26th International Conference on Concurrency Theory

CONCUR'15, September 1–4, 2015, Madrid, Spain

Edited by

Luca Aceto

David de Frutos Escrig

Editors

Luca Aceto
School of Computer Science
Reykjavik University
luca@ru.is

David de Frutos Escrig
Facultad de Matemáticas
Universidad Complutense de Madrid
defrutos@sip.ucm.es

ACM Classification 1998
C.2 Computer-Communication Networks, D.2 Software Engineering, F.1 Computation by Abstract Devices, F.3 Logics and Meanings of Programs, I.6 Simulation and Modeling

ISBN 978-3-939897-91-0

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-91-0.

Publication date
August, 2015

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 http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.CONCUR.2015.i

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

http://www.dagstuhl.de/lipics

Contents

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig

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

Preface

This volume contains the proceedings of the 26th Conference on Concurrency Theory (CONCUR 2015), which was held in Madrid, Spain, in the period 1–4 September, 2015. CONCUR 2015 was organized by the Universidad Complutense de Madrid.

The purpose of the CONCUR conferences is to bring together researchers, developers and students in order to contribute to the development and dissemination of the theory of concurrency and its applications. Twenty five years after our first meeting in 1990, it is still the reference annual event for researchers in this field.

The principal topics include basic models of concurrency such as abstract machines, domain theoretic models, game theoretic models, process algebras, and Petri nets; logics for concurrency such as modal logics, probabilistic and stochastic logics, temporal logics, and resource logics; models of specialized systems such as biology-inspired systems, circuits, hybrid systems, mobile and collaborative systems, multi-core processors, probabilistic systems, real-time systems, service-oriented computing, and synchronous systems; verification and analysis techniques for concurrent systems such as abstract interpretation, atomicity checking, model checking, race detection, pre-order and equivalence checking, run-time verification, state-space exploration, static analysis, synthesis, testing, theorem proving, and type systems; related programming models such as distributed, component-based, object-oriented, and web services.

This edition of the conference attracted 93 full paper submissions. We would like to thank all their authors for their interest in CONCUR 2015. After careful reviewing and discussions, the Program Committee selected 33 papers for presentation at the conference. Each submission was reviewed by at least three reviewers, who wrote detailed evaluations and gave insightful comments. The Conference Chairs warmly thank all the members of the Program Committee and all the additional reviewers for their excellent work, as well as for the constructive discussions.

The conference program was further greatly enriched by the invited talks by Gianluigi Zavattaro (joint invited speaker with TGC 2015), James Worrell (plenary speaker at Madrid meet 2015), Mohammad Reza Mousavi, and Alexandra Silva. We had also a second plenary talk to celebrate the 25th anniversary of CONCUR. To mark this special occasion, it was really a pleasure and a great honour to host a keynote address by Matthew Hennessy, who was one of the invited speakers at the first CONCUR, and is a researcher who has had a big influence on the development of the Theory of Concurrency as a whole, and, in particular, on the professional trajectory of the two PC chairs of this edition of the conference. We are grateful to all the invited speakers for having accepted our invitation.

This year the conference was jointly organized with the 12th International Conference on Quantitative Evaluation of Systems (QEST 2015), the 13h International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2015), the 10th International Symposium on Trustworthy Global Computing (TGC 2015), and the International Symposium on Web Services, Formal Methods and Behavioural Types (WS-FM/BEAT 2015). In addition, 'Madrid meet 2015' included seven more satellite events: EPEW 2015: the 12th European Workshop on Performance Engineering; FOCLASA 2015: the 14th International Workshop on Foundations of Coordination Languages and Self-Adaptation; EXPRESS/SOS 2015: Combined 22nd International Workshop on Expressiveness in Concurrency and 12th Workshop on Structured Operational Semantics; HSB 2015: the 4th International Workshop of Hybrid Systems and Biology; TRENDS 2015: the 4th IFIP WG 1.8 Workshop on Trends

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig

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

in Concurrency Theory; PV 2015: the 2nd Workshop on Parameterized Verification; and finally, YR-CONCUR 2015: the Young Researchers Workshop on Concurrency Theory.

For the first time this year CONCUR has an open-access publication of its proceedings, initiating a new collection in LIPIcs. In this way, the Proceedings will be available to everybody, thus giving a greater dissemination to the works presented at the Conference. We are sure that the open publication of the Proceedings will bring a new profitable age to our Conference, as desired by its Steering Committee when leaning toward this new publication outlet. However, we also want to thank Springer Verlag for its great work publishing the Proceedings of the Conference during all these years.

Committees

Programme Committee

Parosh Abdulla

Luca Aceto (co-chair)

Roberto Bruni

Ilaria Castellani

Véronique Cortier

Romain Demangeon

Josée Desharnais

David de Frutos-Escrig (co-chair)

Thomas Hildebrandt

Petr Jančar

Joost-Pieter Katoen

Bartek Klin

Maciej Koutny

Barbara König

Michele Loreti

Bas Luttik

Roland Meyer

Hernán Melgratti

Madhavan Mukund

Jan Rutten

Ana Sokolova

Jiri Srba

Alwen Tiu

Stavros Tripakis

Nikos Tzevelekos

Vasco Vasconcelos

Mahesh Viswanathan

Walter Vogler

Nobuko Yoshida

Lijun Zhang

Steering Committee

Jos Baeten (NL)

Javier Esparza (DE)

Joost-Pieter Katoen (DE)

Kim G. Larsen (DK)

Ugo Montanari (IT)

Scott Smolka (US)

Workshop Chair

Fernando Rosa-Velardo

Organising Committee

Dario Della Monica (Reykjavik Univ.)

Ignacio Fábregas (Reykjavik Univ.)

Carlos Gregorio Rodríguez (UCM)

María Martos-Salgado (UCM)

Yolanda Ortega-Mallén (UCM)

Miguel Palomino (UCM)

Ismael Rodríguez (UCM)

David Romero Hernández (UCM)

Fernando Rosa-Velardo (UCM)

Gustavo Santos-García (Univ. Salamanca)

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Faris Abou-Saleh
Robin Adams
C. Aiswarya
S. Akshay
Mohamed Faouzi Atig
Giorgio Bacci
Giovanni Bacci
Massimo Bartoletti
Henning Basold
Nicolas Basset
Emmanuel Beffara
Francesco Belardinelli
Nikola Benes
Saddek Bensalem
Robin Bergenthum
Simon Bliudze
Stanislav Böhm
Benedikt Bollig
Filippo Bonchi
Luca Bortolussi
Patricia Bouyer
Tomas Brazdil
Ferenc Bujtor
Elias Castegren
Simon Castellan
Souymodip Chakraborty
Taolue Chen
José Manuel Colom
Silvia Crafa
Emanuele D'Osualdo
Mohammad Torabi Dashti
Frank De Boer
Ugo De Liguoro
Simão Melo De Sousa
Søren Debois
Aldric Degorre
Dario Della Monica
Yuxin Deng
Pierre-Malo Denielou
Raymond Devillers
Cinzia Di Giusto
Alessandra Di Pierro
Iulia Dragomir
Derek Dreyer
Parasara Sridhar Duggirala
Constantin Enea
Ignacio Fábregas
Jerome Feret
Nathanaël Fijalkow
Dana Fisman
Vojtěch Forejt
Adrian Francalanza
Lars-Åke Fredlund
Goran Frehse
Hongfei Fu
Hubert Garavel
Paul Gastin
Blaise Genest
Raffaella Gentilini
Elena Giachino
Georgios Giantamidis
Daniele Gorla
Julian Gutierrez
Andreas Haas
Axel Haddad
Matthew Hague
Henri Hansen
Frédéric Herbreteau
Daniel Hirschkoff
Lukas Holik
Andreas Holzer
Florent Jacquemard
Ryszard Janicki
Nils Jansen
Peter Gjøl Jensen
Ayrat Khalimov
Victor Khomenko
Stefan Kiefer
Ines Klimann
Alexander Knapp
Martin Kot
Tomer Kotek
Vasileios Koutavas
K. Narayan Kumar
Orna Kupferman
Salvatore La Torre
Ivan Lanese
Julien Lange
Francois Laroussinie
Ranko Lazic

