.

All examples satisfy safety, deadlock-freedom, and liveness; **Adder** and **Broadcast** satisfy termination; no example satisfies never-termination.


Unsurprisingly, all examples demonstrate an increase in verification times and the number of states and transitions when comparing unreliable to reliable versions. Even **Adder**, which represents minimal crash-handling, demonstrates relevant increases to the number of states and transitions: this is a direct consequence of the unreliable roles, and the resulting generation of crash and crash-detection transitions in the LTS generated by mCRL2. Verification times also increase because the verified properties follow crash and communication actions, thus requiring the exploration of a larger state space compared to the fully reliable versions.

Nevertheless, our verification times do not increase as quickly as the state space grows, and are always under 100 ms. This is because our μ -calculus formulæ only follow communication, crash, and crash detection transitions, and thus, their verification may not need to follow every possible transition into every state. This suggests greater scalability of the approach that would otherwise be suggested by the size of the state space. This also lends greater motivation to the use of model checkers, as it is infeasible to manually determine the properties of a large LTS with complex crash-handling behaviour.

An Infinitary Proof Theory of Linear Logic Ensuring Fair Termination in the Linear π -Calculus

Luca Ciccone  

University of Torino, Italy

Luca Padovani  

University of Torino, Italy

Abstract

Fair termination is the property of programs that may diverge “in principle” but that terminate “in practice”, *i.e.* under suitable fairness assumptions concerning the resolution of non-deterministic choices. We study a conservative extension of μMALL^∞ , the infinitary proof system of the multiplicative additive fragment of linear logic with least and greatest fixed points, such that cut elimination corresponds to fair termination. Proof terms are processes of πLIN , a variant of the linear π -calculus with (co)recursive types into which binary and (some) multiparty sessions can be encoded. As a result we obtain a behavioral type system for πLIN (and indirectly for session calculi through their encoding into πLIN) that ensures fair termination: although well-typed processes may engage in arbitrarily long interactions, they are *fairly* guaranteed to eventually perform all pending actions.

2012 ACM Subject Classification Theory of computation \rightarrow Linear logic; Theory of computation \rightarrow Process calculi; Theory of computation \rightarrow Program analysis

Keywords and phrases Linear π -calculus, Linear Logic, Fixed Points, Fair Termination

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.36

Related Version *Full Version:* <https://arxiv.org/abs/2207.03749> [10]

Acknowledgements We are grateful to Simona Ronchi Della Rocca and Francesco Dagnino for their comments on an early draft of this paper, to Stephanie Balzer for having provided pointers to related work and to the anonymous CONCUR reviewers for their questions and detailed feedback.

1 Introduction

The *linear π -calculus* [28] is a typed refinement of Milner’s π -calculus in which linear channels can be used only once, for a one-shot communication. As it turns out, the linear π -calculus is the fundamental model underlying a broad family of communicating processes. In particular, all *binary sessions* [22, 23, 25] and some *multiparty sessions* [24] can be encoded into the linear π -calculus [27, 5, 11, 8]. Sessions are private communication channels linking two or more processes and whose usage is disciplined by a *session type*, a type representing a structured communication protocol. In all session type systems, session endpoints are *linearized channels* that can be used repeatedly but in a sequential manner. The key insight for encoding sessions into the linear π -calculus is to encode linearized channels in a continuation-passing style: when some payload is transmitted over a linear channel, it can be paired with another linear channel (the continuation) on which the subsequent interaction takes place.

In this work we propose a type system for πLIN , a linear π -calculus with (co)recursive types, such that well-typed processes are *fairly terminating*. Fair termination [21, 17] is a liveness property stronger than *lock freedom* [26, 32] – *i.e.* the property that every pending action can be eventually performed – but weaker than *termination*. In particular, a fairly terminating program may diverge, but all of its infinite executions are considered “unfair” – read impossible or unrealistic – and so they can be ignored insofar termination is concerned. A simple example of fairly terminating program is that modeling a repeated interaction



© Luca Ciccone and Luca Padovani;
licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 36; pp. 36:1–36:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

between a buyer and a seller in which the buyer may either pay the seller and terminate or may add an item to the shopping cart and then repeat the same behavior. In principle, there is an execution of the program in which the buyer keeps adding items to the shopping cart and never pays. In practice, this behavior is considered unfair and the program terminates under the fairness assumption that the buyer eventually pays the seller.

Our type system is a conservative extension of μMALL^∞ [3, 16, 2], the infinitary proof system for the multiplicative additive fragment of linear logic with least and greatest fixed points. In fact, the modifications we make to μMALL^∞ are remarkably small: we add one (standard) rule to deal with *non-deterministic choices*, those performed autonomously by a process, and we relax the validity condition on μMALL^∞ proofs so that it only considers the “fair behaviors” of the program it represents. The fact that there is such a close correspondence between the typing rules of πLIN and the inference rules of μMALL^∞ is not entirely surprising. After all, there have been plenty of works investigating the relationship between π -calculus terms and linear logic proofs, from those of Abramsky [1], Bellin and Scott [4] to those on the interpretation of linear logic formulas as session types [15, 40, 6, 30, 37, 35]. Nonetheless, we think that the connection between πLIN and μMALL^∞ stands out for two reasons. First, πLIN is conceptually simpler and more general than the session-based calculi that can be encoded in it. In particular, all the session calculi based on linear logic rely on an asymmetric interpretation of the multiplicative connectives \otimes and \wp so that $\varphi \otimes \psi$ (respectively, $\varphi \wp \psi$) is the type of a session endpoint used for sending (respectively, receiving) a message of type φ and then used according to ψ . In our setting, the connectives \otimes and \wp retain their symmetry since we interpret $\varphi \otimes \psi$ and $\varphi \wp \psi$ formulas as the output/input of pairs, in the same spirit of the original encoding of linear logic proofs proposed by Bellin and Scott [4]. This interpretation gives πLIN the ability of modeling *bifurcating protocols* of which binary sessions are just a special case. The second reason why πLIN and μMALL^∞ get along has to do with the cut elimination result for μMALL^∞ . In finitary proof systems for linear logic, cut elimination may proceed by removing *topmost cuts*. In μMALL^∞ there is no such notion as a topmost cut since μMALL^∞ proofs may be infinite. As a consequence, the cut elimination result for μMALL^∞ is proved by eliminating *bottom-most cuts* [2]. This strategy fits perfectly with the reduction semantics of πLIN – and that of any other conventional process calculus, for that matter – whereby reduction rules act only on the exposed (*i.e.* unguarded) part of processes but not behind prefixes. As a result, the reduction semantics of πLIN is completely ordinary, unlike other logically-inspired process calculi that incorporate commuting conversions [40, 30], perform reductions behind prefixes [35] or swap prefixes [4].

In previous work [9] we have proposed a type system ensuring the fair termination of binary sessions. The present work achieves the same objective using a more basic process calculus and exploiting its strong logical connection with μMALL^∞ . In fact, the soundness proof of our type system piggybacks on the cut elimination property of μMALL^∞ . Other session typed calculi based on linear logic with fixed points have been studied by Lindley and Morris [30] and by Derakhshan and Pfenning [14, 13]. The type systems described in these works respectively guarantee termination and strong progress, whereas our type system guarantees fair termination which is somewhat in between these properties. Overall, our type system seems to hit a sweet spot: on the one hand, it is deeply rooted in linear logic and yet it can deal with common communication patterns (like the buyer/seller interaction described above) that admit potentially infinite executions and therefore are out of scope of other logic-inspired type systems; on the other hand, it guarantees lock freedom [26, 32], strong progress [14, Theorem 12.3] and also termination, under a suitable fairness assumption.

■ **Table 1** Syntax of πLIN .

$P, Q ::= x \leftrightarrow y$	link	$(x)(P \parallel Q)$	composition
$\text{case } x\{\}$	empty input	$P \oplus Q$	choice
$x().P$	unit input	$\bar{x}()$	unit output
$x(z, y).P$	pair input	$\bar{x}(z, y)(P \parallel Q)$	pair output
$\text{case } x(y)\{P, Q\}$	sum input	$\text{in}_i \bar{x}(y).P$	sum output $i \in \{1, 2\}$
$\text{corec } x(y).P$	corecursion	$\text{rec } \bar{x}(y).P$	recursion

The paper continues as follows. Section 2 presents πLIN and the fair termination property ensured by our type system. Section 3 describes πLIN types, which are suitably embellished μMALL^∞ formulas. Section 4 describes the inference rules of μMALL^∞ rephrased as typing rules for πLIN . Section 5 identifies the valid typing derivations and states the properties of well-typed processes. Section 6 discusses related work in more detail and Section 7 presents ideas for future developments. Supplementary material, additional examples and proofs can be found in the long version of the paper [10].

2 Syntax and Semantics of πLIN

In this section we define syntax and reduction semantics of πLIN , a variant of the linear π -calculus [28] in which all channels are meant to be used for *exactly* one communication. The calculus supports (co)recursive data types built using units, pairs and disjoint sums. These data types are known to be the essential ingredients for the encoding of sessions in the linear π -calculus [27, 11, 38].

We assume given an infinite set of *channels* ranged over by x, y and z . πLIN processes are coinductively generated by the productions of the grammar shown in Table 1 and their informal meaning is given below. A *link* $x \leftrightarrow y$ acts as a *linear forwarder* [18] that forwards a single message either from x to y or from y to x . The uncertainty in the direction of the message is resolved once the term is typed and the polarity of the types of x and y is fixed (Section 4). The term $\text{case } x\{\}$ represents a process that receives an empty message from x and then fails. This form is only useful in the metatheory: the type system guarantees that well-typed processes never fail, since it is not possible to send empty messages. The term $\bar{x}()$ models a process that sends the unit on x , effectively indicating that the interaction is terminated, whereas $x().P$ models a process that receives the unit from x and then continues as P . The term $\bar{x}(y, z)(P \parallel Q)$ models a process that creates two new channels y and z , sends them in a pair on channel x and then forks into two parallel processes P and Q . Dually, $x(y, z).P$ models a process that receives a pair containing two channels y and z from channel x and then continues as P . The term $\text{in}_i \bar{x}(y).P$ models a process that creates a new channel y and sends $\text{in}_i(y)$ (that is, the i -th injection of y in a disjoint sum) on x . Dually, $\text{case } x(y)\{P_1, P_2\}$ receives a disjoint sum from channel x and continues as either P_1 or P_2 depending on the tag in_i it has been built with. For clarity, in some examples we will use more descriptive labels such as **add** and **pay** instead of in_1 and in_2 . The terms $\text{rec } \bar{x}(y).P$ and $\text{corec } x(y).P$ model processes that respectively send and receive a new channel y and then continue as P . They do not contribute operationally to the interaction being modeled, but they indicate the points in a program where (co)recursive types are unfolded. A term $(x)(P \parallel Q)$ denotes the parallel composition of two processes P and Q that interact through the fresh channel x . Finally, the term $P \oplus Q$ models a non-deterministic choice between two behaviors P and Q .

π LIN binders are easily recognizable because they enclose channel names in round parentheses. Note that all outputs are in fact *bound outputs*. The output of free channels can be modeled by combining bound outputs with links [30]. For example, the output $\bar{x}(y, z)$ of a pair of free channels y and z can be modeled as the term $\bar{x}(y', z')(y \leftrightarrow y' \parallel z \leftrightarrow z')$. We identify processes modulo renaming of bound names, we write $\text{fn}(P)$ for the set of channel names occurring free in P and we write $\{y/x\}$ for the capture-avoiding substitution of y for the free occurrences of x . We impose a well-formedness condition on processes so that, in every sub-term of the form $\bar{x}(y, z)(P \parallel Q)$, we have $y \notin \text{fn}(Q)$ and $z \notin \text{fn}(P)$.

We omit any concrete syntax for representing infinite processes. Instead, we work directly with infinite trees obtained by corecursively unfolding contractive equations of the form $A(x_1, \dots, x_n) = P$. For each such equation, we assume that $\text{fn}(P) \subseteq \{x_1, \dots, x_n\}$ and we write $A(y_1, \dots, y_n)$ for its unfolding $P\{y_i/x_i\}_{1 \leq i \leq n}$. The technical report [10] describes the changes for supporting a more conventional (but slightly heavier) handling of infinite processes.

► **Notation 1.** To reduce clutter due to the systematic use of bound outputs, by convention we omit the continuation called y in Table 1 when its name is chosen to coincide with that of the channel x on which y is sent/received. For example, with this notation we have $x(z).P = x(z, x).P$ and $\text{in}_i \bar{x}.P = \text{in}_i \bar{x}(x).P$ and $\text{case } x\{P, Q\} = \text{case } x(x)\{P, Q\}$. ◻

A welcome side effect of adopting Notation 1 is that it gives the illusion of working with a session calculus in which the same channel x may be used repeatedly for multiple input/output operations, while in fact x is a linear channel used for exchanging a single message along with a fresh continuation that turns out to have the same name. If one takes this notation as native syntax for a session calculus, its linear π -calculus encoding [11] turns out to be precisely the π LIN term it denotes. Besides, the idea of rebinding the same name over and over is widespread in session-based functional languages [20, 34] as it provides a simple way of “updating the type” of a session endpoint after each use.

► **Example 2.** Below we model the interaction informally described in Section 1 between buyer and seller using the syntactic sugar defined in Notation 1:

$$(x)(\text{Buyer}\langle x \rangle \parallel \text{Seller}\langle x, y \rangle) \quad \text{where} \quad \begin{aligned} \text{Buyer}(x) &= \text{rec } \bar{x}.(\text{add } \bar{x}.\text{Buyer}\langle x \rangle \oplus \text{pay } \bar{x}.\bar{x}()) \\ \text{Seller}(x, y) &= \text{corec } x.\text{case } x\{\text{Seller}\langle x, y \rangle, x().\bar{y}()\} \end{aligned}$$

At each round of the interaction, the buyer decides whether to **add** an item to the shopping cart and repeat the same behavior (left branch of the choice) or to **pay** the seller and terminate (right branch of the choice). The seller reacts dually and signals its termination by sending a unit on the channel y . As we will see in Section 4, **rec** \bar{x} and **corec** x identify the points within processes where (co)recursive types are unfolded.

If we were to define *Buyer* using distinct bound names we would write an equation like

$$\text{Buyer}(x) = \text{rec } \bar{x}(y).(\text{add } \bar{y}(z).\text{Buyer}\langle z \rangle \oplus \text{pay } \bar{y}(z).\bar{z}())$$

and similarly for *Seller*. ◻

The operational semantics of the calculus is given in terms of the *structural precongruence* relation \preceq and the *reduction relation* \rightarrow defined in Table 2. As usual, structural precongruence relates processes that are syntactically different but semantically equivalent. In particular, [S-LINK] states that linking x with y is the same as linking y with x , whereas [S-COMM] and [S-ASSOC] state the expected commutativity and associativity laws for parallel composition. Concerning the latter, the side condition $x \in \text{fn}(Q)$ makes sure that Q (the process brought

■ **Table 2** Structural pre-congruence and reduction semantics of πLIN .

[S-LINK]	$x \leftrightarrow y \preceq y \leftrightarrow x$	
[S-COMM]	$(x)(P \parallel Q) \preceq (x)(Q \parallel P)$	
[S-ASSOC]	$(x)(P \parallel (y)(Q \parallel R)) \preceq (y)((x)(P \parallel Q) \parallel R)$	if $x \in \text{fn}(Q)$ and $y \notin \text{fn}(P)$
[R-LINK]	$(x)(x \leftrightarrow y \parallel P) \rightarrow P\{y/x\}$	
[R-UNIT]	$(x)(\bar{x}() \parallel x().P) \rightarrow P$	
[R-PAIR]	$(x)(\bar{x}(z, y)(P_1 \parallel P_2) \parallel x(z, y).Q) \rightarrow (z)(P_1 \parallel (y)(P_2 \parallel Q))$	
[R-SUM]	$(x)(\text{in}_i \bar{x}(y).P \parallel \text{case } x(y)\{P_1, P_2\}) \rightarrow (y)(P \parallel P_i)$	$i \in \{1, 2\}$
[R-REC]	$(x)(\text{rec } \bar{x}(y).P \parallel \text{corec } x(y).Q) \rightarrow (y)(P \parallel Q)$	
[R-CHOICE]	$P_1 \oplus P_2 \rightarrow P_i$	$i \in \{1, 2\}$
[R-CUT]	$(x)(P \parallel R) \rightarrow (x)(Q \parallel R)$	if $P \rightarrow Q$
[R-STRUCT]	$P \rightarrow Q$	if $P \preceq R \rightarrow Q$

closer to P when the relation is read from left to right) is indeed connected with P by means of the channel x . Note that [S-ASSOC] only states the right-to-left associativity of parallel composition and that the left-to-right associativity law $(x)((y)(P \parallel Q) \parallel R) \preceq (y)(P \parallel (x)(Q \parallel R))$ is derivable when $x \in \text{fn}(Q)$. The reduction relation is mostly unremarkable. Links are reduced with [R-LINK] by effectively merging the linked channels. All the reductions that involve the interaction between processes except [R-UNIT] create new continuation channels that connect the reducts. The rule [R-CHOICE] models the non-deterministic choice between two behaviors. Finally, [R-CUT] and [R-STRUCT] close reductions by cuts and structural pre-congruence. In the following we write \Rightarrow for the reflexive, transitive closure of \rightarrow and we say that P is *stuck* if there is no Q such that $P \rightarrow Q$.

We conclude this section by formalizing fair termination. To this aim, we introduce the notion of *run* as a maximal execution of a process. Hereafter ω stands for the lowest transfinite ordinal number and o ranges over the elements of $\omega + 1$.

► **Definition 3 (run).** A run of P is a sequence $(P_i)_{i \in o}$ where $o \in \omega + 1$ and $P_0 = P$ and $P_i \rightarrow P_{i+1}$ for all $i + 1 \in o$ and either the sequence is infinite or it ends with a stuck process.

Hereafter we use ρ to range over runs. Using runs we can define a range of termination properties for processes. In particular, P is *terminating* if all of its runs are finite and P is *weakly terminating* if it has a finite run. Fair termination is a termination property somewhat in between termination and weak termination in which only the “fair” runs of a process are taken into account insofar its termination is concerned. There exist several notions of fair run corresponding to different fairness assumptions [17, 29, 39]. The notion of fair run used here is an instance of the *fair reachability of predicates* by Queille and Sifakis [36].

► **Definition 4 (fair termination).** A run is fair if it contains finitely many weakly terminating processes. We say that P is fairly terminating if every fair run of P is finite.

To better understand our notion of fair run, it may be useful to think of the cases in which a run is *not* fair. An *unfair* run is necessarily infinite and describes the execution of a process that always has the chance to terminate but systematically avoids doing so. Think of the system modeled in Example 2: the (only) run in which the buyer adds items to the shopping cart forever and never pays the seller is unfair; any other run of the system is fair and finite. So, the system in Example 2 is not terminating (it admits an infinite execution) but it is fairly terminating (all the fair executions are finite).

The following result characterizes fair termination without using fair runs. Most importantly, it provides us with the key proof principle for the soundness of our type system.

► **Theorem 5** (proof principle for fair termination). *P is fairly terminating if and only if $P \Rightarrow Q$ implies that Q is weakly terminating.*

Theorem 5 says that any reasonable type system that ensures weak process termination also ensures fair process termination. By “reasonable” we mean a type system for which type preservation (also known as *subject reduction*) holds, which is usually the case. Indeed, if we consider a well-typed process P such that $P \Rightarrow Q$, we can deduce that Q is also well typed. Now, the soundness of the type system guarantees that Q is weakly terminating. By applying Theorem 5 from right to left, we conclude that every well-typed P is fairly terminating.

3 Formulas and Types

The types of π LIN are built using the multiplicative additive fragment of linear logic enriched with least and greatest fixed points. In this section we specify the syntax of types along with all the auxiliary notions that are needed to present the type system and prove its soundness.

The syntax of pre-formulas relies on an infinite set of *propositional variables* ranged over by X and Y and is defined by the grammar below:

Pre-formula $\varphi, \psi ::= \mathbf{0} \mid \top \mid \mathbf{1} \mid \perp \mid \varphi \oplus \psi \mid \varphi \& \psi \mid \varphi \otimes \psi \mid \varphi \wp \psi \mid \mu X.\varphi \mid \nu X.\varphi \mid X$

As usual, μ and ν are the binders of propositional variables and the notions of free and bound variables are defined accordingly. We assume that the body of fixed points extends as much as possible to the right of a pre-formula, so $\mu X.X \oplus \mathbf{1}$ means $\mu X.(X \oplus \mathbf{1})$ and not $(\mu X.X) \oplus \mathbf{1}$. We write $\{\varphi/X\}$ for the capture-avoiding substitution of all free occurrences of X with φ . We write φ^\perp for the *dual* of φ , which is the involution defined by the equations

$$\begin{aligned} \mathbf{0}^\perp &= \top & (\varphi \oplus \psi)^\perp &= \varphi^\perp \& \psi^\perp & (\mu X.\varphi)^\perp &= \nu X.\varphi^\perp \\ \mathbf{1}^\perp &= \perp & (\varphi \otimes \psi)^\perp &= \varphi^\perp \wp \psi^\perp & X^\perp &= X \end{aligned}$$

A *formula* is a closed pre-formula. In the context of π LIN, formulas describe how linear channels are used. Positive formulas (those built with the constants $\mathbf{0}$ and $\mathbf{1}$, the connectives \oplus and \otimes and the least fixed point) indicate output operations whereas negative formulas (the remaining forms) indicate input operations. The formulas $\varphi \oplus \psi$ and $\varphi \& \psi$ describe a linear channel used for sending/receiving a tagged channel of type φ or ψ . The tag (either in_1 or in_2) distinguishes between the two possibilities. The formulas $\varphi \otimes \psi$ and $\varphi \wp \psi$ describe a linear channel used for sending/receiving a pair of channels of type φ and ψ ; $\mu X.\varphi$ and $\nu X.\varphi$ describe a linear channel used for sending/receiving a channel of type $\varphi\{\mu X.\varphi/X\}$ or $\varphi\{\nu X.\varphi/X\}$ respectively. The constants $\mathbf{1}$ and \perp describe a linear channel used for sending/receiving the unit. Finally, the constants $\mathbf{0}$ and \top respectively describe channels on which nothing can be sent and from which nothing can be received.

► **Example 6.** Looking at the structure of *Buyer* and *Seller* in Example 2, we can make an educated guess on the type of the channel x they use. Concerning x , we see that it is used according to $\varphi \stackrel{\text{def}}{=} \mu X.X \oplus \mathbf{1}$ in *Buyer* and according to $\psi \stackrel{\text{def}}{=} \nu X.X \& \perp$ in *Seller*. Note that $\varphi = \psi^\perp$, suggesting that *Buyer* and *Seller* may interact correctly when connected. ◻

We write \preceq for the *subformula ordering*, that is the least partial order such that $\varphi \preceq \psi$ if φ is a subformula of ψ . For example, consider $\varphi \stackrel{\text{def}}{=} \mu X.\nu Y.X \oplus Y$ and $\psi \stackrel{\text{def}}{=} \nu Y.\varphi \oplus Y$. Then we have $\varphi \preceq \psi$ and $\psi \not\preceq \varphi$. When Φ is a set of formulas, we write $\min \Phi$ for its \preceq -minimum formula if it is defined. Occasionally we let \star stand for an arbitrary binary connective \oplus , \otimes , $\&$, or \wp and σ stand for an arbitrary fixed point operator μ or ν .

When two πLIN processes interact on some channel x , they may exchange other channels on which their interaction continues. We can think of these subsequent interactions stemming from a shared channel x as being part of the same conversation (the literature on *sessions* [22, 25] builds on this idea [27, 11]). The soundness proof of the type system is heavily based on the proof of the cut elimination property of μMALL^∞ , which relies on the ability to uniquely identify the types of the channels that belong to the same conversation and to trace conversations within typing derivations. Following the literature on μMALL^∞ [3, 16, 2], we annotate formulas with addresses. We assume an infinite set \mathcal{A} of *atomic addresses*, \mathcal{A}^\perp being the set of their duals such that $\mathcal{A} \cap \mathcal{A}^\perp = \emptyset$ and $\mathcal{A}^{\perp\perp} = \mathcal{A}$. We use a and b to range over elements of $\mathcal{A} \cup \mathcal{A}^\perp$. An *address* is a string aw where $w \in \{i, l, r\}^*$. The dual of an address is defined as $(aw)^\perp = a^\perp w$. We use α and β to range over addresses, we write \sqsubseteq for the prefix relation on addresses and we say that α and β are *disjoint* if $\alpha \not\sqsubseteq \beta$ and $\beta \not\sqsubseteq \alpha$.

A *type* is a formula φ paired with an address α written φ_α . We use S and T to range over types and we extend to types several operations defined on formulas: we use logical connectives to compose types so that $\varphi_{\alpha l} \star \psi_{\alpha r} \stackrel{\text{def}}{=} (\varphi \star \psi)_\alpha$ and $\sigma X.\varphi_{\alpha i} \stackrel{\text{def}}{=} (\sigma X.\varphi)_\alpha$; the dual of a type is obtained by dualizing both its formula and its address, that is $(\varphi_\alpha)^\perp \stackrel{\text{def}}{=} \varphi_{\alpha^\perp}^\perp$; type substitution preserves the address in the type within which the substitution occurs, but forgets the address of the type being substituted, that is $\varphi_\alpha\{\psi_\beta/X\} \stackrel{\text{def}}{=} \varphi\{\psi/X\}_\alpha$.

We often omit the address of constants (which represent terminated conversations) and we write \bar{S} for the formula obtained by forgetting the address of S . Finally, we write \rightsquigarrow for the least reflexive relation on types such that $S_1 \star S_2 \rightsquigarrow S_i$ and $\sigma X.S \rightsquigarrow S\{\sigma X.S/X\}$.

► **Example 7.** Consider once again the formula $\varphi \stackrel{\text{def}}{=} \mu X.X \oplus \mathbf{1}$ that describes the behavior of *Buyer* (Example 6) and let a be an arbitrary atomic address. We have

$$\varphi_a \rightsquigarrow (\varphi \oplus \mathbf{1})_{ai} \rightsquigarrow \varphi_{ail} \rightsquigarrow (\varphi \oplus \mathbf{1})_{aili} \rightsquigarrow \mathbf{1}_{ailir}$$

where the fact that the types in this sequence all share a common non-empty prefix “ a ” indicates that they belong to the same conversation. Note how the symbols i , l and r composing an address indicate the step taken in the syntax tree of types for making a move in this sequence: i means “inside”, when a fixed point operator is unfolded, whereas l and r mean “left” and “right”, when the corresponding branch of a connective is selected. \square

4 Type System

We now present the typing rules for πLIN . As usual we introduce typing contexts to track the type of the names occurring free in a process. A *typing context* is a finite map from names to types written $x_1 : S_1, \dots, x_n : S_n$. We use Γ and Δ to range over contexts, we write $\text{dom}(\Gamma)$ for the domain of Γ and Γ, Δ for the union of Γ and Δ when $\text{dom}(\Gamma) \cap \text{dom}(\Delta) = \emptyset$. Typing judgments have the form $P \vdash \Gamma$. We say that P is *quasi typed* in Γ if the judgment $P \vdash \Gamma$ is coinductively derivable using the rules shown in Table 3 and described below. For the time being we say “quasi typed” and not “well typed” because some infinite derivations using the rules in Table 3 are invalid. Well-typed processes are quasi-typed processes whose typing derivation satisfies some additional validity conditions that we detail in Section 5.

Rule [AX] states that a link $x \leftrightarrow y$ is quasi typed provided that x and y have dual types, but not necessarily dual addresses. Rule [CUT] states that a process composition $(x)(P \parallel Q)$ is quasi typed provided that P and Q use the linear channel x in complementary ways, one according to some type S and the other according to the dual type S^\perp . Note that the context Γ, Δ in the conclusion of the rule is defined provided that Γ and Δ have disjoint domains. This condition entails that P and Q do not share any channel other than x ensuring that the

■ **Table 3** Typing rules for π LIN.

$$\begin{array}{c}
\frac{}{x \leftrightarrow y \vdash x : \varphi_\alpha, y : \varphi_\beta^\perp} \text{[AX]} \qquad \frac{P \vdash \Gamma, x : S \quad Q \vdash \Delta, x : S^\perp}{(x)(P \parallel Q) \vdash \Gamma, \Delta} \text{[CUT]} \\
\\
\frac{}{\text{case } x\{\} \vdash \Gamma, x : \top} \text{[\top]} \qquad \frac{P \vdash \Gamma}{x().P \vdash \Gamma, x : \perp} \text{[\perp]} \qquad \frac{}{\bar{x}() \vdash x : \mathbf{1}} \text{[1]} \\
\\
\frac{P \vdash \Gamma, y : S, z : T}{x(y, z).P \vdash \Gamma, x : S \wp T} \text{[\wp]} \qquad \frac{P \vdash \Gamma, y : S \quad Q \vdash \Delta, z : T}{\bar{x}(y, z)(P \parallel Q) \vdash \Gamma, \Delta, x : S \otimes T} \text{[\otimes]} \\
\\
\frac{P \vdash \Gamma, y : S \quad Q \vdash \Gamma, y : T}{\text{case } x(y)\{P, Q\} \vdash \Gamma, x : S \& T} \text{[\&]} \qquad \frac{P \vdash \Gamma, y : S_i}{\text{in}_i \bar{x}(y).P \vdash \Gamma, x : S_1 \oplus S_2} \text{[\oplus]} \\
\\
\frac{P \vdash \Gamma, y : S\{\nu X.S/X\}}{\text{corec } x(y).P \vdash \Gamma, x : \nu X.S} \text{[\nu]} \qquad \frac{P \vdash \Gamma, y : S\{\mu X.S/X\}}{\text{rec } \bar{x}(y).P \vdash \Gamma, x : \mu X.S} \text{[\mu]} \qquad \frac{P \vdash \Gamma \quad Q \vdash \Gamma}{P \oplus Q \vdash \Gamma} \text{[CHOICE]}
\end{array}$$

interaction between P and Q may proceed without deadlocks. Rule $[\top]$ deals with a process that receives an empty message from channel x . Since this cannot happen, we allow the process to be quasi typed in any context. Rules $[1]$ and $[\perp]$ concern the exchange of units. The former rule states that $\bar{x}()$ is quasi typed in a context that contains a single association for the x channel with type $\mathbf{1}$, whereas the latter rule removes x from the context (hence from the set of usable channels), requiring the continuation process to be quasi typed in the remaining context. Rules $[\otimes]$ and $[\wp]$ concern the exchange of pairs. The former rule requires the two forked processes P and Q to be quasi typed in the respective contexts enriched with associations for the continuation channels y and z being created. The latter rule requires the continuation process to be quasi typed in a context enriched with the channels extracted from the received pair. Rules $[\oplus]$ and $[\&]$ deal with the exchange of disjoint sums in the expected way. Rules $[\mu]$ and $[\nu]$ deal with fixed point operators by unfolding the (co)recursive type of the channel x . As in μ MALL $^\infty$, the two rules have exactly the same structure despite the fact that the two fixed point operators being used are dual to each other. Clearly, the behavior of least and greatest fixed points must be distinguished by some other means, as we will see in Section 5 when discussing the validity of a typing derivation. Finally, $[\text{CHOICE}]$ deals with non-deterministic choices by requiring that each branch of a choice must be quasi typed in exactly the same typing context as the conclusion.

Besides the structural constraints imposed by the typing rules, we implicitly require that the types in the range of all typing contexts have pairwise disjoint addresses. This condition ensures that it is possible to uniquely trace a communication protocol in a typing derivation: if we have two channels x and y associated with two types φ_α and ψ_β such that $\alpha \sqsubseteq \beta$, then we know that y is a continuation resulting from a communication that started from x . In a sense, x and y represent different moments in the same conversation.