Anthony Lin
Wanwei Liu
Yang Liu
Kamal Lodaya
Hugo A. López
Gavin Lowe
Etienne Lozes
Jean-Marie Madiot
Nicolas Markey
Francisco Martins
Mieke Massink
Christoph Matheja
Richard Mayr
Ondrej Meca
Artur Meski
Marius Mikučionis
Lukasz Mikulski
Peter Bro Miltersen
Yasuhiko Minamide
Fabrizio Montesi
Dimitris Mostrous
Mohammad Reza Mousavi
Marco Muniz
Aniello Murano
Thomas Noll
Håkon Normann
Ulrik Nyman
Mads Chr. Olesen
Paulo Oliva
Peter Ölveczky
Luca Padovani
Michele Pagani
Miguel Palomino
David Parker
Jorge A. Pérez
Kirstin Peters
Srinivas Pinisetty
Maria Pittou
Danny Bøgsted Poulsen
Damien Pous
M. Praveen
Viorel Preoteasa
Arjun Radhakrishna
Steven Ramsay
Michel Reniers
Ahmed Rezine
Othmane Rezine
Fernando Rosa-Velardo
Philipp Ruemmer
Joshua Sack
Matteo Sammartino
Zdenek Sawa
Gerhard Schellhorn
Alan Schmitt
Sylvain Schmitz
Ali Sezgin
Gautham R. Shenoy
Yasser Shoukry
Pawel Sobocinski
Lei Song
Jari Stenman
Jan Strejcek
Kohei Suenaga
Eijiro Sumii
Alexander J. Summers
Jakob Haahr Taankvist
Luca Tesei
Peter Thiemann
Francesco Tiezzi
Marco Tinacci
Sophie Tison
Ashutosh Trivedi
Andrea Turrini
Irek Ulidowski
Christian Urban
Valentín Valero
Rob Van Glabbeek
Thomas Weidner
Anton Wijs
Tim Willemse
Harro Wimmel
Xiaoxiao Yang
Hans Zantema
Paolo Zuliani

Automatic Application Deployment in the Cloud: from Practice to Theory and Back*

Roberto Di Cosmo[2], Michael Lienhardt[1], Jacopo Mauro[1], Stefano Zacchiroli[2], Gianluigi Zavattaro[1], and Jakub Zwolakowski[2]

1 **University of Bologna/INRIA, Italy**
2 **Université Paris Diderot, Sorbonne Paris Cité, PPS, UMR 7126 CNRS, F-75205 Paris, France**

Abstract

The problem of deploying a complex software application has been formally investigated in previous work by means of the abstract component model named Aeolus. As the problem turned out to be undecidable, simplified versions of the model were investigated in which decidability was restored by introducing limitations on the ways components are described.

In this paper, we take an opposite approach, and investigate the possibility to address a relaxed version of the deployment problem without limiting the expressiveness of the component model. We identify three problems to be solved in sequence: (i) the verification of the existence of a final configuration in which all the constraints imposed by the single components are satisfied, (ii) the generation of a concrete configuration satisfying such constraints, and (iii) the synthesis of a plan to reach such a configuration possibly going through intermediary configurations that violate the non-functional constraints.

1998 ACM Subject Classification D.2.9 Management, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Automatic deployment, Planning, DevOps, Constraint Programming

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.1

Category Invited Paper

1 Introduction

Modern software systems are based on a large number of interconnected software components (e.g., packages or services) that must be deployed on (possibly virtual) machines that can be created and connected on-the-fly by exploiting currently available cloud computing technologies. The configuration and management of such applications is a challenging task, and several tools and technologies are under development to support application architects and managers in this complex activity. The mainstream approach is to exploit pre-configured virtual machines images, which contain all the needed software packages and services, and that just need to be run on the target cloud system (e.g., Bento Boxes [10], Cloud Blueprints [2], or AWS CloudFormation [1], to name just a few options). The main drawback of this approach is that pre-configured images do not support customization and often force users to run their applications on specific cloud providers, inducing an undesirable vendor lock-in effect.

* Partly funded by the EU project FP7-610582 ENVISAGE. Work partially performed at IRILL, center for Free Software Research and Innovation in Paris, France, `http://www.irill.org`. Unless noted otherwise, all URLs in the text have been retrieved on July 1st, 2015.

More advanced techniques, on the contrary, would allow application architects to design their own software architectures by using high level description languages, like the graphical drag-and-drop approach of Juju [12] or the declarative deployment languages ConfSolve [11] and Engage [9].

In previous work [7, 6, 5] we have investigated this deployment problem from a foundational point of view, trying to capture its most relevant aspects, and identifying among them those that contribute to the difficulty of the problem. We have defined a formal model called *Aeolus*, in which the classical notion of component, seen as a black-box that exposes *provide* and *require*-ports, is extended with a finite state automaton describing the component life-cycle. The automaton states correspond to different configuration modalities, like *uninstalled*, *installed*, *running*, *stopped*, etc, and the transitions represent configuration actions like install, run, stop, etc. Depending on the internal state, the ports on the interface can be either active or inactive. For instance, an *uninstalled* component usually does not activate any require-port, while it can activate require-ports when it is in the *installed* state, and finally activate some provide-port when it actually enters the *running* state. Another specific feature of the *Aeolus* model is that capacity constraints can be associated to the ports: a provide-port could have a maximal number of connected require-ports, a require-port can ask for multiple providers offering a given functionality (used to model replication requirements) or even impose that no other component can provide a given functionality (used to model the notion of conflict among components).

The deployment problem is then formalized as the problem of verifying the possibility to configure at least one component of a given type in a given target state, by performing a sequence of actions like component creation/deletion, component binding/unbinding, and component internal state change.

In [7] we have proved that the deployment problem is, in general, undecidable. To overcome this negative result, we have considered in [6, 5] various simplifications of the Aeolus model for which deployment turns out to be decidable. In particular, we have proved that deployment is polynomial when capacity constraints and conflicts are not considered, while it is Ackermann-hard for the fragment without capacity constraints but with conflicts.

In this paper we take a different and more pragmatic approach by relaxing the deployment problem. Conceptually, we break the deployment problem in two independent subtasks to be executed one after the other. The first subtask abstracts away from component states, and considers the problem of computing a final correct configuration in which the target component is present and all the capacity and conflict constraints are satisfied. The second subtask abstracts away from the capacity constraints and the conflicts, and verifies the possibility to reach in this simplified scenario that desired target configuration.

We formalize these subtasks and discuss their complexity. In particular, for the first subtask, we consider two subproblems: the *Configuration* problem, consisting of checking whether it is possible to satisfy all the constraints directly or indirectly imposed by the target component on the final configuration, and the *Generation* problem addressing the concrete production of a configuration that actually satisfies these constraints. The *Configuration* problem is proved to be NP-complete, while the *Generation* problem is EXP-time. It is important to highlight that from a pragmatic point of view, the Configuration problem is much more challenging because the Generation problem has a more standard solution and has a higher theoretical complexity simply because there could be cases (that we consider rare in practice) in which the configuration to be generated is of exponential size. Finally, in a third phase called *Planning* we synthesize, if there exists, a sequence of deployment action to reach the desired final configuration. This problem is poly-time but, but as explained

above the intermediary configurations traversed during the execution of the plan could not satisfy capacity constraints or conflicts because we have decided to loose correctness to favor tractability.

The split of the deployment problem in two subtasks is reflected also by two tools that we have implemented. On the one hand, *Zephyrus* [4] addresses the problem of generating a final configuration that besides satisfying all the constraints that can be expressed in the Aeolus model, also considers the problem of the optimal distribution of the components to be deployed on available virtual machines. The second tool is called *Metis* [13] and it solves the deployment problem, but only for the simplified Aeolus model without capacity constraints and conflicts.

2 The Aeolus model

In this section we give a recap of the *Aeolus component model* following [14]. This formalization differs from other definitions of the *Aeolus model* reported in [7, 5] due to the absence of the so-called *multiple state change* actions. These actions are of interest when components are used to represent mutually dependent packages that must be contemporaneously installed, but are less relevant when components are used to model service deployment and configuration. We have opted for the formalization without multiple state changes as our focus in this paper is mainly on services and not on packages.

In the *Aeolus model*, components are represented as grey boxes offering services via provide-ports and requiring service through require-ports. They are grey boxes because the component configuration life-cycle is exposed in terms of a finite state automaton indicating the different internal configuration states, and the corresponding configuration actions changing such states. When a component changes state, the sets of ports it requires or provides might also change; in other words, the component interface depends on its internal configuration state.

As an example, let us consider Figure 1 depicting the installation of a WordPress blog service according to the *Aeolus model*. Three services are modeled, viz. *wordpress*, *mysql*, and the HTTP daemon service *apache2*. Each service can be in the *uninstalled*, *installed*, or *running* state. Depending on the current state, the require and provide-ports could be active or inactive. For instance, the wordpress component in the installed state activates the require-port httpd, but does not activate the require-port mysql_up and the provide-port wordpress. The httpd require-ports activated by wordpress can be connected to the provide-port offered by apache2 when it is installed or running. Similarly, to be running, it also requires an active mysql service. This is represented by the requirement of the port mysql_up provided by mysql in its running state. The wordpress component in its running state is finally able to provide the wordpress functionality.

We now move to the formal definition of the *Aeolus component model*. Let us assume given the following disjoint sets: $\mathcal{I}$ for interfaces and $\mathcal{Z}$ for components. We use $\mathbb{N}$ to denote natural numbers and $\mathbb{N}^+_\infty$ for $\mathbb{N} \setminus \{0\} \cup \{\infty\}$.