► **Example 8 (buyer and seller).** Let us show that the system described in Example 2 is quasi typed. To this aim, let $\varphi \stackrel{\text{def}}{=} \mu X.X \oplus \mathbf{1}$ and $\psi \stackrel{\text{def}}{=} \nu X.X \& \perp$ respectively be the formulas describing the behavior of *Buyer* and *Seller* on the channel x . Note that $\psi = \varphi^\perp$ and let a be an arbitrary atomic address. We derive

$$\frac{\frac{\vdots}{Buyer\langle x \rangle \vdash x : \varphi_{ai}} \text{ [}\oplus\text{]} \quad \frac{\frac{\overline{\bar{x}() \vdash x : \mathbf{1}} \text{ [}\mathbf{1}\text{]}}{\text{pay } \bar{x}.\bar{x}() \vdash x : (\varphi \oplus \mathbf{1})_{ai}} \text{ [}\oplus\text{]}}{\text{add } \bar{x}.\text{Buyer}\langle x \rangle \oplus \text{pay } \bar{x}.\bar{x}() \vdash x : (\varphi \oplus \mathbf{1})_{ai}} \text{ [CHOICE]}}{\text{Buyer}\langle x \rangle \vdash x : \varphi_a} \text{ [}\mu\text{]}$$

and also

$$\frac{\frac{\vdots}{Seller\langle x, y \rangle \vdash x : \psi_{a^\perp i}, y : \mathbf{1}} \text{ [}\perp\text{]} \quad \frac{\frac{\overline{\bar{y}() \vdash y : \mathbf{1}} \text{ [}\mathbf{1}\text{]}}{x().\bar{y}() \vdash x : \perp, y : \mathbf{1}} \text{ [}\perp\text{]}}{\text{case } x\{Seller\langle x, y \rangle, x().\bar{y}()\} \vdash x : (\psi \& \perp)_{a^\perp i}, y : \mathbf{1}} \text{ [}\&\text{]}}{\text{Seller}\langle x, y \rangle \vdash x : \psi_{a^\perp}, y : \mathbf{1}} \text{ [}\nu\text{]}$$

showing that *Buyer* and *Seller* are quasi typed. Note that both derivations are infinite, but for dual reasons. In *Buyer* the infinite branch corresponds to the behavior in which *Buyer* chooses to add one more item to the shopping cart. This choice is made independently of the behavior of other processes in the system. In *Seller*, the infinite branch corresponds to the behavior in which *Seller* receives one more **add** message from *Buyer*. By combining these derivations we obtain

$$\frac{\frac{\vdots}{Buyer\langle x \rangle \vdash x : \varphi_a} \quad \frac{\vdots}{Seller\langle x, y \rangle \vdash x : \psi_{a^\perp}, y : \mathbf{1}}}{(x)(Buyer\langle x \rangle \parallel Seller\langle x, y \rangle) \vdash y : \mathbf{1}} \text{ [CUT]}$$

showing that the system as a whole is quasi typed. \lrcorner

As we have anticipated, there exist infinite typing derivations that are unsound from a logical standpoint, because they allow us to prove $\mathbf{0}$ or the empty sequent. For example, if we consider the non-terminating process $\Omega(x) = \Omega\langle x \rangle \oplus \Omega\langle x \rangle$ we obtain the infinite derivation

$$\frac{\frac{\vdots}{\Omega\langle x \rangle \vdash x : \mathbf{0}} \quad \frac{\vdots}{\Omega\langle x \rangle \vdash x : \mathbf{0}}}{\Omega\langle x \rangle \vdash x : \mathbf{0}} \text{ [CHOICE]} \quad (1)$$

showing that $\Omega\langle x \rangle$ is quasi typed. As illustrated by the next example, there exist non-terminating processes that are quasi typed also in logically sound contexts.

► **Example 9** (compulsive buyer). Consider the following variant of the *Buyer* process

$$Buyer(x, z) = \text{rec } \bar{x}.\text{add } \bar{x}.\text{Buyer}\langle x, z \rangle$$

that models a “compulsive buyer”, namely a buyer that adds infinitely many items to the shopping cart but never pays. Using $\varphi \stackrel{\text{def}}{=} \mu X.X \oplus \mathbf{1}$ and an arbitrary atomic address a we can build the following infinite derivation

$$\frac{\frac{\vdots}{Buyer\langle x \rangle \vdash x : \varphi_{ai}} \text{ [}\oplus\text{]}}{\text{add } \bar{x}.\text{Buyer}\langle x \rangle \vdash x : (\varphi \oplus \mathbf{1})_{ai}} \text{ [}\mu\text{]}}{\text{Buyer}\langle x \rangle \vdash x : \varphi_a}$$

showing that this process is quasi typed. By combining this derivation with the one for *Seller* in Example 8 we obtain a derivation establishing that $(x)(Buyer\langle x \rangle \parallel Seller\langle x, y \rangle)$ is quasi typed in the context $y : \mathbf{1}$, although this composition cannot terminate. \dashv

5 From Quasi-Typed to Well-Typed Processes

To rule out unsound derivations like those in Equation (1) and Example 9 it is necessary to impose a validity condition on derivations [3, 16]. Roughly speaking, μMALL^∞ 's validity condition requires every infinite branch of a derivation to be supported by the continuous unfolding of a greatest fixed point. In order to formalize this condition, we start by defining *threads*, which are sequences of types describing sequential interactions at the type level.

► **Definition 10 (thread).** A thread of S is a sequence of types $(S_i)_{i \in o}$ for some $o \in \omega + 1$ such that $S_0 = S$ and $S_i \rightsquigarrow S_{i+1}$ whenever $i + 1 \in o$.

Hereafter we use t to range over threads. For example, if we consider $\varphi \stackrel{\text{def}}{=} \mu X.X \oplus \mathbf{1}$ from Example 2 we have that $t \stackrel{\text{def}}{=} (\varphi_a, (\varphi \oplus \mathbf{1})_{ai}, \varphi_{ail}, \dots)$ is an infinite thread of φ_a . A thread is *stationary* if it has an infinite suffix of equal types. The above thread t is not stationary.

Among all threads, we are interested in finding those in which a ν -formula is unfolded infinitely often. These threads, called ν -threads, are precisely defined thus:

► **Definition 11 (ν -thread).** Let $t = (S_i)_{i \in \omega}$ be an infinite thread, let \bar{t} be the corresponding sequence $(\bar{S}_i)_{i \in \omega}$ of formulas and let $\text{inf}(t)$ be the set of elements of \bar{t} that occur infinitely often in \bar{t} . We say that t is a ν -thread if $\min \text{inf}(t)$ is defined and is a ν -formula.

If we consider the infinite thread t above, we have $\text{inf}(t) = \{\varphi, \varphi \oplus \mathbf{1}\}$ and $\min \text{inf}(t) = \varphi$, so t is not a ν -thread because φ is not a ν -formula. Consider instead $\varphi \stackrel{\text{def}}{=} \nu X.\mu Y.X \oplus Y$ and $\psi \stackrel{\text{def}}{=} \mu Y.\varphi \oplus Y$ and observe that ψ is the “unfolding” of φ . Now $t_1 \stackrel{\text{def}}{=} (\varphi_a, \psi_{ai}, (\varphi \oplus \psi)_{aai}, \varphi_{aail}, \dots)$ is a thread of φ_a such that $\text{inf}(t_1) = \{\varphi, \psi, \varphi \oplus \psi\}$ and we have $\min \text{inf}(t_1) = \varphi$ because $\varphi \preceq \psi$, so t_1 is a ν -thread. If, on the other hand, we consider the thread $t_2 \stackrel{\text{def}}{=} (\varphi_a, \psi_{ai}, (\varphi \oplus \psi)_{aai}, \psi_{aair}, (\varphi \oplus \psi)_{aairi}, \dots)$ such that $\text{inf}(t_2) = \{\psi, \varphi \oplus \psi\}$ we have $\min \text{inf}(t_2) = \psi$ because $\psi \preceq \varphi \oplus \psi$, so t_2 is not a ν -thread. Intuitively, the \preceq -minimum formula among those that occur infinitely often in a thread is the outermost fixed point operator that is being unfolded infinitely often. It is possible to show that this minimum formula is always well defined [16]. If such minimum formula is a greatest fixed point operator, then the thread is a ν -thread.

Now we proceed by identifying threads along branches of typing derivations. To this aim, we provide a precise definition of *branch*.

► **Definition 12 (branch).** A branch of a typing derivation is a sequence $(P_i \vdash \Gamma_i)_{i \in o}$ of judgments for some $o \in \omega + 1$ such that $P_0 \vdash \Gamma_0$ occurs somewhere in the derivation and $P_{i+1} \vdash \Gamma_{i+1}$ is a premise of the rule application that derives $P_i \vdash \Gamma_i$ whenever $i + 1 \in o$.

An infinite branch is valid if supported by a ν -thread that originates somewhere therein.

► **Definition 13 (valid branch).** Let $\gamma = (P_i \vdash \Gamma_i)_{i \in \omega}$ be an infinite branch in a derivation. We say that γ is valid if there exists $j \in \omega$ such that $(S_k)_{k \geq j}$ is a non-stationary ν -thread and S_k is in the range of Γ_k for every $k \geq j$.

For example, the infinite branch in the typing derivation for *Seller* of Example 2 is valid since it is supported by the ν -thread $(\psi_{a^+}, (\psi \& \perp)_{a^+i}, \psi_{a^+il}, \dots)$ where $\psi \stackrel{\text{def}}{=} \nu X.X \& \perp$ happens to be the \preceq -minimum formula that is unfolded infinitely often. On the other hand, the infinite branch in the typing derivation for *Buyer* of Example 9 is invalid, because the only infinite thread in it is $(\varphi_a, (\varphi \oplus \mathbf{1})_{ai}, \varphi_{ail}, \dots)$ which is not a ν -thread.

A μMALL^∞ derivation is valid if so is every infinite branch in it [3, 16]. For the purpose of ensuring fair termination, this condition is too strong because some infinite branches in a typing derivation may correspond to unfair executions that, by definition, we neglect insofar its termination is concerned. For example, the infinite branch in the derivation for *Buyer* of Example 8 corresponds to an unfair run in which the buyer insists on adding items to the shopping cart, despite it periodically has a chance of paying the seller and terminate the interaction. That typing derivation for *Buyer* would be considered an invalid proof in μMALL^∞ because the infinite branch is not supported by a ν -thread (in fact, there is a μ -formula that is unfolded infinitely many times along that branch, as in Example 9).

It is generally difficult to understand if a branch corresponds to a fair or unfair run because the branch describes the evolution of an incomplete process whose behavior is affected by the interactions it has with processes found in other branches of the derivation. However, we can detect (some) unfair branches by looking at the non-deterministic choices they traverse, since choices are made autonomously by processes. To this aim, we introduce the notion of *rank* to estimate the least number of choices a process can possibly make during its lifetime.

► **Definition 14 (rank).** *Let r and s range over the elements of $\mathbb{N}^\infty \stackrel{\text{def}}{=} \mathbb{N} \cup \{\infty\}$ equipped with the expected total order \leq and operation $+$ such that $r + \infty = \infty + r = \infty$. The rank of a process P , written $|P|$, is the least element of \mathbb{N}^∞ such that*

$$\begin{array}{lll} |x \leftrightarrow y| = 0 & |x(y, z).P| = |P| & |\text{case } x(y)\{P, Q\}| = \max\{|P|, |Q|\} \\ |\text{case } x\{| & |\text{in}_i \bar{x}(y).P| = |P| & |P \oplus Q| = 1 + \min\{|P|, |Q|\} \\ |\bar{x}()| = 0 & |\text{rec } \bar{x}(y).P| = |P| & |(x)(P \parallel Q)| = |P| + |Q| \\ |x().P| = |P| & |\text{corec } x(y).P| = |P| & |\bar{x}(y, z)(P \parallel Q)| = |P| + |Q| \end{array}$$

Roughly, the rank of terminated processes is 0, that of processes with a single continuation P coincides with the rank of P , and that of processes spawning two continuations P and Q is the sum of the ranks of P and Q . Then, the rank of a sum input with continuations P and Q is conservatively estimated as the maximum of the ranks of P and Q , since we do not know which one will be taken, whereas the rank of a choice with continuations P and Q is 1 plus the minimum of the ranks of P and Q . If we take *Buyer* and *Seller* from Example 2 we have $|\text{Buyer}\langle x \rangle| = 1$ and $|\text{Seller}\langle x, y \rangle| = 0$. We also have $|\Omega\langle x \rangle| = \infty$. Note that $|P|$ only depends on the structure of P but not on the actual names occurring in P , so it is well and uniquely defined as the least solution of a finite system of equations [10].

► **Definition 15.** *A branch is fair if it traverses finitely many, finitely-ranked choices.*

A finitely-ranked choice is at finite distance from a region of the process in which there are no more choices. An *unfair* branch gets close to such region infinitely often, but systematically avoids entering it. Note that every finite branch is also fair, but there are fair branches that are infinite. For instance, all the infinite branches of the derivation in Equation (1) and the only infinite branch in the derivation for *Seller* $\langle x, y \rangle$ of Example 8 are fair since they do not traverse any finitely-ranked choice. On the contrary, the only infinite branch in the derivation for *Buyer* $\langle x \rangle$ of the Example 8 is unfair since it traverses infinitely many finitely-ranked choices. All fair branches in the same derivation for *Buyer* are finite.

At last we can define our notion of well-typed process.

► **Definition 16 (well-typed process).** *We say that P is well typed in Γ , written $P \Vdash \Gamma$, if the judgment $P \vdash \Gamma$ is derivable and each fair, infinite branch in its derivation is valid.*

Note that Ω is ill typed since the fair, infinite branches in Equation (1) are all invalid. We can now formulate the key properties of well-typed processes, starting from subject reduction.

36:12 Fair Termination in the Linear π -Calculus

► **Theorem 17** (subject reduction). *If $P \Vdash \Gamma$ and $P \rightarrow Q$ then $Q \Vdash \Gamma$.*

All reductions in Table 2 except those for non-deterministic choices correspond to cut-elimination steps in a quasi typing derivation. As an illustration, below is a fragment of derivation tree for two processes exchanging a pair of y and z on channel x .

$$\frac{\frac{\frac{\vdots}{P \vdash \Gamma, y : S} \quad \frac{\vdots}{Q \vdash \Delta, z : T}}{\bar{x}(y, z)(P \parallel Q) \vdash \Gamma, \Delta, x : S \otimes T} [\otimes] \quad \frac{\frac{\vdots}{R \vdash \Gamma', y : S^\perp, z : T^\perp}}{x(y, z).R \vdash \Gamma', x : S^\perp \wp T^\perp} [\wp]}{\frac{\bar{x}(y, z)(P \parallel Q) \parallel x(y, z).R \vdash \Gamma, \Delta, \Gamma'}{(x)(\bar{x}(y, z)(P \parallel Q) \parallel x(y, z).R) \vdash \Gamma, \Delta, \Gamma'} [\text{CUT}]}$$

As the process reduces, the quasi typing derivation is rearranged so that the cut on x is replaced by two cuts on y and z . The resulting quasi typing derivation is shown below.

$$\frac{\frac{\frac{\vdots}{P \vdash \Gamma, y : S} \quad \frac{\frac{\vdots}{Q \vdash \Delta, z : T} \quad \frac{\vdots}{R \vdash \Gamma', y : S^\perp, z : T^\perp}}{(z)(Q \parallel R) \vdash \Delta, \Gamma', y : S^\perp} [\text{CUT}]}{(y)(P \parallel (z)(Q \parallel R)) \vdash \Gamma, \Delta, \Gamma'} [\text{CUT}]}$$

It is also interesting to observe that, when $P \rightarrow Q$, the reduct Q is well typed in the same context as P but its rank may be different. In particular, the rank of Q can be *greater* than the rank of P . Recalling that the rank of a process estimates the number of choices that the process must perform to terminate, the fact that the rank of Q increases means that Q *moves away* from termination instead of getting closer to it (we will see an instance where this phenomenon occurs in Example 23). What really matters is that a well-typed process is weakly terminating. This is the second key property ensured by our type system.

► **Lemma 18** (weak termination). *If $P \Vdash x : \mathbf{1}$ then $P \Rightarrow \bar{x}()$.*

The proof of Lemma 18 is a refinement of the cut elimination property of μMALL^∞ . Essentially, the only new case we have to handle is when a choice $P_1 \oplus P_2$ “emerges” towards the bottom of the typing derivation, meaning that it is no longer guarded by any action. In this case, we reduce the choice to the P_i with smaller rank, which is guaranteed to lay on a fair branch of the derivation. An auxiliary result used in the proof of Lemma 18 is that our type system is a conservative extension of μMALL^∞ .

► **Lemma 19.** *If $P \Vdash x_1 : S_1, \dots, x_n : S_n$ then $\vdash S_1, \dots, S_n$ is derivable in μMALL^∞ .*

The property that well-typed processes can always successfully terminate is a simple consequence of Theorem 17 and Lemma 18.

► **Theorem 20** (soundness). *If $P \Vdash x : \mathbf{1}$ and $P \Rightarrow Q$ then $Q \Rightarrow \bar{x}()$.*

Theorem 20 entails all the good properties we expect from well-typed processes: *failure freedom* (no unguarded sub-process `case $y\{\}$` ever appears), *deadlock freedom* (if the process stops it is terminated), *lock freedom* [26, 32] (every pending action can be completed in finite time) and *junk freedom* (every channel can be depleted). The combination of Theorems 5 and 20 also guarantees the termination of every fair run of the process.

► **Corollary 21** (fair termination). *If $P \Vdash x : \mathbf{1}$ then P is fairly terminating.*

Observe that zero-ranked process do not contain any non-deterministic choice. In that case, every infinite branch in their typing derivation is fair and our validity condition coincides with that of μMALL^∞ . As a consequence, we obtain the following strengthening of Corollary 21:

► **Proposition 22.** *If $P \Vdash x : \mathbf{1}$ and $|P| = 0$ then P is terminating.*

For regular processes (those consisting of finitely many distinct sub-trees, up to renaming of bound names) it is possible to easily adapt the algorithm that decides the validity of a μMALL^∞ proof so that it decides the validity of a πLIN typing derivation. The algorithm is sketched in more detail in the technical report [10, Appendix D].

► **Example 23** (parallel programming). In this example we see a πLIN modeling of a *parallel programming pattern* whereby a *Work* process creates an unbounded number of workers each one dedicated to an independent task and a *Gather* process collects and combines the partial results from the workers. The processes *Work* and *Gather* are defined as follows:

$$\begin{aligned} \text{Work}(x) &= \text{rec } \bar{x}.(\text{complex } \bar{x}.\bar{x}(y)(\bar{y}() \parallel \text{Work}\langle x \rangle) \oplus \text{simple } \bar{x}.\bar{x}()) \\ \text{Gather}(x, z) &= \text{corec } x.\text{case } x\{x(y).y().\text{Gather}\langle x, z \rangle, x().\bar{z}()\} \end{aligned}$$

At each iteration, the *Work* process non-deterministically decides whether the task is **complex** (left hand side of the choice) or **simple** (right hand side of the choice). In the first case, it bifurcates into a new worker, which in the example simply sends a unit on y , and another instance of itself. In the second case it terminates. The *Gather* process joins the results from all the workers before signalling its own termination by sending a unit on z . Note that the number of actions *Gather* has to perform before terminating is unbounded, as it depends on the non-deterministic choices made by *Work*.

Below is a typing derivation for *Work* where $\varphi \stackrel{\text{def}}{=} \mu X.(\mathbf{1} \otimes X) \oplus \mathbf{1}$ and a is an arbitrary atomic address:

$$\frac{\frac{\frac{\bar{y}() \vdash y : \mathbf{1}}{[1]} \quad \frac{\vdots}{\text{Work}\langle x \rangle \vdash x : \varphi_{\text{air}}}{} \quad [\otimes]}{\bar{x}(y)(\bar{y}() \parallel \text{Work}\langle x \rangle) \vdash x : (\mathbf{1} \otimes \varphi)_{\text{ail}}} [\oplus]}{\text{complex } \bar{x} \dots \vdash x : ((\mathbf{1} \otimes \varphi) \oplus \mathbf{1})_{\text{ai}}} [\oplus] \quad \frac{}{\text{simple } \bar{x}.\bar{x}() \vdash x : ((\mathbf{1} \otimes \varphi) \oplus \mathbf{1})_{\text{ai}}} [1], [\oplus]}{\text{complex } \bar{x}.\bar{x}(y)(\bar{y}() \parallel \text{Work}\langle x \rangle) \oplus \text{simple } \bar{x}.\bar{x}() \vdash x : ((\mathbf{1} \otimes \varphi) \oplus \mathbf{1})_{\text{ai}}} [\text{CHOICE}]}{\text{Work}\langle x \rangle \vdash x : \varphi_a} [\mu]$$

Note that the only infinite branch in this derivation is unfair because it traverses infinitely many choices with rank $1 = |\text{Work}\langle x \rangle|$. So, *Work* is well typed.

Concerning *Gather*, we obtain the following typing derivation where $\psi \stackrel{\text{def}}{=} \nu X.(\perp \wp X) \& \perp$:

$$\frac{\frac{\frac{\vdots}{\text{Gather}\langle x, z \rangle \vdash x : \psi_{a^+ilr}, z : \mathbf{1}}{[1]} \quad \frac{}{y().\text{Gather}\langle x, z \rangle \vdash x : \psi_{a^+ilr}, y : \perp, z : \mathbf{1}}{[\perp]} \quad [\wp]}{x(y).y().\text{Gather}\langle x, z \rangle \vdash x : (\perp \wp \psi)_{a^+il}, z : \mathbf{1}} [\&]}{\text{case } x\{x(y).y().\text{Gather}\langle x, z \rangle, x().\bar{z}()\} \vdash x : ((\perp \wp \psi) \& \perp)_{a^+i}, z : \mathbf{1}} [\&]}{\text{Gather}\langle x, z \rangle \vdash x : \psi_{a^+}, z : \mathbf{1}} [\nu]$$

Here too there is just one infinite branch, which is fair and supported by the ν -thread $t = (\psi_{a^+}, ((\perp \wp \psi) \& \perp)_{a^+i}, (\perp \wp \psi)_{a^+il}, \psi_{a^+ilr}, \dots)$. Indeed, all the formulas in \bar{t} occur infinitely often and $\min \inf(t) = \psi$ which is a ν -formula. Hence, *Gather* is well typed and so is the composition $(x)(\text{Work}\langle x \rangle \parallel \text{Gather}\langle x, z \rangle)$ in the context $z : \mathbf{1}$. We conclude that the program is fairly terminating, despite the fact that the composition of *Work* and *Gather* may grow arbitrarily large because *Work* may spawn an unbounded number of workers. \square

► **Example 24** (forwarder). In this example we illustrate a deterministic, well-typed process that unfolds a least fixed point infinitely many times. In particular, we consider once again the formulas $\varphi \stackrel{\text{def}}{=} \mu X.X \oplus \mathbf{1}$ and $\psi \stackrel{\text{def}}{=} \nu X.X \& \perp$ and the process Fwd defined by the equation

$$Fwd(x, y) = \text{corec } x.\text{rec } \bar{y}.\text{case } x\{\text{in}_1 \bar{y}.Fwd(x, y), \text{in}_2 \bar{y}.x().\bar{y}()\}$$

which forwards the sequence of messages received from channel x to channel y . We derive

$$\frac{\frac{\frac{\vdots}{Fwd(x, y) \vdash x : \psi_{ail}, y : \varphi_{bil}}{\text{in}_1 \bar{y}.Fwd(x, y) \vdash x : \psi_{ail}, y : (\varphi \oplus \mathbf{1})_{bi}} [\oplus]}{\text{case } x\{\text{in}_1 \bar{y}.Fwd(x, y), \text{in}_2 \bar{y}.x().\bar{y}()\} \vdash x : (\psi \& \perp)_{ai}, y : (\varphi \oplus \mathbf{1})_{bi}} [\&]}{\frac{\frac{\frac{\vdots}{x().\bar{y}() \vdash x : \perp, y : \mathbf{1}}{\text{in}_2 \bar{y}.x().\bar{y}() \vdash x : \perp, y : (\varphi \oplus \mathbf{1})_{bi}} [\oplus]}{\text{case } x\{\text{in}_1 \bar{y}.Fwd(x, y), \text{in}_2 \bar{y}.x().\bar{y}()\} \vdash x : (\psi \& \perp)_{ai}, y : (\varphi \oplus \mathbf{1})_{bi}} [\&]}{Fwd(x, y) \vdash x : \psi_a, y : \varphi_b} [\nu], [\mu]}$$

and observe that $|Fwd(x, y)| = 0$. This typing derivation is valid because the only infinite branch is fair and supported by the ν -thread of ψ_a . Note that $\varphi = \psi^\perp$ and that the derivation proves an instance of [AX]. In general, the axiom is admissible in μMALL^∞ [3]. \lrcorner

► **Example 25** (slot machine). Rank finiteness is not a necessary condition for well typedness. As an example, consider the system $(x)(\text{Player}(x) \parallel \text{Machine}(x, y))$ where

$$\begin{aligned} \text{Player}(x) &= \text{rec } \bar{x}.\text{play } \bar{x}.\text{case } x\{\text{Player}(x), \text{rec } \bar{x}.\text{quit } \bar{x}.\bar{x}()\} \oplus \text{quit } \bar{x}.\bar{x}() \\ \text{Machine}(x, y) &= \text{corec } x.\text{case } x\{\text{win } \bar{x}.\text{Machine}(x, y) \oplus \text{lose } \bar{x}.\text{Machine}(x, y), x().\bar{y}()\} \end{aligned}$$

which models a game between a player and a slot machine. At each round, the player decides whether to **play** or to **quit**. In the first case, the slot machine answers with either **win** or **lose**. If the player **wins**, it also **quits**. Otherwise, it repeats the same behavior. It is possible to show that $\text{Player}(x) \vdash x : \varphi_a$ and $\text{Machine}(x, y) \vdash x : \psi_{a^\perp}, y : \mathbf{1}$ are derivable where $\varphi \stackrel{\text{def}}{=} \mu X.(X \& X) \oplus \mathbf{1}$ and $\psi \stackrel{\text{def}}{=} \nu X.(X \oplus X) \& \perp$ [10]. The only infinite branch in the derivation for Player is unfair since $|\text{Player}(x)| = 1$, so Player is well typed. There are infinitely many branches in the derivation for Machine accounting for all the sequences of **win** and **lose** choices that can be made. Since $|\text{Machine}(x, y)| = \infty$, all these branches are fair but also valid. So, the system as a whole is well typed. \lrcorner

6 Related Work

On account of the known encodings of sessions into the linear π -calculus [27, 11, 38], πLIN belongs to the family of process calculi providing logical foundations to sessions and session types. Some representatives of this family are πDILL [6] and its variant equipped with a circular proof theory [14, 13], CP [40] and μCP [30], among others. There are two main aspects distinguishing πLIN from these calculi. The first one is that these calculi take sessions as a native feature. This fact can be appreciated both at the level of processes, where session endpoints are *linearized* resources that can be used *multiple times* albeit in a sequential way, and also at the level of types, where the interpretation of the $\varphi \otimes \psi$ and $\varphi \wp \psi$ formulas is skewed so as to distinguish the type φ of the message being sent/received on a channel from the type ψ of the channel after the exchange has taken place. In contrast, πLIN adopts a more fundamental communication model based on *linear* channels, and is thus closer to the spirit of the encoding of linear logic proofs into the π -calculus proposed by Bellin and Scott [4] while retaining the same expressiveness of the aforementioned calculi. To some extent, πLIN is also more general than those calculi, since a formula $\varphi \otimes \psi$ may be

interpreted as a protocol that bifurcates into two independent sub-protocols φ and ψ (we have seen an instance of this pattern in Example 23). So, πLIN is natively equipped with the capability of modeling some multiparty interactions, in addition to all of the binary ones. A session-based communication model identical to πLIN , but whose type system is based on intuitionistic rather than classical linear logic, has been presented by DeYoung et al. [15]. In that work, the authors advocate the use of explicit continuations with the purpose of modeling an asynchronous communication semantics and they prove the equivalence between such model and a buffered session semantics. However, they do not draw a connection between the proposed calculus and the linear π -calculus [28] through the encoding of binary sessions [27, 11] and, in the type system, the multiplicative connectives are still interpreted asymmetrically. The second aspect that distinguishes πLIN from the other calculi is its type system, which is still deeply rooted into linear logic and yet it ensures fair termination instead of progress [6, 40, 35], termination [30] or strong progress [14, 13]. Fair termination entails progress, strong progress and lock freedom [26, 32], but at the same time it does not always rule out processes admitting infinite executions. Simply, infinite executions are deemed unrealistic because they are unfair.

Another difference between πLIN and other calculi based on linear logic is that its operational semantics is completely ordinary, in the sense that it does not include commuting conversions, reductions under prefixes, or the swapping of communication prefixes. The cut elimination result of μMALL^∞ , on which the proof of Theorem 20 is based, works by reducing cuts from the bottom of the derivation instead of from the top [3, 16, 2]. As a consequence, it is not necessary to reduce cuts guarded by prefixes or to push cuts deep into the derivation tree to enable key reductions in πLIN processes.

Some previous works [9, 7] have studied type systems ensuring the fair termination of binary and multiparty sessions. Although the enforced property is the same and the communication models of these works are closely related to the one we consider here, the used techniques are quite different and the families of well-typed processes induced by the two type systems are not comparable. One aspect that is shared among all of these works, including the present one, is the use of a *rank* annotation or function that estimates how far a process is from termination. In previous works for session-based communications [9, 7], ranks account for the number of sessions that processes create and the number of times they use subtyping before they terminate. Since ranks are required to be finite, this means that deterministic processes can only create a finite number of sessions and can only use subtyping a finite number of times. As a consequence, the forwarder in Example 24 is ill typed according to those typing disciplines because it applies subtyping (in an implicit way, whenever it sends a tag) an unbounded number of times. At the same time, a “compulsive” variant of the player in Example 25 that keeps playing until it wins is well typed in previous type systems [9, 7] but ill typed in the present one. This difference stems, at least in part, from the fact that the previous type systems support processes defined using general recursion, whereas πLIN ’s type system relies on the duality between recursion and corecursion. A deeper understanding of these differences requires further investigation.

The extension of calculi based on linear logic with non-deterministic features has recently received quite a lot of attention. Rocha and Caires [37] have proposed a session calculus with shared cells and non-deterministic choices that can model mutable state. Their typing rule for non-deterministic choices is the same as our own, but in their calculus choices do not reduce. Rather, they keep track of every possible evolution of a process to be able to prove a confluence result. Qian et al. [35] introduce *coexponentials*, a new pair of modalities that enable the modeling of concurrent clients that compete in order to interact with a shared

server that processes requests sequentially. In this setting, non-determinism arises from the unpredictable order in which different clients are served. Interestingly, the coexponentials are derived by resorting to their semantics in terms of least and greatest fixed points. For this reason, the cut elimination result of μMALL^∞ might be useful to attack the termination proof in their setting.

7 Concluding Remarks

We have studied a conservative extension of μMALL^∞ [3, 16, 2], an infinitary proof system of multiplicative additive linear logic with fixed points, that serves as type system for πLIN , a linear π -calculus with (co)recursive types. Well-typed processes fairly terminate.

One drawback of the proposed type system is that establishing whether a quasi-typed process is also well typed requires a global check on the whole typing derivation. The need for a global check seems to arise commonly in infinitary proof systems [12, 16, 3, 2], so an obvious aspect to investigate is whether the analysis can be localized. A possible source of inspiration for devising a local type system for πLIN might come from the work of Derakhshan and Pfenning [14]. They propose a compositional technique for dealing with infinitary typing derivations in a session calculus, although their type system is limited to the additive fragment of linear logic.

Fair subtyping [31, 33] is a refinement of the standard subtyping relation for session types [19] that preserves fair termination and that plays a key role in our previous type system ensuring fair termination for binary and multiparty sessions [9, 7]. Given the rich literature exploring the connections between linear logic and session types, πLIN and its type system might provide the right framework for investigating the logical foundations of fair subtyping.

References

- 1 Samson Abramsky. Proofs as processes. *Theor. Comput. Sci.*, 135(1):5–9, 1994. doi:10.1016/0304-3975(94)00103-0.
- 2 David Baelde, Amina Doumane, Denis Kuperberg, and Alexis Saurin. Bouncing threads for circular and non-wellfounded proofs. In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS, 2022*, 2022. arXiv:2005.08257.
- 3 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, August 29 – September 1, 2016, Marseille, France*, volume 62 of *LIPICs*, pages 42:1–42:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICs.CSL.2016.42.
- 4 Gianluigi Bellin and Philip J. Scott. On the pi-calculus and linear logic. *Theor. Comput. Sci.*, 135(1):11–65, 1994. doi:10.1016/0304-3975(94)00104-9.
- 5 Luís Caires and Jorge A. Pérez. Multiparty session types within a canonical binary theory, and beyond. In Elvira Albert and Ivan Lanese, editors, *Formal Techniques for Distributed Objects, Components, and Systems – 36th IFIP WG 6.1 International Conference, FORTE 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings*, volume 9688 of *Lecture Notes in Computer Science*, pages 74–95. Springer, 2016. doi:10.1007/978-3-319-39570-8_6.
- 6 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.
- 7 Luca Ciccone, Francesco Dagnino, and Luca Padovani. Fair termination of multiparty sessions. In Karim Ali and Jan Vitek, editors, *36th European Conference on Object-Oriented Programming, ECOOP 2022, June 6-10, 2022, Berlin, Germany*, volume 222 of *LIPICs*, pages 26:1–26:26. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ECOOP.2022.26.

- 8 Luca Ciccone and Luca Padovani. A dependently typed linear π -calculus in agda. In *PPDP '20: 22nd International Symposium on Principles and Practice of Declarative Programming, Bologna, Italy, 9-10 September, 2020*, pages 8:1–8:14. ACM, 2020. doi:10.1145/3414080.3414109.
- 9 Luca Ciccone and Luca Padovani. Fair termination of binary sessions. *Proc. ACM Program. Lang.*, 6(POPL):1–30, 2022. doi:10.1145/3498666.
- 10 Luca Ciccone and Luca Padovani. An infinitary proof theory of linear logic ensuring fair termination in the linear π -calculus. Technical report, Università di Torino, 2022. URL: <http://www.di.unito.it/~padovani/>.
- 11 Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. *Inf. Comput.*, 256:253–286, 2017. doi:10.1016/j.ic.2017.06.002.
- 12 Christian Dax, Martin Hofmann, and Martin Lange. A proof system for the linear time μ -calculus. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, volume 4337 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2006. doi:10.1007/11944836_26.
- 13 Farzaneh Derakhshan. *Session-Typed Recursive Processes and Circular Proofs*. PhD thesis, Carnegie Mellon University, 2021. URL: https://www.andrew.cmu.edu/user/fderakhs/publications/Dissertation_Farzaneh.pdf.
- 14 Farzaneh Derakhshan and Frank Pfenning. Circular proofs as session-typed processes: A local validity condition. *CoRR*, abs/1908.01909, 2019. arXiv:1908.01909.
- 15 Henry DeYoung, Luís Caires, Frank Pfenning, and Bernardo Toninho. Cut reduction in linear logic as asynchronous session-typed communication. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) – 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, volume 16 of *LIPICs*, pages 228–242. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPICs.CSL.2012.228.
- 16 Amina Doumane. *On the infinitary proof theory of logics with fixed points. (Théorie de la démonstration infinitaire pour les logiques à points fixes)*. PhD thesis, Paris Diderot University, France, 2017. URL: <https://tel.archives-ouvertes.fr/tel-01676953>.
- 17 Nissim Francez. *Fairness*. Texts and Monographs in Computer Science. Springer, 1986. doi:10.1007/978-1-4612-4886-6.
- 18 Philippa Gardner, Cosimo Laneve, and Lucian Wischik. Linear forwarders. *Inf. Comput.*, 205(10):1526–1550, 2007. doi:10.1016/j.ic.2007.01.006.
- 19 Simon J. 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.
- 20 Simon J. Gay and Vasco Thudichum Vasconcelos. Linear type theory for asynchronous session types. *J. Funct. Program.*, 20(1):19–50, 2010. doi:10.1017/S0956796809990268.
- 21 Orna Grumberg, Nissim Francez, and Shmuel Katz. Fair termination of communicating processes. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '84, pages 254–265, New York, NY, USA, 1984. Association for Computing Machinery. doi:10.1145/800222.806752.
- 22 Kohei Honda. Types for dyadic interaction. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993. doi:10.1007/3-540-57208-2_35.
- 23 Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In Chris Hankin, editor, *Programming Languages and Systems – ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 – April 4, 1998, Proceedings*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998. doi:10.1007/BFb0053567.