▶ **Definition 1** (Component type). The set Γ of *component types* of the Aeolus model, ranged over by $\mathcal{T}_1, \mathcal{T}_2, \ldots$ contains 5-ples $\langle Q, q_0, T, P, D\rangle$ where:

- Q is a finite set of states;
- $q_0 \in Q$ is the initial state and $T \subseteq Q \times Q$ is the set of *transitions*;
- $P = \langle \mathbf{P}, \mathbf{R}\rangle$, with $\mathbf{P}, \mathbf{R} \subseteq \mathcal{I}$, is a pair composed of the set of *provide* and the set of *require*-ports, respectively;
- D is a function from Q to pairs in $(\mathbf{P} \nrightarrow \mathbb{N}^+_\infty) \times (\mathbf{R} \nrightarrow \mathbb{N})$.

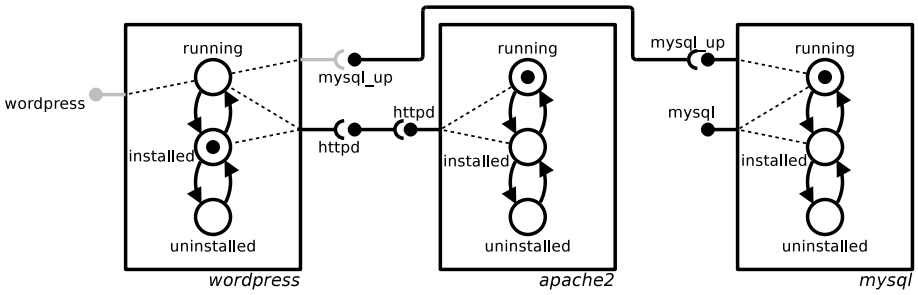


Figure 1 WordPress installation in Aeolus.

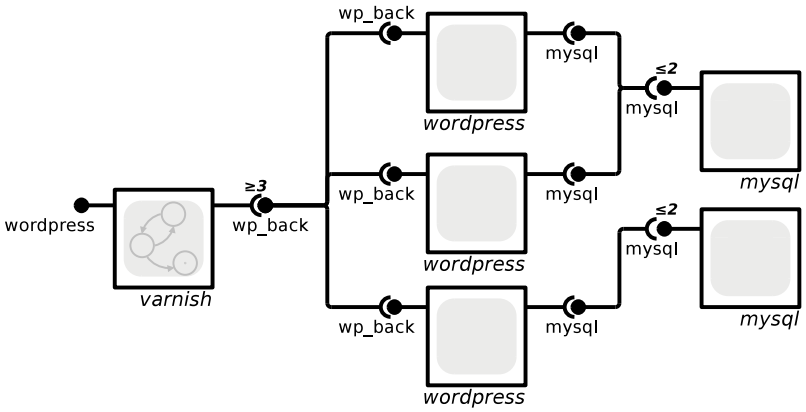


Figure 2 Redundancy and capacity constraints for a complex WordPress installation (internal state machines are omitted for simplicity).

Given a state $q \in Q$, $D(q)$ returns two partial functions $(\mathbf{P} \nrightarrow \mathbb{N}^+_\infty)$ and $(\mathbf{R} \nrightarrow \mathbb{N})$ that indicate respectively the provide and require-ports that q activates. The functions associate to the activate ports a numerical constraint indicating:

- for provide-ports, the *maximum* number of bindings the port can satisfy,
- for require-ports, the *minimum* number of required bindings to *distinct* components,
  - as a special case: if the number is 0 this indicates a conflict, meaning that there should be no other active port, in any other component, with the same name.

When the numerical constraint is not explicitly indicated, we assume as default value ∞ for provide-ports (i.e., they can satisfy an unlimited amount of requires) and 1 for require (i.e., one provide is enough to satisfy the requirement). We also assume that the initial state q_0 makes no demands (i.e., the second function of $D(q_0)$ has an empty domain).

As an example of the use of numerical constraints, Figure 2 shows the modeling of a WordPress hosting scenario where we want to offer high availability by putting the Varnish reverse proxy/load balancer in front of several WordPress instances, all connected to a cluster of MySQL databases.[1] For a configuration to be correct, the model requires that Varnish is connected to at least 3 (active and distinct) WordPress back-ends, and that each MySQL instance does not serve more than 2 clients.

We now define configurations that describe systems composed by component instances and bindings that interconnect them. A configuration, ranged over by $\mathcal{C}_1, \mathcal{C}_2, \ldots$, is given by

[1] All WordPress instances run within distinct Apache instances, which have been omitted for simplicity.

a set of component types, a set of deployed components with a type and an actual state, and a set of bindings. Formally:

▶ **Definition 2** (Configuration). A *configuration* $\mathcal{C}$ is a quadruple $\langle U, Z, S, B\rangle$ where:

- $U \subseteq \Gamma$ is the finite *universe* of all available component types;
- $Z \subseteq \mathcal{Z}$ is the set of the currently deployed *components*;
- S is the component *state description*, i.e., a function that associates to components in Z a pair $\langle \mathcal{T}, q\rangle$ where $\mathcal{T} \in U$ is a component type $\langle Q, q_0, T, P, D\rangle$, and $q \in Q$ is the current component state;
- $B \subseteq \mathcal{I} \times Z \times Z$ is the set of *bindings*, namely 3-ples composed by an interface, the component that requires that interface, and the component that provides it; we assume that the two components are distinct.

In the following we will use a notion of configuration equivalence that relate configurations having the same instances up to renaming. This is used to abstract away from component identifiers and bindings.

▶ **Definition 3** (Configuration equivalence). A configuration $\langle U, Z, S, B\rangle$ is equivalent to $\langle U, Z', S', B'\rangle$, noted $\langle U, Z, S, B\rangle \equiv \langle U, Z', S', B'\rangle$, iff there exists a bijective function ρ from Z to Z' s.t.:

1. $S(z) = S'(\rho(z))$ for every $z \in Z$; and
2. $\langle r, z_1, z_2\rangle \in B$ iff $\langle r, \rho(z_1), \rho(z_2)\rangle \in B'$.

Notation: we write $\mathcal{C}[z]$ as a lookup operation that retrieves the pair $\langle \mathcal{T}, q\rangle = S(z)$, where $\mathcal{C} = \langle U, Z, S, B\rangle$. On such a pair we then use the postfix projection operators `.type` and `.state` to retrieve $\mathcal{T}$ and q, respectively. Similarly, given a component type $\langle Q, q_0, T, \langle \mathbf{P}, \mathbf{R}\rangle, D\rangle$, we use projections to (recursively) decompose it: `.states`, `.init`, and `.trans` return the first three elements; `.prov`, `.req` return $\mathbf{P}$ and $\mathbf{R}$; $.\mathbf{P}(q)$ and $.\mathbf{R}(q)$ return the two elements of the $D(q)$ tuple. When there is no ambiguity we take the liberty to apply the component type projections to $\langle \mathcal{T}, q\rangle$ pairs. For example, $\mathcal{C}[z].\mathbf{R}(q)$ stands for the partial function indicating the active require-ports (and their arities) of component z in configuration $\mathcal{C}$ when it is in state q.

We are now ready to formalize the notion of configuration correctness:

▶ **Definition 4** (Configuration correctness). Let us consider the configuration $\mathcal{C} = \langle U, Z, S, B\rangle$. We write $\mathcal{C} \models_{req} (z, r, n)$ to indicate that the require-port of component z, with interface r, and associated number n is satisfied. Formally, if $n = 0$ all components other than z cannot have an active provide-port with interface r, namely for each $z' \in Z \setminus \{z\}$ such that $\mathcal{C}[z'] = \langle \mathcal{T}', q'\rangle$ we have that r is not in the domain of $\mathcal{T}'.\mathbf{P}(q')$. If $n > 0$ then the port is bound to at least n active ports, i.e., there exist n distinct components $z_1, \ldots, z_n \in Z \setminus \{z\}$ such that for every $1 \leq i \leq n$ we have that $\langle r, z, z_i\rangle \in B$, $\mathcal{C}[z_i] = \langle \mathcal{T}^i, q^i\rangle$ and r is in the domain of $\mathcal{T}^i.\mathbf{P}(q^i)$.

Similarly for provides, we write $\mathcal{C} \models_{prov} (z, p, n)$ to indicate that the provide-port of component z, with interface p, and associated number n is not bound to more than n active ports. Formally, there exist no m distinct components $z_1, \ldots, z_m \in Z \setminus \{z\}$, with $m > n$, such that for every $1 \leq i \leq m$ we have that $\langle p, z_i, z\rangle \in B$, $S(z_i) = \langle \mathcal{T}^i, q^i\rangle$ and p is in the domain of $\mathcal{T}^i.\mathbf{R}(q^i)$.