- 24 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *J. ACM*, 63(1):9:1–9:67, 2016. doi:10.1145/2827695.
- 25 Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniérou, Dimitris Mostros, 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.
- 26 Naoki Kobayashi. A type system for lock-free processes. *Inf. Comput.*, 177(2):122–159, 2002. doi:10.1006/inco.2002.3171.
- 27 Naoki Kobayashi. Type systems for concurrent programs. In *10th Anniversary Colloquium of UNU/IIST*, LNCS 2757, pages 439–453. Springer, 2002. Extended version at <http://www.kb.ecei.tohoku.ac.jp/~koba/papers/tutorial-type-extended.pdf>. doi:10.1007/978-3-540-40007-3_26.
- 28 Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. *ACM Trans. Program. Lang. Syst.*, 21(5):914–947, 1999. doi:10.1145/330249.330251.
- 29 M.Z. Kwiatkowska. Survey of fairness notions. *Information and Software Technology*, 31(7):371–386, 1989. doi:10.1016/0950-5849(89)90159-6.
- 30 Sam Lindley and J. Garrett Morris. Talking bananas: structural recursion for session types. In Jacques Garrigue, Gabriele Keller, and Eijiro Sumii, editors, *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*, pages 434–447. ACM, 2016. doi:10.1145/2951913.2951921.
- 31 Luca Padovani. Fair subtyping for open session types. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *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 373–384. Springer, 2013. doi:10.1007/978-3-642-39212-2_34.
- 32 Luca Padovani. Deadlock and lock freedom in the linear π -calculus. 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 '14, Vienna, Austria, July 14–18, 2014*, pages 72:1–72:10. ACM, 2014. doi:10.1145/2603088.2603116.
- 33 Luca Padovani. Fair subtyping for multi-party session types. *Math. Struct. Comput. Sci.*, 26(3):424–464, 2016. doi:10.1017/S096012951400022X.
- 34 Luca Padovani. A simple library implementation of binary sessions. *J. Funct. Program.*, 27:e4, 2017. doi:10.1017/S0956796816000289.
- 35 Zesen Qian, G. A. Kavvos, and Lars Birkedal. Client-server sessions in linear logic. *Proc. ACM Program. Lang.*, 5(ICFP):1–31, 2021. doi:10.1145/3473567.
- 36 Jean-Pierre Queille and Joseph Sifakis. Fairness and related properties in transition systems – A temporal logic to deal with fairness. *Acta Informatica*, 19:195–220, 1983. doi:10.1007/BF00265555.
- 37 Pedro Rocha and Luís Caires. Propositions-as-types and shared state. *Proc. ACM Program. Lang.*, 5(ICFP):1–30, 2021. doi:10.1145/3473584.
- 38 Alceste Scalas, Ornela Dardha, Raymond Hu, and Nobuko Yoshida. A linear decomposition of multiparty sessions for safe distributed programming (artifact). *Dagstuhl Artifacts Ser.*, 3(2):03:1–03:2, 2017. doi:10.4230/DARTS.3.2.3.
- 39 Rob van Glabbeek and Peter Höfner. Progress, justness, and fairness. *ACM Comput. Surv.*, 52(4):69:1–69:38, 2019. doi:10.1145/3329125.
- 40 Philip Wadler. Propositions as sessions. *J. Funct. Program.*, 24(2-3):384–418, 2014. doi:10.1017/S095679681400001X.

On Session Typing, Probabilistic Polynomial Time, and Cryptographic Experiments

Ugo Dal Lago  

University of Bologna, Italy
INRIA Sophia Antipolis, France

Giulia Giusti  

University of Bologna, Italy

Abstract

A system of session types is introduced as induced by a Curry Howard correspondence applied to bounded linear logic, suitably extended with probabilistic choice operators and ground types. The resulting system satisfies some expected properties, like subject reduction and progress, but also unexpected ones, like a polynomial bound on the time needed to reduce processes. This makes the system suitable for modelling experiments and proofs from the so-called computational model of cryptography.

2012 ACM Subject Classification Theory of computation → Process calculi; Theory of computation → Program semantics; Security and privacy → Mathematical foundations of cryptography

Keywords and phrases Session Types, Probabilistic Computation, Bounded Linear Logic, Cryptographic Experiments

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2022.37

Related Version *Full Version*: <https://arxiv.org/abs/2207.03360> [20]

Funding The authors are partially supported by the ERC CoG “DIAPASoN”, GA 818616.

Acknowledgements The authors would like to thank the anonymous referees for the many insightful comments.

1 Introduction

Session types [30, 23, 31] are a typing discipline capable of regulating the interaction between the parallel components in a concurrent system in such a way as to *prevent* phenomena such as deadlock or livelock, at the same time *enabling* the parties to interact following the rules of common communication protocols. In the twenty-five years since their introduction, session types have been shown to be a flexible tool, being adaptable to heterogeneous linguistic and application scenarios (see, e.g., [45, 10, 32, 16]). A particularly fruitful line of investigation concerns the links between session-type disciplines and Girard’s linear logic [26]. This intimate relationship, known since the introduction of session types, found a precise formulation in the work of Caires and Pfenning on a Curry-Howard correspondence between session types and intuitionistic linear logic [11], which has been developed along many directions [47, 48, 44, 18]. In Caires and Pfenning’s type system, proofs of intuitionistic linear logic become type derivations for terms of Milner’s π -calculus. Noticeably, typable processes satisfy properties (e.g. progress and deadlock freedom) which do not hold for untyped processes.

Process algebras, and in particular algebras in the style of the π -calculus, have been used, among other things, as specification formalisms for cryptographic protocols in the so-called *symbolic* (also known as *formal*) model of cryptography, i.e. in the model, due to Dolev and Yao [25], in which aspects related to computational complexity and probability theory, themselves central to the *computational* model, are abstracted away: strings become symbolic



© Ugo Dal Lago and Giulia Giusti;

licensed under Creative Commons License CC-BY 4.0

33rd International Conference on Concurrency Theory (CONCUR 2022).

Editors: Bartek Klin, Slawomir Lasota, and Anca Muscholl; Article No. 37; pp. 37:1–37:18

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

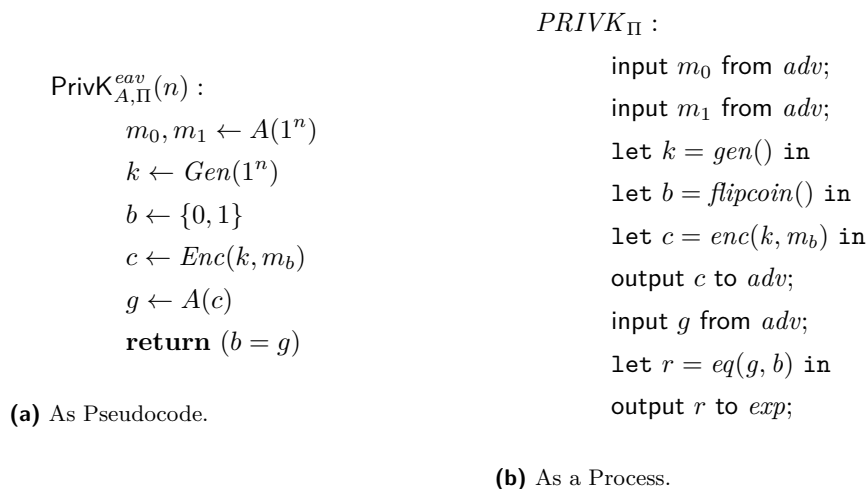
expressions, adversaries are taken as having arbitrary computing power, and nondeterminism replaces probabilism in regulating the interaction between the involved parties. This includes π -calculus dialects akin to the applied π -calculus [2], or the spi-calculus [3].

Is it possible to model cryptographic protocols by way of process algebras in the so-called computational model *itself*? A widely explored path in this direction consists in the so-called *computational soundness* results for symbolic models, which have been successfully proved in the realm of process algebras [1, 17, 46, 5]. In computationally sound symbolic models, any computational attack can be simulated by a symbolic attack, this way proving that whenever a protocol is secure in the latter, it must be secure in the former, too. If one is interested in calculi precisely and fully capturing the computational model, computational soundness is not enough, i.e., one wants a model capturing all *and only* the computational adversaries. And indeed, there have been some attempts to define process algebras able to faithfully capture the computational model by way of operators for probabilistic choice and constraints on computational complexity [42]. The literature, however, is much sparser than for process algebras in symbolic style. We believe that this is above all due to the fact that the contemporary presence of probabilistic evolution and the intrinsic nondeterminism of process algebras leads to complex formal systems which are hard to reason about.

This paper shows that session typing can be exploited for the sake of designing a simple formal system in which, indeed, complexity constraints and probabilistic choices can be both taken into account, this way allowing for the modelling of cryptographic experiments. At the level of types, we build on the approach by Caires and Pfenning, refining it through the lenses of bounded linear logic, a logical system which captures polynomial time complexity in the sequential setting [27, 29], at the same time allowing for a high degree of intensional expressivity [21]. At the level of processes, we enrich proof terms with first-order function symbols computing probabilistic polytime functions, namely the basic building blocks of any cryptographic protocol. This has two consequences: process evolution becomes genuinely probabilistic, while process terms and types are enriched so as to allow for the exchange of strings, this way turning the calculus to an applied one. From a purely definitional perspective, then, the introduced calculus, called π **DIBLL**, is relatively simple, and does not significantly deviate from the literature, being obtained by mixing well-known ingredients in a novel way. The calculus π **DIBLL** is introduced in Section 3 below.

Despite its simplicity π **DIBLL** is on the one hand capable of expressing some simple cryptographic experiments, and on the other hand satisfies some strong meta-theoretical properties. This includes type soundness, which is expected, and can be spelled out as *subject reduction* and *progress*, but also a polynomial bound on the length of reduction sequences, a form of *reachability* property which is essential for our calculus to be considered a model of cryptographic adversaries. All this is described in Section 4.

As interesting as they are, these properties are not by themselves sufficient for considering π **DIBLL** a proper calculus for computational cryptography. What is missing, in fact, is a way to capture computational indistinguishability, in the sense of the computational model [35, 28]. Actually, this is where the introduced calculus shows its peculiarities with respect to similar calculi from the literature, and in particular with respect to the CCS-style calculus by Mitchell and Scedrov [42]. Indeed, π **DIBLL** typable processes enjoy a confluence property which cannot hold for untyped processes. The latter, in turn, implies that firing internal actions on any typable process results in a *unique* distribution of processes, all of them ready to produce an observable action. This makes relational reasoning handier. We in particular explore observational equivalence in Section 5, then showing how this can be of help in a simple experiment-based security proof in Section 6.



■ **Figure 1** The Indistinguishability Experiment.

Due to space constraints, many details had to be elided, but can be found in the long version of this paper [20], which is available online.

2 A Bird's Eye View on Cryptographic Experiments and Sessions

In this section, we introduce the reader to cryptographic experiments, and we show how they and the parties involved can be conveniently modelled as session-typed processes. We will also hint at how relational reasoning could be useful in supporting proofs of security. We will do all this by way of an example, namely the one of private key encryption schemes and security against passive adversaries. We will try to stay self-contained, and the interested reader can check either this paper's extended version [20] or textbooks [35] for more details or for the necessary cryptographic preliminaries.

► **Definition 1** (Negligible Functions). *A function f from the natural numbers to the non-negative real numbers is negligible iff for every positive polynomial p there is an $N \in \mathbb{N}$ such that for all natural numbers $n > N$ it holds that $f(n) < 1/p(n)$.*

► **Definition 2** (Probabilistic Polynomial Time Algorithms). *An algorithm A is called probabilistic polynomial time (PPT in the following) iff it is randomized, and there exists a polynomial p which is an upper limit to the computational complexity of A regardless of the probabilistic choices made by the latter.*

Since the running time of any cryptographic algorithm has to be polynomially bounded w.r.t. the *value* of the security parameter n , the latter is passed in unary (i.e. as 1^n) to the algorithm, so that n is also a lower bound to the *length* of the input.

A private-key encryption scheme is a triple of algorithms $\Pi = (Gen, Enc, Dec)$, the first one responsible for key generation, the latter two being the encryption and decryption algorithms, respectively. When could we say that such a scheme Π is *secure*? Among the many equivalent definitions in the literature, one of the handiest is the one based on indistinguishability, which is based on the experiment PrivK^{adv} reported in Figure 1a exactly in the form it has in [35]. As the reader may easily notice, the experiment is nothing more than a randomized algorithm interacting with both the adversary A and the scheme Π . The interaction between PrivK^{adv} and the adversary A can be put in evidence by switching

to a language for processes, see Figure 1b. The process $PRIVK_{\Pi}$ communicates with the adversary through the channel adv and outputs the result of its execution to the channel exp . Apart from the fact that the adversary has been factored out, the process is syntactically very similar to the experiment PrivK^{adv} . Actually, we could have made the interaction between $PRIVK$ and Π explicit by turning the latter into a process interacting with the former through a dedicated channel.

The interaction between an adversary ADV and $PRIVK_{\Pi}$ can be modelled through the parallel composition operator, i.e., by studying the behaviour of $ADV \mid PRIVK_{\Pi}$. As we will soon see, we would like the aforementioned parallel composition to output true on the channel exp with probability very close to $\frac{1}{2}$, and this is indeed what cryptography actually prescribes [35]. We should not, however, be too quick to proclaim the problem solved. What, for example, if ADV communicates with $PRIVK_{\Pi}$ in a way different from the one prescribed by the experiment, e.g. by *not* passing two strings to it, thus blocking the interaction? Even worse, what if $PRIVK_{\Pi}$ becomes the parallel composition $PRIVK \mid \Pi$ and ADV cheats on the communication by intercepting the messages exchanged between Π and $PRIVK$? These scenarios are of course very interesting from a security viewpoint, but we are not interested at those here: the only thing ADV is allowed to do is to send the two messages and to use its internal computational capabilities to guess the value b the experiment produces.

How to enforce all this at the level of processes? Actually, this is what session types are good for! It would be nice, for example, to be able to type the two processes above as follows:

$$\begin{aligned} adv : \mathbb{S}[p] \otimes \mathbb{S}[p] \otimes (\mathbb{S}[p] \multimap \mathbb{B}) \vdash PRIVK_{\Pi} :: exp : \mathbb{B} \\ \vdash ADV :: adv : \mathbb{S}[p] \otimes \mathbb{S}[p] \otimes (\mathbb{S}[p] \multimap \mathbb{B}) \end{aligned}$$

where \mathbb{B} is the type of booleans and $\mathbb{S}[p]$ is the type of strings of length p . Moreover, we would like to somehow force a restriction νadv to be placed next to the parallel composition $ADV \mid PRIVK_{\Pi}$, so as to prescribe that ADV can only communicate with the experiment, and not with the outside world. Finally, we would like ADV to range over processes working in polynomial time. All this is indeed taken care by our session type discipline as introduced in Section 3.

But now, would it be possible to not only *express* simple cryptographic situations, but also to *prove* some security properties about them from within the realm of processes? As already mentioned, this amounts to requiring that for every efficient adversary (i.e. for every PPT algorithm) A it holds that $\Pr[\text{PrivK}_{A,\Pi}^{adv}(n)] \leq \frac{1}{2} + \varepsilon(n)$, where ε is negligible. In the world of processes, this becomes the following equation

$$\nu adv.(PRIVK_{\Pi} \mid ADV) \sim FAIRFLIP_{exp} \quad (1)$$

where $FAIRFLIP_{exp}$ behaves like a fair coin outputting its value on the channel exp , and \sim expresses approximate equivalence as induced by negligible functions. Making all this formal is nontrivial for at least three reasons:

- First of all, the statement only holds for *efficient* adversaries. The relation \sim , however specified, must then take this constraint into account.
- Secondly, the relation \sim only holds in an approximate sense, and the acceptable degree of approximation crucially depends on n , the so-called security parameter. This is due to negligibility, without which cryptography would be essentially vacuous.
- Finally, the computational security of Π can at the time of writing be proved only based on *assumptions*, e.g. that one-way functions or pseudorandom generators exist. In other words, Equation (1) only holds in a conditional sense, and cryptographic proofs have to be structured accordingly.

The calculus πDIBLL successfully addresses all these challenges, as we are going to show in the rest of this paper.

3 Processes and Session Typing

This section is devoted to introducing π **DIBLL**, a variation on π **DILL** [11] in which a polynomial constraint on the replicated processes is enforced following the principles of bounded linear logic [27]. For the sake of properly representing cryptographic protocols in the computational model, π **DIBLL** is also equipped with indexed ground types and a notion of probabilistic choice.

3.1 Preliminaries

Preliminary to the definition of the π **DIBLL** session type system are three concepts, namely polynomials, probability distributions and indexed ground types. Let us start this section introducing polynomials.

► **Definition 3** (Polynomials). Polynomial variables are indicated with metavariables like n and m , and form a set \mathcal{PV} . Polynomial expressions are built from natural number constants, polynomial variables, addition and multiplication. A polynomial p depending on the polynomial variables $\bar{n} = n_1, \dots, n_k$ is sometime indicated as $p(n_1, \dots, n_k)$ and abbreviated as $p(\bar{n})$. Such a polynomial is said to be a \mathcal{V} -polynomial whenever all variables in the sequence \bar{n} are in $\mathcal{V} \subseteq \mathcal{PV}$. If $\mathcal{V} \subseteq \mathcal{PV}$, any map $\rho : \mathcal{V} \rightarrow \mathbb{N}$ is said to be a \mathcal{V} -substitution, and the natural number obtained by interpreting any variable $m \in \mathcal{V}$ occurring in a \mathcal{V} -polynomial p with $\rho(m)$ is indicated just as $p(\rho)$. If \mathcal{V} is a singleton $\{n\}$, the substitution mapping n to the natural number i is indicated as ρ_i , and \mathcal{V} is indicated, abusing notation, with n .

Distributions play a crucial role in probability theory and represent the likelihood of observing an element from a given set. In this paper, they will be the key ingredient in giving semantics to types and processes.

► **Definition 4** (Probability Distributions). A probability distribution on the finite set A is a function $\mathcal{D} : A \rightarrow \mathbb{R}_{[0,1]}$ such that: $\sum_{v \in A} \mathcal{D}(v) = 1$. A probability distribution is often indicated by way of the notation $\{v_1^{r_1}, \dots, v_m^{r_m}\}$ (where v_1, \dots, v_m are distinct elements of A), which stands for the distribution \mathcal{D} such that $r_i = \mathcal{D}(v_i)$ for every $i \in \{1, \dots, m\}$. Given a probability distribution \mathcal{D} on A , its support $S(\mathcal{D}) \subseteq A$ contains precisely those elements of A to which \mathcal{D} attributes a strictly positive probability. The set of all probability distributions on a set A is indicated as $\mathcal{D}(A)$.

In the computational model, the agents involved exchange binary strings. We keep the set of ground types slightly more general, so as to treat booleans as a separate type. As a crucial step towards dealing with polytime constraints, the type of strings is indexed by a polynomial, which captures the length of binary strings inhabiting the type.

► **Definition 5** (Ground Types). Ground types are expressions generated by the grammar $B ::= \mathbb{B} \mid \mathbb{S}[p]$, where p is a polynomial expression. A \mathcal{V} -ground type is a ground type B such that all polynomial variables occurring in it are taken from \mathcal{V} , and as such can be given a semantics in the context of a \mathcal{V} -substitution ρ : $\llbracket \mathbb{B} \rrbracket_\rho = \{0, 1\}$, $\llbracket \mathbb{S}[p] \rrbracket_\rho = \{0, 1\}^{p(\rho)}$.

The precise nature of any ground type $\mathbb{S}[p(\bar{n})]$ is only known when the polynomial variables in \bar{n} , which stand for so-called security parameters, are attributed a natural number value. For reasons of generality, we actually allow more than one security parameter, even if cryptographic constructions almost invariably need only one of them.

3.2 Terms

Terms are expressions which are *internally* evaluated by processes, the result of this evaluation having a ground type and being exchanged between the different (sub)processes.

Function Symbols. We work with a set \mathcal{F} of *function symbols*, ranged over by metavariables like f and g . In the context of this paper, it is important that function symbols can be evaluated in probabilistic polynomial time, and this can be achieved by taking function symbols from a language guaranteeing the aforementioned complexity bounds [40, 22]. Each function symbol $f \in \mathcal{F}$ comes equipped with:

- A type $\text{typeof}(f)$ having the form $B_1, \dots, B_m \rightarrow C$ where the B_i and C are $\{n\}$ -ground types.
- A family $\llbracket f \rrbracket = \{\llbracket f \rrbracket_i\}_{i \in \mathbb{N}}$ of functions giving semantics to f such that $\llbracket f \rrbracket_i$ goes from $\llbracket B_1 \rrbracket_{\rho_i} \times \dots \times \llbracket B_m \rrbracket_{\rho_i}$ to $\mathcal{D}(\llbracket C \rrbracket_{\rho_i})$, where $\text{typeof}(f)$ is $B_1, \dots, B_m \rightarrow C$.
- We assume each function symbol f to be associated with an $\{n\}$ -polynomial $\text{comof}(f)$ bounding the complexity of computing f , in the following sense: there must be a PPT algorithm $\text{algot}(f)$ which, on input 1^i and a tuple t in $\llbracket B_1 \rrbracket_{\rho_i} \times \dots \times \llbracket B_m \rrbracket_{\rho_i}$ returns in time at most $\text{comof}(f)(\rho_i)$ each value $x \in \llbracket C \rrbracket_{\rho_i}$ with probability $\llbracket f \rrbracket_i(t)(x)$, where $\text{typeof}(f) = B_1, \dots, B_m \rightarrow C$.

Term Syntax and Semantics. Finally, we are able to define terms and values, which are expressions derivable in the following grammars:

$$a, b, c ::= v \mid f_p(v_1, \dots, v_n) \qquad v, w ::= z \mid \text{true} \mid \text{false} \mid s.$$

Here f is a function symbol, p is a polynomial, **true** and **false** are the usual boolean constants, s is any binary string, and z is a *term variable* taken from a set \mathcal{TV} disjoint from \mathcal{PV} . Terms are assumed to be well-typed according to an elementary type system that we elide for the sake of simplicity (see [20] for more details). Reduction rules between terms and distributions of values are given only for terms which are closed with respect to both term variables and polynomial variables. Here are the rules:

$$\frac{}{v \leftrightarrow \{v^1\}} \qquad \frac{}{f_i(v_1, \dots, v_m) \leftrightarrow \llbracket f \rrbracket_i(v_1, \dots, v_m)}$$

3.3 Processes

It is finally time to introduce the process terms of πDIBLL , which as already mentioned are a natural generalization of those of πDILL [11]. Due to space reasons, we concentrate on the aspects in which πDIBLL differs from πDILL , at the same time trying to be self-contained. More details can be found in [20].

► **Definition 6** (Process Syntax). *Given an infinite set of names, the set of processes, indicated with metavariables like P and Q is defined by the following grammar:*

$$\begin{aligned} P, Q ::= & 0 \mid P \mid Q \mid (\nu y) P \mid x\langle y \rangle.P \mid x(y).P \mid [x \leftarrow v] \mid \text{let } x = a \text{ in } P \mid \\ & x.P \mid !x(y).P \mid x.\text{inl}; P \mid x.\text{inr}; P \mid x.\text{case}(P, Q) \mid \\ & \text{if } v \text{ then } P \text{ else } Q \end{aligned}$$

where x, y are channel names from a set \mathcal{CV} such that $\mathcal{TV} \subseteq \mathcal{CV}$, a is a term and v is a value.

Most of the operators from the grammar above have the same meaning as in $\pi\mathbf{DILL}$, and are standard. Let us briefly describe the novel ones. The process $[x \leftarrow v]$ *outputs* the value v on the channel x . Dually, the process $x.P$ *inputs* a value v through the channel x , substituting it for any free occurrence of x in the process P . The process $\mathbf{let} \ x = a \ \mathbf{in} \ P$ serves to *evaluate* the term a , then substituting the outcome for x in P . As such, it is the operator incepting the probabilistic behaviour coming from terms into processes. Finally, the process $\mathbf{if} \ v \ \mathbf{then} \ P \ \mathbf{else} \ Q$ is as expected a *conditional* construction; observe that the argument is a value, and thus does not need to be evaluated. The grammar of processes we have just introduced is perfectly adequate to represent the process $PRIVK_{\Pi}$ as from Figure 1b. As an example, a process $P \oplus Q$ which evolves as either P or Q depending on the outcome of the flip of fair random coin can be written as the process

$$\mathbf{let} \ x = \mathit{flipcoin}() \ \mathbf{in} \ \mathbf{if} \ x \ \mathbf{then} \ P \ \mathbf{else} \ Q,$$

where $\mathit{flipcoin}$ is a function symbol such that $\mathit{typeof}(\mathit{flipcoin})$ is $\epsilon \rightarrow \mathbb{B}$ and the underlying algorithm $\mathit{comof}(\mathit{flipcoin})$ is the trivial constant-time procedure one can imagine.

The set of names occurring free in the process P (hereby denoted $fn(P)$) is defined as usual. The same holds for the capture avoiding substitution of a channel x or value v for y in a process P (denoted $P\{x/y\}$ and $P\{v/y\}$, respectively).

3.4 Process Reduction

Process reduction in $\pi\mathbf{DIBLL}$ is intrinsically probabilistic, and as such deserves to be described with some care. Structural congruence can be defined in a standard way as a binary relation \equiv on processes, see [20] for a formal definition. The reduction relation between processes is *not* a plain binary relation anymore, and instead puts a process P in correspondence with a *distribution* \mathcal{D} of processes, namely an object in the form $\{P_1^{r_1}, \dots, P_m^{r_m}\}$, where the P_i are processes and the r_i are positive real numbers summing to 1. We write $P \rightarrow \mathcal{D}$ in this case.

Let us now consider reduction rules, starting from the one for the \mathbf{let} construct, namely the only genuinely probabilistic one:

$$\frac{a \hookrightarrow \{v_i^{r_i}\}_{i \in I}}{\mathbf{let} \ x = a \ \mathbf{in} \ P \rightarrow \{P\{v_i/x\}^{r_i}\}_{i \in I}}$$

The other new operators can be evaluated in the expected way, giving rise to trivial (i.e. Dirac) distributions:

$$\frac{}{[x \leftarrow v] \mid x.P \rightarrow \{P\{v/x\}^1\}}$$

$$\frac{}{\mathbf{if} \ \mathbf{true} \ \mathbf{then} \ P \ \mathbf{else} \ Q \rightarrow \{P^1\}} \quad \frac{}{\mathbf{if} \ \mathbf{false} \ \mathbf{then} \ P \ \mathbf{else} \ Q \rightarrow \{Q^1\}}$$

Reduction axioms and rules from $\pi\mathbf{DILL}$ are all available in $\pi\mathbf{DIBLL}$, but must be appropriately tailored to the presence of distributions:

$$\frac{Q \rightarrow \{Q_i^{r_i}\}_{i \in I}}{P \mid Q \rightarrow \{P \mid Q_i^{r_i}\}_{i \in I}} \quad \frac{P \rightarrow \{Q_i^{r_i}\}_{i \in I}}{(\nu y) P \rightarrow \{(\nu y) Q_i^{r_i}\}_{i \in I}}$$

$$\frac{P \equiv R \quad R \rightarrow \{Q_i^{r_i}\}_{i \in I} \quad Q_i \equiv T_i \ \text{for every } i \in I}{P \rightarrow \{T_i^{r_i}\}_{i \in I}}$$

3.5 Type System

Traditionally, session typing serves the purpose of guaranteeing *safety* properties, like the absence of deadlocks. In this paper, however, they also enforce some bounds on the *complexity* of the reduction process, and as such have to be made more restricted.

Types. First of all, let us introduce the language of types, which is defined as follows:

$$A, B ::= 1 \mid A \multimap B \mid !_p A \mid A \otimes B \mid A \oplus B \mid A \& B \mid \mathbb{B} \mid \mathbb{S}[p]$$

Again, most of the type constructions are taken from $\pi\mathbf{DILL}$, but there are some significant and crucial differences:

- The indexed exponential type $!_p A$ takes the place of $!A$ as the type of replicated inputs; the presence of the polynomial p serves to impose an upper bound on the number of replicas, all of them of type A , which can be spawned.
- The ground types \mathbb{B} and $\mathbb{S}[p]$ are available as session types too, and here become the type of channels carrying a single value.

Type Environments. In dual intuitionistic linear logic, and thus in $\pi\mathbf{DILL}$, the typing environment to the left of the turnstile is split into two parts, namely an unrestricted part Γ and a linear part Δ . Here, we adopt the same notation, but the unrestricted part is modified in such a way that the polynomial limitation is made explicit when giving types to channels. In $\pi\mathbf{DIBLL}$, in other words, Γ contains assignments in the form $x_p : A$ where x is the name of an unrestricted channel, p is a polynomial and A is a type. Type environments, however, have to be further enriched with a *third* portion, denoted by Θ , consisting of assignments in the form $x : B$ where x is a term variable and B is a *ground* type. This reflects the possibility of having occurrences of *term* variables inside *processes*. The unrestricted portion of the type environment has to be manipulated with great care while typing processes, in particular in all binary typing rules. This requires the introduction of a (partial) binary operation \boxplus on unrestricted type environments such that if

$$\Gamma = \{x_{p_1}^1 : A_1, \dots, x_{p_n}^n : A_n\}; \quad \Xi = \{x_{q_1}^1 : A_1, \dots, x_{q_n}^n : A_n\};$$

then $\Gamma \boxplus \Xi$ is defined as $\{x_{p_1+q_1}^1 : A_1, \dots, x_{p_n+q_n}^n : A_n\}$. With the same hypotheses, we write that $\Gamma \sqsubseteq \Xi$ when $p_i \leq q_i$ for every $i \in \{1, \dots, n\}$, this way turning \sqsubseteq into a preorder.

Type Judgments. A *type judgment* is an expression in the form

$$\Gamma; \Delta; \Theta \vdash^{\mathcal{V}} P :: z : C$$

where Γ, Δ, Θ are the three aforementioned portions of the type environment, and P is a process offering a session of type C along the channel z . Polynomials can occur in type environments and types, and \mathcal{V} serves to declare all variables which might occur in those polynomials. As in $\pi\mathbf{DILL}$, we assume that all channels and variables declared in Γ, Δ , and Θ are distinct and different from z .

Typing Rules. One way $\pi\mathbf{DIBLL}$ deviates from $\pi\mathbf{DILL}$ are the typing rules related to the exponential connective $!$, namely those introducing the connective on the right and on the left, but also the rule capturing contraction, T_{copy} , and the so-called exponential cut rule.

$$\begin{array}{c}
\frac{\Gamma; \Delta; \Theta, x : B \vdash^{\mathcal{V}} Q :: T}{\Gamma; \Delta, x : B; \Theta \vdash^{\mathcal{V}} x.Q :: T} [TBL] \quad \frac{\Theta \vdash^{\mathcal{V}} v : B}{\Gamma; \Delta; \Theta \vdash^{\mathcal{V}} [x \leftarrow v] :: x : B} [TBR] \\
\\
\frac{\Theta \vdash^{\mathcal{V}} a : A \quad \Gamma; \Delta; \Theta, x : A \vdash^{\mathcal{V}} P :: T}{\Gamma; \Delta; \Theta \vdash^{\mathcal{V}} \text{let } x = a \text{ in } P :: T} [T\text{let}] \\
\\
\frac{\Theta \vdash^{\mathcal{V}} v : \mathbb{B} \quad \Gamma; \Delta; \Theta \vdash^{\mathcal{V}} P :: x : A \quad \Gamma; \Delta; \Theta \vdash^{\mathcal{V}} Q :: x : A}{\Gamma; \Delta; \Theta \vdash^{\mathcal{V}} \text{if } v \text{ then } P \text{ else } Q :: x : A} [T\text{if}]
\end{array}$$

■ **Figure 2** Typing Rules for Ground Types, the `let` construct, and Conditionals.