The configuration $\mathcal{C}$ is *correct* if for each component $z \in Z$, given $S(z) = \langle \mathcal{T}, q\rangle$ with $\mathcal{T} = \langle Q, q_0, T, P, D\rangle$ and $D(q) = \langle \mathcal{P}, \mathcal{R}\rangle$, we have that $(p \mapsto n_p) \in \mathcal{P}$ implies $\mathcal{C} \models_{prov} (z, p, n_p)$, and $(r \mapsto n_r) \in \mathcal{R}$ implies $\mathcal{C} \models_{req} (z, r, n_r)$.

We now formalize how configurations evolve from one state to another, by means of atomic actions:

▶ **Definition 5** (Actions). The set $\mathcal{A}$ contains the following actions:

- $stateChange(z, q_1, q_2)$ where $z \in \mathcal{Z}$: change the state of the component z from q_1 to q_2;
- $bind(r, z_1, z_2)$ where $z_1, z_2 \in \mathcal{Z}$ and $r \in \mathcal{I}$: add a binding between z_1 and z_2 on port r;
- $unbind(r, z_1, z_2)$ where $z_1, z_2 \in \mathcal{Z}$ and $r \in \mathcal{I}$: remove the specified binding;
- $new(z : \mathcal{T})$ where $z \in \mathcal{Z}$ and $\mathcal{T}$ is a component type: add a new component z of type $\mathcal{T}$;
- $del(z)$ where $z \in \mathcal{Z}$: remove the component z from the configuration.

The execution of actions can now be formalized using a labeled transition systems on configurations, which uses actions as labels.

▶ **Definition 6** (Reconfigurations). Reconfigurations are denoted by transitions $\mathcal{C} \xrightarrow{\alpha} \mathcal{C}'$ meaning that the execution of $\alpha \in \mathcal{A}$ on the configuration $\mathcal{C}$ produces a new configuration $\mathcal{C}'$. The transitions from a configuration $\mathcal{C} = \langle U, Z, S, B \rangle$ are defined as follows:

$$\mathcal{C} \xrightarrow{stateChange(z,q_1,q_2)} \langle U, Z, S', B \rangle$$
$$\text{if } \mathcal{C}[z].\texttt{state} = q_1$$
$$\text{and } (q_1, q_2) \in \mathcal{C}[z].\texttt{trans}$$
$$\text{and } S'(z') = \begin{cases} \langle \mathcal{C}[z].\texttt{type}, q_2 \rangle & \text{if } z' = z \\ \mathcal{C}[z'] & \text{otherwise} \end{cases}$$

$$\mathcal{C} \xrightarrow{bind(r,z_1,z_2)} \langle U, Z, S, B \cup \langle r, z_1, z_2 \rangle \rangle$$
$$\text{if } \langle r, z_1, z_2 \rangle \notin B$$
$$\text{and } r \in \mathcal{C}[z_1].\texttt{req} \cap \mathcal{C}[z_2].\texttt{prov}$$

$$\mathcal{C} \xrightarrow{unbind(r,z_1,z_2)} \langle U, Z, S, B \setminus \langle r, z_1, z_2 \rangle \rangle \quad \text{if } \langle r, z_1, z_2 \rangle \in B$$

$$\mathcal{C} \xrightarrow{new(z:\mathcal{T})} \langle U, Z \cup \{z\}, S', B \rangle$$
$$\text{if } z \notin Z,\ \mathcal{T} \in U$$
$$\text{and } S'(z') = \begin{cases} \langle \mathcal{T}, \mathcal{T}.\texttt{init} \rangle & \text{if } z' = z \\ \mathcal{C}[z'] & \text{otherwise} \end{cases}$$

$$\mathcal{C} \xrightarrow{del(z)} \langle U, Z \setminus \{z\}, S', B' \rangle$$
$$\text{if } S'(z') = \begin{cases} \bot & \text{if } z' = z \\ \mathcal{C}[z'] & \text{otherwise} \end{cases}$$
$$\text{and } B' = \{\langle r, z_1, z_2 \rangle \in B \mid z \notin \{z_1, z_2\}\}$$

We can now define a *deployment run*, which is a sequence of actions that transform an initial configuration into a final correct one without violating correctness along the way. A deployment run is the output we expect from a planner, when it is asked how to reach a desired target configuration.

▶ **Definition 7** (Deployment run). A *deployment run* is a sequence $\alpha_1 \dots \alpha_m$ of actions such that there exist $\mathcal{C}_i$ correct configurations such that $\mathcal{C} = \mathcal{C}_0$, $\mathcal{C}_{j-1} \xrightarrow{\alpha_j} \mathcal{C}_j$ for every $j \in \{1, \dots, m\}$.

As an example, the following is a deployment run allowing to reach the configuration depicted in Figure 1 starting from an empty configuration (we use z_1, z_2, and z_3 to identify the wordpress, apache2 and mysql components, respectively):

$new(z_1 : \mathsf{wordpress}), new(z_2 : \mathsf{apache2}), stateChange(z_2, \mathsf{uninstalled}, \mathsf{installed}),$
$bind(\mathsf{httpd}, z_1, z_2), stateChange(z_1, \mathsf{uninstalled}, \mathsf{installed}),$
$new(z_3 : \mathsf{mysql}), stateChange(z_3, \mathsf{uninstalled}, \mathsf{installed}), stateChange(z_3, \mathsf{installed}, \mathsf{running}),$
$bind(\mathsf{mysql\_up}, z_1, z_3)$

We now have all the ingredients to define the notion of *achievability*, that is our main concern: given a universe of component types, we want to know whether it is possible to deploy at least one component of a given component type $\mathcal{T}$ in a given state q.

▶ **Definition 8** (Achievability problem)**.** The *achievability problem* has as input a universe U of component types, a component type $\mathcal{T} \in U$, and a target state q. It returns as output **true** if there exists a deployment run $\alpha_1 \dots \alpha_m$ such that $\langle U, \emptyset, \emptyset, \emptyset \rangle \xrightarrow{\alpha_1} \mathcal{C}_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_m} \mathcal{C}_m$ and $\mathcal{C}_m[z] = \langle \mathcal{T}, q \rangle$, for some component z in $\mathcal{C}_m$. Otherwise, it returns **false**.

In our running example, if we consider the wordpress component in the active state as target, we have that such target is achievable by the deployment run obtained by adding the action $stateChange(z_1, \mathsf{installed}, \mathsf{running})$ to the deployment run described above.

Notice that the restriction in this decision problem to one target component in a given state is not limiting. One can easily encode any given final configuration by adding a dummy provide-port enabled only by the required final states and a dummy component with requirements on all such provides.

In [5] we have proved that the achievability problem is undecidable. In this paper we present a way to relax the achievability problem in order to restore decidability. In particular, we require correctness only for the final configuration, while numerical constraints and conflicts are ignored for the intermediary states traversed during the deployment run. In many cases dealing with service deployment, violating capacity constraints during installation and configuration is not problematic because the services become publicly available only at the end. More precisely, we split the achievability problem in three separate phases: the verification of the existence of a final correct configuration that includes the target component (*Configuration* problem), the synthesis of such a configuration (*Generation* problem), and the computation of a deployment run reaching such a configuration (*Planning* problem). In this last phase, we exploit the efficient poly-time algorithm developed for the simplified *Aeolus* model without numerical constraints and conflicts. For this reason, it could indeed happen that such constraints are violated during the execution of the deployment run.

3 Configuration problem

In this section we deal with the *Configuration* problem, consisting in checking the existence of a correct component configuration including at least one instance of the target component.

The Configuration problem can be viewed as a *Constraint Satisfaction Problem* (CSP). A CSP consists of a finite set of variables, each of which associated with a domain of possible values that it could take, and a set of constraints that defines all the admissible assignments of values to the variables [15]. Given a CSP the goal is normally to find a solution – that is an assignment to the variables that satisfies all the constraints of the problem – through *one* suitable constraint solver.

For the encoding of the Configuration problem into a CSP, we can focus on an abstract representation of a configuration where components of the same type and state are grouped together. Also the specific port bindings are abstracted away: we only capture how many bindings connect the provide-ports with interface p of the components of type $\mathcal{T}$ in state q to require-ports of components of type $\mathcal{T}'$ in state q'. Formally:

▶ **Definition 9** (Abstract Configuration)**.** An abstract configuration $\mathcal{B}$ is a pair of mappings $\langle \texttt{comp}, \texttt{bind} \rangle$ such that:

- $\texttt{comp} : (\Gamma \times Q) \to \mathbb{N}$ associates to every component type $\mathcal{T}$ and one state in $\mathcal{T}.\texttt{states}$ a natural number;
- $\texttt{bind} : \mathcal{I} \times (\Gamma \times Q) \times (\Gamma \times Q) \to \mathbb{N}$ that associates to every port p and couple of component-state pairs a natural number.

▶ **Definition 10** (Concretization)**.** Given an abstract configuration $\mathcal{B} = \langle \texttt{comp}, \texttt{bind} \rangle$ we say that a correct configuration $\mathcal{C} = \langle U, Z, S, B \rangle$ is one concretization of $\mathcal{B}$ if

- the number of components of type $\mathcal{T}$ in state q in the configuration $\mathcal{C}$ is equal to $\texttt{comp}(\langle \mathcal{T}, q \rangle)$ for every type-state pair $\langle \mathcal{T}, q \rangle$
- the number of bindings between the port p provided by components of type $\mathcal{T}$ in state q and required by components of type $\mathcal{T}'$ in state q' is $\texttt{bind}(p, \langle \mathcal{T}, q \rangle, \langle \mathcal{T}', q' \rangle)$.

We write $\gamma(\mathcal{B})$ for the set of concretizations of $\mathcal{B}$.

An abstract configuration $\mathcal{B}$ is *correct* if it has at least one concretization (formally $\gamma(\mathcal{B}) \neq \emptyset$).

Not all possible abstract configurations have concretizations since the number of bindings may violate the capacity constraints and the number of components may violate the conflicts. It is however possible to characterize abstract configurations that always admit a concretization. This can be done by means of the following set of constraints.

$$\bigwedge_{p \in \mathcal{I}} \bigwedge_{\langle \mathcal{T}, q \rangle} \mathcal{T}.\mathbf{R}(q)(p) \times \texttt{comp}(\langle \mathcal{T}, q \rangle) \leq \sum_{\langle \mathcal{T}', q' \rangle} \texttt{bind}(p, \langle \mathcal{T}', q' \rangle, \langle \mathcal{T}, q \rangle) \tag{1a}$$

$$\bigwedge_{p \in \mathcal{I}} \bigwedge_{\langle \mathcal{T}, q \rangle \,.\, \mathcal{T}.\mathbf{P}(q)(p) < \infty} \mathcal{T}.\mathbf{P}(q)(p) \times \texttt{comp}(\langle \mathcal{T}, q \rangle) \geq \sum_{\langle \mathcal{T}', q' \rangle} \texttt{bind}(p, \langle \mathcal{T}, q \rangle, \langle \mathcal{T}', q' \rangle) \tag{1b}$$

$$\bigwedge_{p \in \mathcal{I}} \bigwedge_{\langle \mathcal{T}, q \rangle \,.\, \mathcal{T}.\mathbf{P}(q)(p) = \infty} \texttt{comp}(\langle \mathcal{T}, q \rangle) = 0 \;\Rightarrow\; \sum_{\langle \mathcal{T}', q' \rangle} \texttt{bind}(p, \langle \mathcal{T}, q \rangle, \langle \mathcal{T}', q' \rangle) = 0 \tag{1c}$$

$$\bigwedge_{p \in \mathcal{I}} \bigwedge_{\substack{\langle \mathcal{T}, q \rangle \,.\, \mathcal{T}.\mathbf{R}(q)(p) = 0 \,\wedge \\ \mathcal{T}.\mathbf{P}(q)(p) > 0)}} \texttt{comp}(\langle \mathcal{T}, q \rangle) \leq 1 \tag{1d}$$

$$\bigwedge_{p \in \mathcal{I}} \bigwedge_{\substack{\langle \mathcal{T}, q \rangle . \\ \mathcal{T}.\mathbf{R}(q)(p) = 0}} \bigwedge_{\substack{\langle \mathcal{T}', q' \rangle \neq \langle \mathcal{T}, q \rangle \,. \\ \mathcal{T}'.\mathbf{P}(q')(p) > 0}} \texttt{comp}(\langle \mathcal{T}, q \rangle) > 0 \;\Rightarrow\; \texttt{comp}(\langle \mathcal{T}', q' \rangle) = 0 \tag{1e}$$

$$\bigwedge_{p \in \mathcal{I}} \bigwedge_{\langle \mathcal{T}, q \rangle} \bigwedge_{\langle \mathcal{T}', q' \rangle \neq \langle \mathcal{T}, q \rangle} \texttt{bind}(p, \langle \mathcal{T}, q \rangle, \langle \mathcal{T}', q' \rangle) \leq \texttt{comp}(\langle \mathcal{T}, q \rangle) \times \texttt{comp}(\langle \mathcal{T}', q' \rangle) \tag{1f}$$

$$\bigwedge_{p \in \mathcal{I}} \bigwedge_{\langle \mathcal{T}, q \rangle} \texttt{bind}(p, \langle \mathcal{T}, q \rangle, \langle \mathcal{T}, q \rangle) \leq \texttt{comp}(\langle \mathcal{T}, q \rangle) \times (\texttt{comp}(\langle \mathcal{T}, q \rangle) - 1) \tag{1g}$$

Constraint 1a enforces the fact that the number of bindings connected to the require-ports p of components of type-state pair $\langle \mathcal{T}, q \rangle$ cannot be smaller than the total requirements computed as the sum of the single requirements of each instance of type-state $\langle \mathcal{T}, q \rangle$. Symmetrically, constraint 1b guarantees that the number of bindings connected to the provide-ports p of

components of type-state pair $\langle \mathcal{T}, q \rangle$ cannot be greater than the total available capacity computed as the sum of the single capacities of each instance of type-state $\langle \mathcal{T}, q \rangle$. In case the port capacity is unbounded (i.e., ∞), it is sufficient to have at least one instance that activates such port to support any possible requirement (see constraint 1c). Constraints 1d and 1e deal with conflicts. In particular the constraint 1d limits to at most one the instances of component type-state pairs that can simultaneously provide and being in conflict with a given port. Constraint 1e enforces instead that when a conflict is active, then no other instance providing the same port exists. Finally, the constraints 1f and 1g guarantee that there are enough pairs of distinct instances to establish all the necessary bindings. Two distinct constraints are used: the first one deals with bindings between components of two different type-state pair, the second one considers those bindings that are established between two components of the same type-state pair.

▶ **Lemma 11.** *An abstract configuration $\mathcal{B}$ satisfies the constraints 1 iff it is correct.*

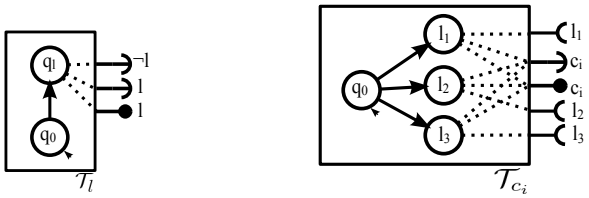
Proof. Suppose that $\mathcal{C}$ is a concretization of the abstract configuration $\mathcal{B}$. Now suppose that one among constraints 1 is violated.

- If 1a is violated, in $\mathcal{C}$ there exists a component of type $\mathcal{T}$ in state q requiring port p and having less than $\mathcal{T}.\mathbf{R}(q)(p)$ bindings. By the correctness of $\mathcal{C}$ this is impossible.
- If 1b is violated, in $\mathcal{C}$ there exists a component of type $\mathcal{T}$ in state q that has more than $\mathcal{T}.\mathbf{P}(q)(p)$ bindings to port p. This is impossible since by definition $\mathcal{C}$ is correct.
- If 1c is violated, in $\mathcal{C}$ there should be bindings connected to components of type-state pairs that do not appear in the configuration. By the correctness of $\mathcal{C}$ this is impossible.
- If 1d is violated, then there are two instances of a component of type $\mathcal{T}$ in state q that simultaneously provide and are in conflict with the port p. By the correctness of $\mathcal{C}$ this is impossible.
- If 1e is violated, then there is an instance of a component that is in conflict with a port p and a different instance that provides p. By the correctness of $\mathcal{C}$ this is impossible.
- If 1f is violated, then there exist a port p and two components of different type-state pair such that there exists two bindings connecting the port p between them. By the correctness of $\mathcal{C}$ this is impossible.
- If 1g is violated, then there exist a port p and either a component activating a provide and a require-port p which are connected, or two components of the same type-state pair with two bindings connecting the port p between them. By the correctness of $\mathcal{C}$ this is impossible.

Therefore, if $\mathcal{B}$ is correct then all of constraints 1 are satisfied.