For example, the T_{copy} rule, which allows to type processes which perform output actions on an unrestricted channel $u_p \in \Gamma$, must now keep track of the increase in p :

$$\frac{\Gamma, u_p : A; \Delta, y : A; \Theta \vdash^{\mathcal{V}} P :: T}{\Gamma, u_{p+1} : A; \Delta; \Theta \vdash^{\mathcal{V}} (\nu y) u(y).P :: T} [T_{copy}]$$

This, however, is not enough to enforce the polynomial bounds we aim at from within the type system. Whenever typing a parallel composition $P \mid Q$, it is necessary to appropriately account for the use of any unrestricted channel by *both* P and Q , thus splitting the unrestricted part of the underlying type environment. More concretely, any typing rule dealing with parallel composition has to make appropriate use of the \boxplus operator, applying to the two unrestricted environments at hand, the result of this sum thus weakened through the partial order \sqsubseteq . For example, the T_{cut} rule is modified as follows

$$\frac{\Gamma_1 \boxplus \Gamma_2 \sqsubseteq \Gamma \quad \Gamma_1; \Delta_1; \Theta \vdash^{\mathcal{V}} P :: x : A \quad \Gamma_2; \Delta_2, x : A; \Theta \vdash^{\mathcal{V}} Q :: T}{\Gamma; \Delta_1, \Delta_2; \Theta \vdash^{\mathcal{V}} (\nu x) (P \mid Q) :: T} [T_{cut}]$$

All typing rules involving a parallel composition between two processes have to be amended similarly. The remaining typing rules, and in particular those for the additive connectives \oplus and $\&$, are exactly as in $\pi\mathbf{DILL}$. We conclude by giving the typing rules for the new process operators, which are in Figure 2, and are all standard. It is worth observing that in these rules we implicitly refer to the type system for terms and values in the rules TBR , $T\text{if}$, and $T\text{let}$.

4 Safety and Reachability

In this section we prove some properties about the transition system induced by the reduction relation \rightarrow , as introduced in Section 3.4. Before delving into the details, a couple of remarks are in order. Although the relation \rightarrow is defined for arbitrary processes, we will be concerned with the reduction of typable *closed* processes, namely those processes which can be typed under *empty* Θ and \mathcal{V} . In fact, reducing processes in which term variables occur free does not make sense when reduction is supposed to model computation (as opposed to equational reasoning), like here. When Θ or \mathcal{V} are empty, we simply omit them from the underlying typing judgment. Reduction being probabilistic, it is convenient to introduce some other reduction relations, all of them derived from \rightarrow :

► **Definition 7** (Auxiliary Reduction Relations). *We first of all define a relation \mapsto on plain processes by stipulating that $P \mapsto R$ iff $P \rightarrow \mathcal{D}$ and $R \in S(\mathcal{D})$. We also need another reduction relation \Rightarrow as the monadic lifting of \rightarrow , thus a relation on process distributions:*

$$\frac{R_i \rightarrow \mathcal{E}_i \text{ for every } i \in I}{\{R_i^{r_i}\}_{i \in I} \Rightarrow \sum_{i \in I} r_i \cdot \mathcal{E}_i}$$

Finally, it is convenient to put any process P in relation with the distribution of irreducible processes to which P evaluates:

$$\frac{P \text{ is irreducible}}{P \Rightarrow \{P^1\}} \quad \frac{P \rightarrow \{R_i^{r_i}\}_{i \in I} \quad R_i \Rightarrow \mathcal{E}_i \text{ for every } i \in I}{P \Rightarrow \sum_{i \in I} r_i \cdot \mathcal{E}_i}$$

The relation \mapsto is perfectly sufficient to capture the qualitative aspects of the other reduction relations, e.g., if $P \rightarrow \mathcal{D}$ then for every $R \in S(\mathcal{D})$ it holds that $P \mapsto R$. Indeed, in the rest of this section we will be concerned with \mapsto , only.

4.1 Subject Reduction

The property of subject reduction is the minimal requisite one asks to a type system, and says that types are preserved along reduction. In $\pi\mathbf{DIBLL}$, as in $\pi\mathbf{DILL}$, this property holds:

► **Theorem 8** (Subject Reduction). *If $\Gamma; \Delta \vdash P :: z : C$ and $P \mapsto R$, then it holds that $\Gamma; \Delta \vdash R :: z : C$.*

Following [11], subject reduction can be proved by carefully inspecting how P can be reduced to R , which can happen as a result of either communication, the evaluation of a term, or the firing of a conditional construction. Many cases have to be analysed, some of them not being present in $\pi\mathbf{DILL}$. A novel aspect is a lemma about the typing of processes obtained by substituting term variables with values. More details can be found in [20], where a progress property is also given, very much in the style of that from [11].

4.2 Polytime Soundness

As already mentioned in Section 1, subject reduction is not the only property one is interested in proving about reduction in $\pi\mathbf{DIBLL}$. In fact, the latter has been designed to guarantee polynomial bounds on reduction time, as prescribed by the computational model of cryptography. But what do we mean by that, exactly? What is the underlying parameter on which the polynomial depends? In cryptography, computation time must be polynomially bounded on the value of the so-called *security parameter* which, as we hinted at already, is modelled by an element of \mathcal{V} . As a consequence, what we are actually referring to are bounds *parametric* on the value of the polynomial variables which P mentions in its type judgments, i.e. the \mathcal{V} in

$$\Gamma; \Delta \vdash^{\mathcal{V}} P :: z : C \tag{2}$$

Doing so, we have to keep in mind that process reduction is only defined on *closed* processes. We can thus proceed in three steps:

- We can first of all assign a \mathcal{V} -polynomial $\mathbf{W}(\pi)$ to every type derivation π with conclusion mentioning \mathcal{V} . This is done by induction on the structure of π .
- We then prove that for closed type derivations, $\mathbf{W}(\cdot)$ strictly decreases along process reduction, at the same time taking the cost of each reduction step into account. In other words, if π is closed and types P where $P \mapsto Q$, then a type derivation ξ for Q can be found such that $\mathbf{W}(\pi) \geq \mathbf{W}(\xi) + k$, where k is the cost of the reduction leading P to Q . (In most cases k is set to be 1, the only exception being the evaluation of a **let** operator, which might involve the evaluation of costly functions.)

- Finally, the previous two points must be proved to interact well, and this is done by showing that for every type derivation π with conclusion in the form (2) and for every \mathcal{V} -substitution ρ , there is a type derivation $\pi\rho$ with conclusion

$$\Gamma\rho; \Delta\rho \vdash P\rho :: z : C\rho$$

such that, crucially, $\mathbf{W}(\pi\rho) = \mathbf{W}(\pi)\rho$. In other words, the weight functor on type derivations commutes well with substitutions.

Altogether, this allows us to reach the following:

- **Theorem 9 (Polytime Soundness).** *For every derivation π typing P , there is a polynomial p_π such that for every substitution ρ , if $P\rho \mapsto^* Q$, then the overall computational cost of the aforementioned reduction is bounded by $p_\pi(\rho)$.*

We can thus claim that, e.g., every process ADV such that

$$\vdash^n ADV :: c : \mathbb{S}[p] \otimes \mathbb{S}[p] \otimes (\mathbb{S}[p] \multimap \mathbb{B})$$

can actually be evaluated in probabilistic polynomial time, since out of it one can type the processes R_0, R_1, R_{fun} computing the three components in ADV 's type. Moreover, since \mathcal{F} can be made large enough to be complete for PPT (see, e.g., [22]), one can also claim that all probabilistic (first-order) polytime behaviours can be captured from within $\pi\mathbf{DIBLL}$.

5 Typable Processes and Their Probabilistic Behaviour

The properties we proved in the last section, although remarkable, are agnostic to the probabilistic nature of $\pi\mathbf{DIBLL}$. It is now time to investigate the genuinely quantitative aspects of the calculus.

5.1 Confluence

It might seem weird that confluence can be proved for a calculus in which internal probabilistic choice is available. In fact, there are *two* forms of nondeterministic evolution $\pi\mathbf{DIBLL}$ processes can give rise to, the first one coming from the presence of terms which can evolve probabilistically, the second one instead due, as in $\pi\mathbf{DILL}$, to the presence of parallel composition, itself offering the possibility of concurrent interaction. Confluence is meant to address the latter, not the former. More specifically, we would like to prove that whenever a typable process P evolves towards two distinct distributions \mathcal{D} and \mathcal{E} , the latter can be somehow unified. This indeed turns out to be the case:

- **Theorem 10 (Confluence).** *If $\Gamma; \Delta \vdash P :: T$ and $\mathcal{D} \leftarrow P \rightarrow \mathcal{E}$ then either $\mathcal{D} = \mathcal{E}$ or there exists \mathcal{F} such that $\mathcal{D} \Rightarrow \mathcal{F} \Leftarrow \mathcal{E}$.*

In other words, while probabilistic evolution coming from terms is unavoidable, the choice of how to reduce a typable process does not matter, in the spirit of what happens in sequential languages like the λ -calculus. Notice, however, that confluence holds in a very strong sense here, i.e., Theorem 10 has the flavour of the so-called diamond property. The proof of it, which can be found in [20], exploits a characterization of reduction semantics by way of labelled semantics, the latter rendering the proof simpler.

Among the corollaries of confluence, one can prove that the way a typable process is reduced is irrelevant as far as the resulting distribution is concerned:

- **Corollary 11 (Strategy Irrelevance).** *If $\Gamma; \Delta \vdash P :: T$ and $\mathcal{D} \Leftarrow P \Rightarrow \mathcal{E}$, then $\mathcal{D} = \mathcal{E}$.*

5.2 Relational Reasoning

Strategy irrelevance has a positive impact on the definition of techniques for relational reasoning on typed processes. In this section, we are concerned precisely with introducing a form of observational equivalence, which turns out to have the shape one expects it to have in the realm of sequential languages. This is a simplification compared to similar notions from the literature on process algebras for the computational model of cryptography [42].

When defining observational equivalence, we want to dub two processes as being equivalent when they behave *the same* in *any environment*. We thus have to formalize environments and observable behaviours. The former, as expected, is captured through the notion of a *context*, which here takes the form of a term in which a single occurrence of the hole $[\cdot]$ is allowed to occur *in linear position* (i.e. outside the scope of the any replication).

$$\begin{aligned} \mathcal{C}[\cdot] ::= & [\cdot] \mid \mathcal{C}[\cdot] \mid Q \mid P \mid \mathcal{C}[\cdot] \mid (\nu y) \mathcal{C}[\cdot] \mid x\langle y \rangle.\mathcal{C}[\cdot] \mid x(y).\mathcal{C}[\cdot] \mid \\ & x.\mathcal{C}[\cdot] \mid x.\text{inl};\mathcal{C}[\cdot] \mid x.\text{inr};\mathcal{C}[\cdot] \mid x.\text{case}(\mathcal{C}[\cdot], Q) \mid x.\text{case}(P, \mathcal{C}[\cdot]) \mid \\ & \text{let } x = a \text{ in } \mathcal{C}[\cdot] \mid \text{if } v \text{ then } \mathcal{C}[\cdot] \text{ else } Q \mid \text{if } v \text{ then } P \text{ else } \mathcal{C}[\cdot] \end{aligned}$$

On top of contexts, it is routine to give a type system deriving judgments in the form

$$\Gamma; \Delta; \Theta \vdash^{\mathcal{V}} \mathcal{C}[(\Xi; \Psi; \Phi \vdash^{\mathcal{V}} \cdot :: T)] :: U$$

guaranteeing that whenever a process P is such that $\Xi; \Psi; \Phi \vdash^{\mathcal{V}} P :: T$, it holds that $\Gamma; \Delta; \Theta \vdash^{\mathcal{V}} \mathcal{C}[P] :: U$.

Talking about observations, what we are interested in observing here is the *probability* of certain very simple events. For example, in Section 2 we hint at the fact that the experiment PrivK^{eav} can be naturally modelled as a process offering a channel carrying just a boolean value. This will very much inform our definition. Suppose that $\vdash^{\mathcal{V}} P :: x : \mathbb{B}$. After having instantiated P through a substitution $\rho : \mathcal{V} \rightarrow \mathbb{N}$, some internal reduction steps turn $P\rho$ into a (unique) distribution \mathcal{D} of irreducible processes, and the only thing that can happen at that time is that a boolean value b is produced in output along the channel x . We write $P \downarrow_x^\rho b$ for this probabilistic event, itself well defined thanks to Theorem 9, Corollary 11 (which witness the existence and unicity of \mathcal{D} , respectively), and Progress (see [20] for more details).

Finally, we are ready to give the definition of observational equivalence.

► **Definition 12** (Observational Equivalence). *Let P and Q two processes such that $\Gamma; \Delta; \Theta \vdash^{\mathcal{V}} P, Q :: T$. We say that P and Q are observationally equivalent iff for every context $\mathcal{C}[\cdot]$ such that $\vdash^{\mathcal{V}} \mathcal{C}[(\Gamma; \Delta; \Theta \vdash^{\mathcal{V}} \cdot :: T)] :: x : \mathbb{B}$, there is a negligible function $\varepsilon : (\mathcal{V} \rightarrow \mathbb{N}) \rightarrow \mathbb{R}_{[0,1]}$ such that for every ρ it holds that $|\Pr[\mathcal{C}[P] \downarrow_x^\rho v] - \Pr[\mathcal{C}[Q] \downarrow_x^\rho v]| \leq \varepsilon(\rho)$. In that case we write $\Gamma; \Delta; \Theta \vdash^{\mathcal{V}} P \cong Q :: T$, or just $P \cong Q$ if this does not cause any ambiguity.*

Proving pairs of processes to be observationally equivalent is notoriously hard, due to the presence of a universal quantification over all contexts. However, the fact πDIBLL enjoys confluence facilitates the task. In particular, we can prove certain pairs of processes to be observationally equivalent, and this will turn out to be very useful in the next section. The first equation we can prove is the commutation of input prefixes and the **let** construct:

$$x(y).\text{let } z = t \text{ in } P \cong \text{let } z = t \text{ in } x(y).P \tag{3}$$

As innocuous as it seems, this equation is not validated by, e.g., probabilistic variations on bisimilarity, being (essentially) the classic counterexample to the coincidence of the latter and trace equivalence. In fact, Equation 3 indeed holds here, see [20] for the details. An equation scheme which can be easily proved to be sound for observational equivalence is the one induced by so-called *Kleene-equivalence*: if P and Q are such that there exists a distribution \mathcal{D} such that both $P\rho \Rightarrow \mathcal{D}$ and $Q\rho \Rightarrow \mathcal{D}$ (for every ρ), then we can safely conclude that $P \cong Q$. Finally, an equation in which the approximate nature of observational equivalence comes into play is the following one:

$$x.[y \leftarrow x] \cong x.\text{let } z = \text{rand in let } b = \text{eq}(x, z) \text{ in if } b \text{ then } [y \leftarrow x] \text{ else } [y \leftarrow z]$$

The two involved processes, both offering a session on y having type $\mathbb{S}[n]$ by way of a session with the same type on x , behave the same, except on *one* string z chosen at random.

6 A Simple Cryptographic Proof

In this section, we put relational reasoning at work on the simple example we introduced in Section 2. More specifically, we will show that the notion of observational equivalence from Section 5.2 is sufficient to prove Equation 1, where \sim is taken to be the non-contextual version of observational equivalence. We will do that for an encryption scheme Π_g such that Enc is based on a pseudorandom generator g , i.e. Enc returns on input a message m and a key k the ciphertext $xor(m, g(k))$. When can such a function g be said to be pseudorandom? This happens when the output of g is indistinguishable from a truly random sequence of the same length. This, in turn, can be spelled out as the equation

$$OUTPR_g \cong OUTR \tag{4}$$

where $OUTR$ is a process outputting a random string of polynomial length of a channel out , while $OUTPR_g$ is a process outputting a *pseudorandom* such string produced according to g .

We now want to prove, given (4), that (1) holds, the latter now taking the following form:

$$\nu adv.(PRIVK_{\Pi} \mid ADV) \sim FAIRFLIP_{exp}.$$

Following the textbook proof of this result (see, e.g., [35]), we can structure the proof as a construction, out of ADV , of a distinguisher D_{ADV} having type $out : \mathbb{S}[p] \vdash^n D_{ADV} :: exp : \mathbb{B}$ such that the following two equations hold:

$$\nu out.(OUTPR_G \mid D_{ADV}) \sim \nu adv.(PRIVK_{\Pi} \mid ADV) \tag{5}$$

$$\nu out.(OUTR \mid D_{ADV}) \sim FAIRFLIP \tag{6}$$

Actually, the construction of D_{ADV} is very simple, being it the process

$$\nu adv.(PRIVKEYK_{OTP} \mid ADV),$$

where OTP is the so-called one-time pad encryption scheme, and $PRIVKEYK_{OTP}$ is the process obtained from $PRIVK_{OTP}$ by delegating the computation of the key to a subprocess:

$PRIVKEYK_{OTP} :$ input m_0 from adv ; input m_1 from adv ; let $b = \text{flipcoin}()$ in input k from out ;	let $c = xor(k, m_b)$ in output c to adv ; input g from adv ; let $r = eq(g, b)$ in output r to exp ;
--	---

By construction, and using some of the equations we mentioned in Section 5, one can prove that $PRIVK_{\Pi_g} \cong \nu out.(OUTPR_g \mid PRIVKEYK_{OTP})$, from which, by congruence of \cong , one derives Equation (5):

$$\begin{aligned} \nu out.(OUTPR_G \mid D_{ADV}) &\equiv \nu out.(OUTPR_G \mid (\nu adv.(PRIVKEYK_{\Pi} \mid ADV))) \\ &\equiv \nu adv.(\nu out.(OUTPR_G \mid PRIVKEYK_{\Pi}) \mid ADV) \\ &\sim \nu adv.(PRIVK_{\Pi} \mid ADV). \end{aligned}$$

Since $PRIVK_{OTP} \cong \nu out.(OUTR \mid PRIVKEYK_{OTP})$, one can similarly derive that

$$\nu out.(OUTR_g \mid D_{ADV}) \sim \nu adv.(PRIVK_{OTP} \mid ADV).$$

It is well known, however, that the *OTP* encryption scheme is perfectly secure, which yields Equation (6).