Now let us suppose that there exists an abstract configuration $\mathcal{B}\langle \texttt{comp}, \texttt{bind} \rangle$ that satisfies the constraints 1. We show the existence of a concretization $\mathcal{C}$, such that for every component type-state pair $\langle \mathcal{T}, q \rangle$ it has $\texttt{comp}(\langle \mathcal{T}, q \rangle)$ different instances of type $\mathcal{T}$ in state q. Conflicts cannot happen between the components, otherwise constraints 1d or 1e would be violated. We now discuss the bindings in $\mathcal{C}$. From constraint 1f we have that it is possible to have a number of bindings $\texttt{bind}(p, \langle \mathcal{T}, q \rangle, \langle \mathcal{T}', q' \rangle)$ between distinct pairs of instances of type-state $\langle \mathcal{T}, q \rangle$ and $\langle \mathcal{T}', q' \rangle$ respectively providing and requiring port p; moreover, from constraint 1g we have that it is possible to have a number of bindings $\texttt{bind}(p, \langle \mathcal{T}, q \rangle, \langle \mathcal{T}, q \rangle)$ between distinct instances of type-state $\langle \mathcal{T}, q \rangle$ providing and requiring port p. It remains to show that there exists at least one distribution of these bindings that satisfies the capacity constraints on all the provide and require-ports.

Assume, by contraposition, that there exists no distribution of bindings that satisfies the capacity constraints. This means that for every possible distribution there are always



(a) Literal encoding. **(b)** Clause encoding.

Figure 3 3-SAT encoding into Aeolus.

provide-ports with a number of bindings greater than the capacity, or require-ports with a number of bindings smaller than those that are required. We call *discrepancy* the overall number of excessive bindings connected to provide-ports plus the number of missing bindings in require-ports. We consider one of the binding distributions with minimal discrepancy. It is not restrictive to assume that there is a component r_1 of type-state $\langle \mathcal{T}, q \rangle$ with a provide-port p having an excessive number of bindings. This assumption is not restrictive because if this is not the case, there is at least one require-port with an insufficient number of bindings, and the following reasoning can be symmetrically applied. By constraint 1b there is another instance r_2 of type-state $\langle \mathcal{T}, q \rangle$ that can host at least one additional binding on its provide-port p without exceeding its capacity. The idea is to rebind one of the bindings on the provide-port p of r_1 to the provide-port p of r_2. This can be done because there exist at least two components connected to the port p of r_1 which are not already connected to r_2 (the port of r_1 strictly exceeds its capacity while that of r_2 can host at least one additional binding). Among these two components, at least one component (let us call it r) is different from r_2. It is safe to rebind the require-port p of r from r_1 to r_2 because $r \neq r_2$ and r is not already connected to r_2. Upon rebinding, in the new configuration, discrepancy is strictly reduced thus contradicting minimality. ◀

In the light of Lemma 11, in order to check if there is a correct configuration containing the target component type-state pair $\langle \mathcal{T}_t, q_t \rangle$, we can simply check if there is a correct abstract configuration where $\texttt{comp}(\langle \mathcal{T}_t, q_t \rangle) > 0$. This can be done by solving a CSP problem where the functions `comp` and `bind` are defined by means of a set of variables having as domain $\mathbb{N}$ and by enforcing the constraints 1 (besides the constraint $\texttt{comp}(\langle \mathcal{T}_t, q_t \rangle) > 0$).

Thanks to this encoding it is possible to prove that the Configuration problem is *NP*-complete.

▶ **Theorem 12.** *The Configuration problem is NP-complete.*

Proof. To prove the *NP*-hardness we reduce the 3-SAT Problem into Configuration.

As depicted in Figure 3 a literal l of a 3-SAT formula φ is encoded into a component type $\mathcal{T}_l$ having, beyond an initial state, a final state q_l that provides a port l, is in conflict with the same port l and with its negation $\neg l$. A clause $c_i = l_1 \vee l_2 \vee l_3$ is encoded into a component type $\mathcal{T}_{c_i}$ having, beyond an initial state, three states l_j that require the corresponding port l_j, provide the port c_i and are in conflict with it. The target component type $\mathcal{T}_t$ is a two state component where the final state q_t requires for every clause c_i the port c_i.

The conflict ports impose that every correct configuration has at most one instance of type $\mathcal{T}_l$ or $\mathcal{T}_{\neg l}$. Similarly, at most one instance of $\mathcal{T}_{c_i}$ could be present. It is easy to see that a formula φ is satisfiable iff there exists a configuration where an instance of $\mathcal{T}_t$ is present in the state q_t. Indeed, if φ is satisfiable then it is possible to consider a configuration having

an instance of $\mathcal{T}_l$ for every literal l assigned to true. By validity of φ under the considered literal assignments, it is also possible to bind all the clause components $\mathcal{T}_{c_i}$ to at least one corresponding literal, and then all the $\mathcal{T}_{c_i}$ instances to the target component $\mathcal{T}_t$ in state q_t. Similarly, if there exists a configuration where $\mathcal{T}_t$ in state q_t is present we can assign to true every literal l_i such that an instance of $\mathcal{T}_l$ is present in the configuration and the remaining literals to false. This assignment satisfies the formula φ.

We now have to prove that Configuration is in NP. To this aim we encode Configuration into an Integer Linear Programming (ILP) problem, a well-known problem in NP [17].

The first problem to resort to ILP is that the constraints 1f and 1g are not linear since there are multiplications between two variables: $\texttt{comp}(\langle\mathcal{T},q\rangle) \times \texttt{comp}(\langle\mathcal{T}',q'\rangle)$ in constraint 1f and $\texttt{comp}(\langle\mathcal{T},q\rangle) \times (\texttt{comp}(\langle\mathcal{T},q\rangle) - 1)$ in constraint 1g. Luckily, these constraints can be reformulated into a polynomial number of linear constraints. We descrive in details how to reformulate constraint 1f; constraint 1g can be translated similarly.

The basic observation is that in a correct configuration there is no need to have more bindings than those strictly needed to satisfy the require-ports. In the light of this observation, it is safe to assume that the number of bindings $\texttt{bind}(p,\langle\mathcal{T},q\rangle,\langle\mathcal{T}',q'\rangle)$ is less than $\texttt{comp}(\langle\mathcal{T}',q'\rangle) \times \mathcal{T}'.\mathbf{R}(q')(p)$ which is the number of bindings required to satisfy the require-ports p of the instances of type-state $\langle\mathcal{T}',q'\rangle$. Hence we can consider the disequation $\texttt{bind}(p,\langle\mathcal{T},q\rangle,\langle\mathcal{T}',q'\rangle) \leq \texttt{comp}(\langle\mathcal{T},q\rangle) \times \texttt{comp}(\langle\mathcal{T}',q'\rangle)$ in 1f only in those cases in which $\texttt{comp}(\langle\mathcal{T},q\rangle) < \mathcal{T}'.\mathbf{R}(q')(p)$, and for all the other cases consider the disequation $\texttt{bind}(p,\langle\mathcal{T},q\rangle,\langle\mathcal{T}',q'\rangle) \leq \mathcal{T}'.\mathbf{R}(q')(p) \times \texttt{comp}(\langle\mathcal{T}',q'\rangle)$ without variable multiplication.

We now discuss how to transform $\texttt{bind}(p,\langle\mathcal{T},q\rangle,\langle\mathcal{T}',q'\rangle) \leq \texttt{comp}(\langle\mathcal{T},q\rangle) \times \texttt{comp}(\langle\mathcal{T}',q'\rangle)$ under the assumption that $\texttt{comp}(\langle\mathcal{T},q\rangle) < \mathcal{T}'.\mathbf{R}(q')(p)$. Let $n = \lfloor\log(\mathcal{T}'.\mathbf{R}(q')(p))\rfloor$. We can consider $2n$ variables: $x_1,\cdots,x_n$ which are the n digits of the binary representation of $\texttt{comp}(\langle\mathcal{T},q\rangle)$, and $y_1,\cdots,y_n$ which coincides with 0 for those indexes i for which x_i is 0, and $2^i \times \texttt{comp}(\langle\mathcal{T}',q'\rangle)$ for those indexes i for which x_i is 1. In this way, the sum of all the variables $y_1,\cdots,y_n$ coincides with $\texttt{comp}(\langle\mathcal{T},q\rangle) \times \texttt{comp}(\langle\mathcal{T}',q'\rangle)$, and this value is obtained without exploiting variable multiplication, at the price of adding only a polynomial number of variables.

Summarizing, the new constraints replacing the constraint 1f are as follows:

$$\texttt{bind}(p,\langle\mathcal{T},q\rangle,\langle\mathcal{T}',q'\rangle) \leq \mathcal{T}'.\mathbf{R}(q')(p) \times \texttt{comp}(\langle\mathcal{T}',q'\rangle) \tag{2a}$$

$$\forall i \in [1,n]\,.\,0 \leq x_i \leq 1 \qquad \forall i \in [1,n]\,.\,0 \leq y_i \leq 1 \tag{2b}$$

$$\texttt{comp}(\langle\mathcal{T},q\rangle) < \mathcal{T}'.\mathbf{R}(q')(p) \Rightarrow \sum_{i=0}^{n} 2^i x_i = \texttt{comp}(\langle\mathcal{T},q\rangle) \tag{2c}$$

$$\forall i \in [1,n]\,.\,x_i = 1 \Rightarrow y_i = 2^i \times \texttt{comp}(\langle\mathcal{T}',q'\rangle) \qquad \forall i \in [1,n]\,.\,x_i = 0 \Rightarrow y_i = 0 \tag{2d}$$

$$\texttt{comp}(\langle\mathcal{T},q\rangle) < \mathcal{T}'.\mathbf{R}(q')(p) \Rightarrow \texttt{bind}(p,\langle\mathcal{T},q\rangle,\langle\mathcal{T}',q'\rangle) \leq \sum_{i=0}^{n} y_i \tag{2e}$$

The first constraint binds the number of bindings to satisfy all the require-ports as explained before. The remaining constraints encode precisely the constraint 1f for the cases when $\texttt{comp}(\langle\mathcal{T},q\rangle)$ is smaller than $\mathcal{T}'.\mathbf{R}(q')(p)$. To do so, $5n+2$ linear constraints are used introducing $2n$ fresh variables (i.e., x_i and y_i) which are used to compute $\texttt{comp}(\langle\mathcal{T},q\rangle) \times \texttt{comp}(\langle\mathcal{T}',q'\rangle)$ without variable multiplication as described above.

Applying this transformation, the Configuration problem can be checked solving a polynomial number of linear constraints. However, some of these constraints are logical implication that have to be compiled into the linear (dis)equations of an ILP problem. This is possible exploiting a result from disjunctive programming [18] that indicates how to encode with the addition of a polynomial amount of variable and (dis)equations the

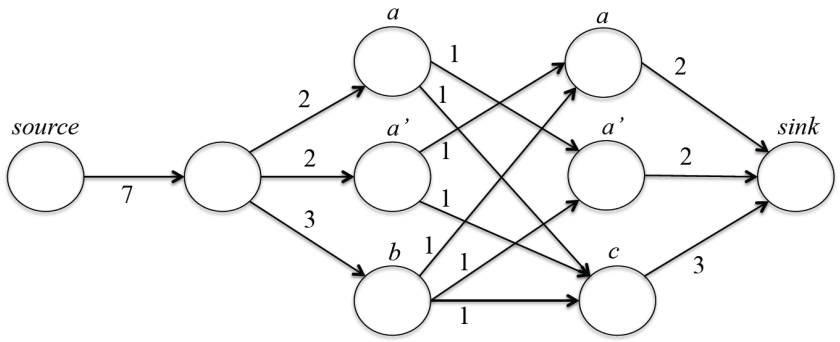


Figure 4 Example of max-flow graph.

requirement that at least one in a set of (dis)equations holds. For instance, the logical implication in the constraint 2c holds if at least one of $\texttt{comp}(\langle \mathcal{T}, q \rangle) \geq \mathcal{T}'.\mathbf{R}(q')(p)$ or $\sum_{i=0}^{n} 2^i x_i = \texttt{comp}(\langle \mathcal{T}, q \rangle)$ holds. The Configuration problem can be therefore mapped into an ILP with a polynomial number of constraints and variables. Since ILP is in NP [17] this also holds for Configuration. ◀

4 Configuration generation

As previously shown, by using a constraint solver it is possible to compute an abstract configuration such that its concretizations have an instance of the target component type in the target state. Thanks to the fact that the Configuration problem is NP-complete, we have that the size of a representation of the abstract configuration, if any, is polynomially bounded. But, in the worst case, the generation of a concretization could require an exponential amount of time since the number of the components could be exponential w.r.t. the size of the abstract configuration representation. For instance, an abstract configuration of $O(\log(n))$ space could require the creation of n components but, to concretely represent the n instances, $O(n)$ space is needed.

In the proof of the following theorem we show how to generate, starting from the target abstract configuration, a correct configuration which is equivalent to one of its concretization. Hence, this configuration will contain the target component in the target state. If we denote with Generation the problem of computing a configuration equivalent to a concretization of the target abstract configuration, we have the following theorem.

▶ **Theorem 13.** *The Generation problem is EXP-time.*

Proof. The first step to solve the problem is to solve the Configuration problem to obtain an abstract configuration and then to compute one configuration that is equivalent to one of its concretizations. Solving a Configuration problem is NP-complete and therefore can be done in EXP-time.

Starting from the abstract configuration $\mathcal{B} = \langle \texttt{comp}, \texttt{bind} \rangle$ it is possible to generate for every component-type/state pair $\langle \mathcal{T}, q \rangle$ exactly $\texttt{comp}(\langle \mathcal{T}, q \rangle)$ component of type $\mathcal{T}$ in state q. This can take an exponential time w.r.t. the size of the input since $\texttt{comp}(\langle \mathcal{T}, q \rangle)$ can be stored in space $O(\log(\texttt{comp}(\langle \mathcal{T}, q \rangle)))$.

The binding generation can be done solving a maximal flow problem. Given a port p it is possible to consider the graph where the nodes represent the component instances providing p (provider) and the component instances requiring p (requirer), and an edge with capacity 1

exists from every provider to every distinct requirer. Every provider node representing an instance of type $\mathcal{T}$ in state q has an incoming edge from an auxiliary node with capacity equal to its provide numerical constraint $\mathcal{T}.\mathbf{P}(q)(p)$. This auxiliary node is connected to a source node with an edge that has a capacity equal to the sum of the bindings connected to the provide-port p on all the components of any type $\sum_{\langle\mathcal{T},q\rangle,\langle\mathcal{T}',q'\rangle} \texttt{bind}(p,\langle\mathcal{T},q\rangle,\langle\mathcal{T}',q'\rangle))$. On the other hand, every requirer node is connected to a sink node with an edge having capacity equal to its require numerical constraint $\mathcal{T}.\mathbf{R}(q)(p)$.

As an example, Figure 4 shows the max-flow graph obtained when there are two kinds of provider of type-state $\langle\mathcal{T}_a,q_a\rangle$ and $\langle\mathcal{T}_b,q_b\rangle$, and two kinds of requirer of type-state $\langle\mathcal{T}_a,q_a\rangle$ and $\langle\mathcal{T}_c,q_c\rangle$. Assuming that two instances of type-state $\langle\mathcal{T}_a,q_a\rangle$ are in the configuration, and one instance for the other two type-states, three nodes a, a' and b are considered as provider and a, a' and c as requirer. Components $\mathcal{T}_a$ in state q_a provide the port p with a constraint of 2, while components of type $\mathcal{T}_b$ in state q_b provide p with a numerical constraint of 3. Components $\mathcal{T}_a$ in state q_a requires at least 2 bindings on its p port, while those of type $\mathcal{T}_c$ in state q_c requires 3 bindings. The solution depicted in the Figure shows the graph when it is required that 7 bindings on ports p should be established.

The maximal flow of this graph is $\sum_{\langle\mathcal{T},q\rangle,\langle\mathcal{T}',q'\rangle} \texttt{bind}(p,\langle\mathcal{T}',q'\rangle,\langle\mathcal{T},q\rangle)$ and the edges between the provider and requirer selected by the maximal flow algorithm correspond to the bindings to create in the configuration. The configuration that can be obtained in this way is equivalent to any of the concretizations of the given abstract configuration.

Since the maximal flow can be computed by the Ford–Fulkerson algorithm [3] in $O(Ef)$, where E is the number of edges and f is the flow, and the number of components is exponential on the size of the input we have that the generation of the bindings is in EXP-time. ◀

5 Plan generation

In [14] we have presented an algorithm for solving the achievability problem under the assumption that the numerical constraints associated to require-ports are always equal to 1, and those associated to provide-ports are always ∞. This means that no conflicts can be expressed, as well as redundancy requirements on require-ports or maximal capacities on provide-ports. The algorithm runs in polynomial time, and is also capable of returning a corresponding deployment run when it exists (or an empty sequence when it does not). We have also realized a prototype called Metis [13] that implements this algorithm: in the following we call $\mathit{Metis}(U,\langle\mathcal{T}_t,q_t\rangle)$ the deployment run returned by our algorithm when executed on the universe of components U and target component type-state $\langle\mathcal{T}_t,q_t\rangle$.

Given a target configuration $\mathcal{C}$, we show how to exploit Metis in order to generate a sequence of action that reaches the final configuration $\mathcal{C}$. As Metis does not take into account the numerical constraints on component ports, it could happen that the intermediary configurations traversed during the execution of the returned sequence of action are not correct. Nevertheless, we guarantee that the finally reached configuration $\mathcal{C}$ is correct. We call Planning this specific problem of generating a sequence of actions that reaches the target configuration $\mathcal{C}$.

The idea is to start from the configuration $\mathcal{C}$ and generate a universe of component types $U_{\mathcal{C}}$, that extends the original universe U with new component types $\mathcal{T}_z$ – one for each instance z in $\mathcal{C}$ – plus a specific target type $\mathcal{T}_{\mathcal{C}}$ with a target state $q_{\mathcal{C}}$, such that the sequence of actions computed by $\mathit{Metis}(U_{\mathcal{C}},\langle\mathcal{T}_{\mathcal{C}},q_{\mathcal{C}}\rangle)$ can be post-processed to obtain the desired solution to the Planning problem.