7 Conclusion

Contributions. In this paper, we show how the discipline of session types can be useful in modelling and reasoning about cryptographic experiments. The use of sessions, in particular, allows to resolve the intrinsic nondeterminism of process algebras without the need for a scheduler, thus simplifying the definitional apparatus. The keystone to that is a confluence result, from which it follows that the underlying reduction strategy (i.e. the scheduler) does not matter: the distribution of irreducible processes one obtains by reducing a typable process is unique. The other major technical results about the introduced system of session types are a polynomial bound on the time necessary to reduce any typable process, together with a notion of observational equivalence through which it is possible to faithfully capture computational indistinguishability, a key notion in modern cryptography.

Future Work. This work, exploratory in nature, leaves many interesting problems open. Currently, the authors are investigating the applicability of $\pi\mathbf{DIBLL}$ to more complex experiments than that considered in Section 6. In particular, the ability to build higher-order sessions enables the modelling of adversaries which have access to an oracle, but also of experiments involving such adversaries. As an example, an *active* adversary $ACTADV$ to an encryption scheme would have type

$$\vdash^n ACTADV :: c \;!_q(\mathbb{S}[p] \multimap \mathbb{S}[p]) \multimap \mathbb{S}[p] \otimes \mathbb{S}[p] \otimes (\mathbb{S}[p] \multimap \mathbb{B}).$$

This reflects the availability of an oracle, modelled as a server for the encryption function, which can be accessed only a polynomial amount of times. Being able to capture *all* those adversaries within our calculus seems feasible, but requires extending the grammar of processes with an iterator combinator. On the side of relational reasoning, notions of equivalence are being studied which are sound with respect to observational equivalence, that is, included in it, at the same time being handier and avoiding any universal quantification on contexts. The use of logical relations or bisimulation, already known in $\pi\mathbf{DILL}$ [11] can possibly be adapted to $\pi\mathbf{DIBLL}$, but does not allow to faithfully capture linearity, falsifying equations (like (3)) which are crucial in concrete proofs. As a consequence, we are considering forms of trace equivalence and distribution-based bisimilarity [19], since the latter are known to be fully abstract with respect to (linear) observational equivalence, even in presence of effects.

Related Work. We are certainly not the first to propose a formal calculus in which to model cryptographic constructions and proofs according to the computational model. The so-called Universal Composability model (UC in the following), introduced by Canetti more than twenty years ago [12, 13], has been the subject of many investigations aimed at determining if it is possible to either simplify it or to capture it by way of a calculus or process algebra (e.g. [14, 38, 37, 6]). In all the aforementioned works, a tension is evident between the need to be expressive, so as to capture UC proofs, and the need to keep the model simple enough, masking the details of probability and complexity as much as possible. π **DIBLL** is too restrictive to capture UC in its generality, but on the other hand it is very simple and handy. As for the approaches based on process algebras, it is once again worth mentioning the series of works due to Mitchell et al. and based, like ours, on a system of types derived from bounded linear logic [39, 41, 42]. As already mentioned, the main difference with this work is the absence of a system of behavioural types such as session types, which forces the framework to be complex, relying on a further quantification on probabilistic schedulers, which is not needed here. Another very interesting line of work is the one about imperative calculi, like the one on which tools like **EasyCrypt** are based [8, 7]. Recently, there have been attempts at incepting some form of probabilistic behaviour into session types, either by allowing for probabilistic internal choice in multiparty sessions [4], or by enriching the type system itself, by making it quantitative in nature [34]. The system π **DIBLL** is certainly more similar to the former, in that randomization does not affect the type structure but only the process structure. This design choice is motivated by our target applications, namely cryptographic experiments, in which randomization affects *which* strings protocols and adversaries produce, rather than their high-level behaviour. Indeed, our calculus is closer in spirit to some previous work on cryptographic constructions in λ -calculi [43] and logical systems [33], although the process algebraic aspects are absent there. Finally, session types have also been used as an handy tool guaranteeing security properties like information flow or access control (see, e.g., [15, 9]), which are however different from those we are interested at here. The literature on session types and their relations with linear logic is vast, and deeply influenced the way π **DIBLL** is defined. As an example, the way we handle ground types is very inspired by the one adopted by Toninho, Caires and Pfenning [47]. Moreover, indexed modalities in the style of bounded linear logic have been already considered, e.g., by Kokke et al [36] as a way to tame the inherent nondeterminism arising from a more permissive typing rule for parallel composition. Das and Pfenning [24], instead, label ground types with a sequence of arithmetic expressions obtaining a notion of refinement which is similar, although much more expressive, than the one we use here. In all these works, however, the underlying objective is different from ours, which consists in enforcing complexity bounds in the context of cryptographic experiments.

References

- 1 Martín Abadi, Ricardo Corin, and Cédric Fournet. Computational secrecy by typing for the pi calculus. In *Proc. of APLAS 2006*, volume 4279 of *LNCS*, pages 253–269. Springer, 2006. doi:10.1007/11924661_16.
- 2 Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proc. of POPL 2001*, pages 104–115. ACM, 2001. doi:10.1145/360204.360213.
- 3 Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. *Inf. Comput.*, 148(1):1–70, 1999. doi:10.1006/inco.1998.2740.
- 4 Bogdan Aman and Gabriel Ciobanu. Probabilities in session types. In *Proc. of FROM 2019*, volume 303 of *EPTCS*, pages 92–106, 2019. doi:10.4204/EPTCS.303.7.

- 5 Michael Backes, Esfandiar Mohammadi, and Tim Ruffing. Computational soundness results for proverif - bridging the gap from trace properties to uniformity. In *Proc. of POST 2014*, volume 8414 of *LNCS*, pages 42–62. Springer, 2014. doi:10.1007/978-3-642-54792-8_3.
- 6 Manuel Barbosa, Gilles Barthe, Benjamin Grégoire, Adrien Koutsos, and Pierre-Yves Strub. Mechanized proofs of adversarial complexity and application to universal composability. In *Proc. of CCS 2021*, pages 2541–2563. ACM, 2021. doi:10.1145/3460120.3484548.
- 7 Gilles Barthe, Thomas Espitau, Benjamin Grégoire, Justin Hsu, Léo Stefanescu, and Pierre-Yves Strub. Relational reasoning via probabilistic coupling. In *Proc. of LPAR 2015*, volume 9450 of *LNCS*, pages 387–401. Springer, 2015. doi:10.1007/978-3-662-48899-7_27.
- 8 Gilles Barthe, Cédric Fournet, Benjamin Grégoire, Pierre-Yves Strub, Nikhil Swamy, and Santiago Zanella Béguelin. Probabilistic relational verification for cryptographic implementations. In *Proc. of POPL 2014*, pages 193–206. ACM, 2014. doi:10.1145/2535838.2535847.
- 9 Massimo Bartoletti, Ilaria Castellani, Pierre-Malo Deniérou, Mariangiola Dezani-Ciancaglini, Silvia Ghilezan, Jovanka Pantovic, Jorge A. Pérez, Peter Thiemann, Bernardo Toninho, and Hugo Torres Vieira. Combining behavioural types with security analysis. *J. Log. Algebraic Methods Program.*, 84(6):763–780, 2015. doi:10.1016/j.jlamp.2015.09.003.
- 10 Laura Bocchi, Tzu-Chun Chen, Romain Demangeon, Kohei Honda, and Nobuko Yoshida. Monitoring networks through multiparty session types. *Theor. Comput. Sci.*, 669:33–58, 2017. doi:10.1016/j.tcs.2017.02.009.
- 11 Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *Proc. of CONCUR 2010*, pages 222–236. Springer, 2010. doi:10.1007/978-3-642-15375-4_16.
- 12 Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. of FOCS 2001*, pages 136–145. IEEE, 2001. doi:10.1109/SFCS.2001.959888.
- 13 Ran Canetti. Universally composable security. *J. ACM*, 67(5):28:1–28:94, 2020. doi:10.1145/3402457.
- 14 Ran Canetti, Alley Stoughton, and Mayank Varia. Easyuc: Using easycrypt to mechanize proofs of universally composable security. In *Proc. of CSF 2019*, pages 167–183. IEEE, 2019. doi:10.1109/CSF.2019.00019.
- 15 Sara Capocchi, Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Tamara Rezk. Session types for access and information flow control. In *Proc. of CONCUR 2010*, volume 6269 of *LNCS*, pages 237–252. Springer, 2010. doi:10.1007/978-3-642-15375-4_17.
- 16 David Castro-Perez, Raymond Hu, Sung-Shik Jongmans, Nicholas Ng, and Nobuko Yoshida. Distributed programming using role-parametric session types in go: statically-typed endpoint apis for dynamically-instantiated communication structures. *Proc. of POPL*, 3:29:1–29:30, 2019. doi:10.1145/3290342.
- 17 Hubert Comon-Lundh, Masami Hagiya, Yusuke Kawamoto, and Hideki Sakurada. Computational soundness of indistinguishability properties without computable parsing. In *Proc. of ISPEC 2012*, volume 7232 of *LNCS*, pages 63–79. Springer, 2012. doi:10.1007/978-3-642-29101-2_5.
- 18 Ugo Dal Lago and Paolo Di Giamberardino. On session types and polynomial time. *Mathematical Structures in Computer Science*, 26(8):1433–1458, 2016. doi:10.1017/S0960129514000632.
- 19 Ugo Dal Lago and Francesco Gavazzo. Resource transition systems and full abstraction for linear higher-order effectful programs. In *Proc. of FSCD 2021*, volume 195 of *LIPICs*, pages 23:1–23:19, 2021. doi:10.48550/arXiv.2106.12849.
- 20 Ugo Dal Lago and Giulia Giusti. On session typing, probabilistic polynomial time, and cryptographic experiments (long version). Available at arXiv:2207.03360, 2022.
- 21 Ugo Dal Lago and Martin Hofmann. Bounded linear logic, revisited. *Log. Methods Comput. Sci.*, 6(4), 2010. doi:10.1007/978-3-642-02273-9_8.
- 22 Ugo Dal Lago, Sara Zuppiroli, and Maurizio Gabbriellini. Probabilistic recursion theory and implicit computational complexity. *Sci. Ann. Comput. Sci.*, 24(2):177–216, 2014. doi:10.48550/arXiv.1406.3378.

- 23 Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. *Inf. Comput.*, 256:253–286, 2017. doi:10.1016/j.ic.2017.06.002.
- 24 Ankush Das and Frank Pfenning. Session Types with Arithmetic Refinements. In *Proc. of CONCUR 2020*, volume 171, pages 13:1–13:18, 2020. doi:10.4230/LIPIcs.CONCUR.2020.13.
- 25 Danny Dolev and Andrew Yao. On the security of public key protocols. *IEEE*, 29(2):198–208, 1983. doi:10.1109/TIT.1983.1056650.
- 26 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.
- 27 Jean-Yves Girard, Andre Scedrov, and Philip J Scott. Bounded linear logic: a modular approach to polynomial-time computability. *Theor. Comput. Sci.*, 97(1):1–66, 1992. doi:10.1016/0304-3975(92)90386-T.
- 28 Oded Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, 2006.
- 29 Martin Hofmann and Philip J. Scott. Realizability models for bil-like languages. *Theor. Comput. Sci.*, 318(1-2):121–137, 2004. doi:10.1016/j.tcs.2003.10.019.
- 30 Kohei Honda. Types for dyadic interaction. In *Proc. of CONCUR 1993*, pages 509–523. Springer, 1993. doi:10.1007/3-540-57208-2_35.
- 31 Hans Hüttel, Ivan Lanese, Vasco T Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniérou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, et al. Foundations of session types and behavioural contracts. *ACM Computing Surveys (CSUR)*, 49(1):1–36, 2016. doi:10.1145/2873052.
- 32 Atsushi Igarashi, Peter Thiemann, Yuya Tsuda, Vasco T. Vasconcelos, and Philip Wadler. Gradual session types. *J. Funct. Program.*, 29:e17, 2019. doi:10.1017/S0956796819000169.
- 33 Russell Impagliazzo and Bruce M. Kapron. Logics for reasoning about cryptographic constructions. *J. Comput. Syst. Sci.*, 72(2):286–320, 2006. doi:10.1016/j.jcss.2005.06.008.
- 34 Omar Inverso, Hernán C. Melgratti, Luca Padovani, Catia Trubiani, and Emilio Tuosto. Probabilistic analysis of binary sessions. In *Proc. of CONCUR 2020*, volume 171 of *LIPIcs*, pages 14:1–14:21, 2020. doi:10.4230/LIPIcs.CONCUR.2020.14.
- 35 Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2020.
- 36 Wen Kokke, J. Garrett Morris, and Philip Wadler. Towards races in linear logic. In *Proc. of COORDINATION 2019*, volume 11533 of *LNCS*, pages 37–53. Springer, 2019. doi:10.1007/978-3-030-22397-7_3.
- 37 Ralf Küsters, Max Tuengerthal, and Daniel Rausch. The IITM model: A simple and expressive model for universal composability. *J. Cryptol.*, 33(4):1461–1584, 2020. doi:10.1007/s00145-020-09352-1.
- 38 Kevin Liao, Matthew A. Hammer, and Andrew Miller. Ilc: a calculus for composable, computational cryptography. In *Proc. of PLDI 2019*, pages 640–654. ACM, 2019. doi:10.1145/3314221.3314607.
- 39 Patrick Lincoln, John C. Mitchell, Mark Mitchell, and Andre Scedrov. Probabilistic polynomial-time equivalence and security analysis. In *Proc. of FM 1999*, volume 1708 of *LNCS*, pages 776–793. Springer, 1999. doi:10.1007/3-540-48119-2_43.
- 40 John C. Mitchell, Mark Mitchell, and Andre Scedrov. A linguistic characterization of bounded oracle computation and probabilistic polynomial time. In *Proc. of FOCS 1998*, pages 725–733. IEEE, 1998. doi:10.1109/SFCS.1998.743523.
- 41 John C. Mitchell, Ajith Ramanathan, Andre Scedrov, and Vanessa Teague. Probabilistic polynomial-time process calculus and security protocol analysis. In *Proc. LICS 2001*, pages 3–5. IEEE, 2001. doi:10.1109/LICS.2001.932477.
- 42 John C. Mitchell, Ajith Ramanathan, Andre Scedrov, and Vanessa Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theor. Comput. Sci.*, 353(1-3):118–164, 2006. doi:10.1016/S1571-0661(04)80968-X.
- 43 David Nowak and Yu Zhang. A calculus for game-based security proofs. In *Proc. of RPROVSEC 2010*, volume 6402 of *LNCS*, pages 35–52. Springer, 2010. doi:10.1007/978-3-642-16280-0_3.

- 44 Jorge A Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations and observational equivalences for session-based concurrency. *Inf. Comput.*, 239:254–302, 2014. doi:10.1016/j.ic.2014.08.001.
- 45 Paula Severi, Luca Padovani, Emilio Tuosto, and Mariangiola Dezani-Ciancaglini. On sessions and infinite data. In Alberto Lluch-Lafuente and José Proença, editors, *Proc. of COORDINATION 2016*, volume 9686 of *LNCS*, pages 245–261. Springer, 2016. doi:10.48550/arXiv.1610.06362.
- 46 Jianxiong Shao, Yu Qin, and Dengguo Feng. Computational soundness results for stateful applied π calculus. In *Proc. of POST 2016*, volume 9635 of *LNCS*, pages 254–275. Springer, 2016. doi:10.1007/978-3-662-49635-0_13.
- 47 Bernardo Toninho, Luís Caires, and Frank Pfenning. Dependent session types via intuitionistic linear type theory. In *Proc. of PPDP 2011*, pages 161–172. ACM, 2011. doi:10.1145/2003476.2003499.
- 48 Philip Wadler. Propositions as sessions. *J. Funct. Program.*, 24(2-3):384–418, 2014. doi:10.1145/2398856.2364568.