We start by presenting the generation of $U_{\mathcal{C}}$. For every component z of type $\mathcal{T}$ in state q

in the configuration $\mathcal{C}$, a new type of component $\mathcal{T}_z$ is added to the original universe. The new type $\mathcal{T}_z$ is equal to $\mathcal{T}$ with the following exceptions:

- every port p provided by $\mathcal{T}$ in state q is replaced by p_z;
- every port r required by $\mathcal{T}$ in state q is replaced with ports $\{r_{z'} \mid \langle r, z', z\rangle \in B\}$ where B is the set of bindings of $\mathcal{C}$ (i.e., every require-port is replaced by ports depending on the component that provides that port);
- the state q provides the port z.

The universe will include also the new target component type $\mathcal{T}_\mathcal{C}$: it has two states, the initial one and a connected state $q_\mathcal{C}$ that requires the port z for every component z in $\mathcal{C}$.

Given these definitions, it is immediate to see that $Metis(U_\mathcal{C}, \langle \mathcal{T}_\mathcal{C}, q_\mathcal{C}\rangle)$ will return either an empty sequence or a sequence of actions leading to a configuration that includes the components and bindings in $\mathcal{C}$ plus other components, like the target dummy components of type $\mathcal{T}_\mathcal{C}$ or other temporary components. By temporary components, we mean instances of components necessary during the execution of the deployment actions (e.g., to satisfy some requirements in intermediary states to be traversed to reach the final ones) but not present in the desired configuration $\mathcal{C}$. These temporary components are easily identifiable in the configuration reached by the synthesized plan because are not connected to the target component of type $\mathcal{T}_\mathcal{C}$. These additional components can be removed by adding to the plan the corresponding *del* actions.

We now discuss how to post-process the plan possibly generated by Metis. First of all, it is necessary to replace the bind actions $bind(r_x, z_1, z_2)$ with $bind(r, z_1, z_2)$ and deleting all the actions involving components of type $\mathcal{T}_\mathcal{C}$, to obtain a sequence of actions reaching a configuration equivalent to $\mathcal{C}$ up-to the additional temporary components (the target $\mathcal{T}_\mathcal{C}$ is not created). Then *del* actions are added to the plan for all the temporary components as described above.

▶ **Theorem 14.** *The Planning problem can be solved in polynomial time.*

Proof. The polynomiality of the algorithm described above derives directly from the polynomiality of Metis [14]. Indeed, the extended universe U' contains only ad additional linear number of component types w.r.t. the size of the input, and the post-processing of the sequence of actions performs a scan of the plan generated by Metis and add some *del* action. This scan as well as the addition of *del* actions can be done in polynomial time since the polynomiality of Metis bounds the plan and the reached configuration to be polynomial. ◀

As a consequence of this result we have that the chain of algorithms solving in sequence the Configuration, Generation, and Planning problems, compute a sequence of deployment actions to reach the desired target configuration in EXP-Time. From the practical point of view this complexity is however reached only when there are components that require a large number of other components to satisfy their needs. Indeed, in this case the space needed to store the port capacity may take $O(n)$ space requiring $O(2^n)$ deployment action to generate the providers.

6 Related work

With the current popularity of cloud computing, the problem of automating application deployment has recently attracted a lot of attention. As of today most industrial products offered by big companies, such as Amazon, HP and IBM, rely on holistic approaches where a complete model for the entire application is defined and the deployment plan is derived in a top-down manner. In this context, one prominent work is represented by the TOSCA

(Topology and Orchestration Specification for Cloud Applications) standard [16], promoted by the OASIS consortium for open standards. TOSCA proposes an XML-like (or YAML) rich language to describe an application. Deployment plans are usually BPEL notations, i.e., work-flow defined in the context of business process modeling.

Using these approaches the burden of specifying what components should be deployed and how to interconnect them is left to system administrators or cloud engineers. On the contrary, Zephyrus [4] and ConfSolve [11] automatize also this task starting from a high-level declarative specification of the desired configuration. As previously mentioned, Zephyrus is grounded on the Aeolus model and takes into account also packages, repositories, as well as the optimality of the proposed solution. ConfSolve relies on constraint solving techniques to propose an optimal allocation of virtual machines to servers, and of applications to virtual machines. An object-oriented declarative language is used to describe the entities (e.g., machines and services), the constraints, and the optimization criteria. Similarly to Zephyrus, but differently from Metis, ConfSolve does not consider the problem of synthesizing a low-level plan to reach the final configuration.

Two recent efforts, Feinerer's work on UML [8] and Engage [9], are more similar to our approach as they both rely on a solver to plan deployments. Feinerer's work is based on the UML component model, which includes conflicts and dependencies, but lacks the aspects concerning virtual machines and deployment. Engage, on the other hand, offers no support for conflicts in the specification language. Neither Feinerer's work nor Engage allows to find a deployment that uses resources in an optimal way, minimizing the number and cost of needed components as can be obtained by Zephyrus. Furthermore, no other tool that we are aware of allows to declare capacity or replication constraints, which are essential non functional constraints for any non-trivial, scalable application.

7 Conclusion

In this article we have studied the problem of reconfiguration using the abstract component model named Aeolus, with all its expressive power. This model allows to describe complex component systems with functional constraints, and non-functional constraints like redundancy, capacity and conflicts.

We have carefully decomposed the reconfiguration problem into three steps, Configuration, Generation and Planning, and we showed how to recover decidability by imposing restrictions only on the transient states of the Planning phase.

These restrictions correspond to the realistic deployment conditions of many current distributed applications, thus paving the way to a new generation of smarter deployment tools that will help automate a larger part of the work that today requires significant human intervention. A research prototype, described in [4], has been developed to show the viability of this approach.

References

1 Amazon. AWS CloudFormation. `http://aws.amazon.com/cloudformation/`.

2 CenturyLink. Cloud Blueprints. `https://www.centurylinkcloud.com/blueprints/`.

3 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.

4 Roberto Di Cosmo, Michael Lienhardt, Ralf Treinen, Stefano Zacchiroli, Jakub Zwolakowski, Antoine Eiche, and Alexis Agahi. Automated synthesis and deployment of cloud applications. In *ASE*, 2014.

5 Roberto Di Cosmo, Jacopo Mauro, Stefano Zacchiroli, and Gianluigi Zavattaro. Aeolus: A component model for the cloud. *Inf. Comput.*, 239:100–121, 2014.

6 Roberto Di Cosmo, Jacopo Mauro, Stefano Zacchiroli, and Gianluigi Zavattaro. Component Reconfiguration in the Presence of Conflicts. In *ICALP 2013: 40th International Colloquium on Automata, Languages and Programming*, volume 7966 of *LNCS*, 2013.

7 Roberto Di Cosmo, Stefano Zacchiroli, and Gianluigi Zavattaro. Towards a Formal Component Model for the Cloud. In *SEFM 2012*, volume 7504 of *LNCS*, 2012.

8 Ingo Feinerer. Efficient large-scale configuration via integer linear programming. *AI EDAM*, 27(1):37–49, 2013.

9 Jeffrey Fischer, Rupak Majumdar, and Shahram Esmaeilsabzali. Engage: a deployment management system. In *PLDI*, 2012.

10 Flexiant. Bento Boxes. `http://www.flexiant.com/2012/12/03/application-provisioning/`.

11 John A. Hewson, Paul Anderson, and Andrew D. Gordon. A Declarative Approach to Automated Configuration. In *LISA*, 2012.

12 Juju, DevOps Distilled. `https://jujucharms.com/`.

13 Tudor A. Lascu, Jacopo Mauro, and Gianluigi Zavattaro. A Planning Tool Supporting the Deployment of Cloud Applications. In *ICTAI*, 2013.

14 Tudor A. Lascu, Jacopo Mauro, and Gianluigi Zavattaro. Automatic Component Deployment in the Presence of Circular Dependencies. In *FACS*, 2013.

15 Alan K. Mackworth. Consistency in Networks of Relations. *Artif. Intell.*, 8(1):99–118, 1977.

16 OASIS. Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0. `http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.html`.

17 Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981.

18 Juan Pablo Vielma and George L. Nemhauser. Modeling disjunctive constraints with a logarithmic number of binary variables and constraints. *Math. Program.*, 128(1-2):49–72, 2011.

Reachability Problems for Continuous Linear Dynamical Systems*

James Worrell

Department of Computer Science, University of Oxford
Parks Road, Oxford OX1 3QD, UK
james.worrell@cs.ox.ac.uk

Abstract

It is well understood that the interaction between discrete and continuous dynamics makes hybrid automata difficult to analyse algorithmically. However it is already the case that many natural verification questions concerning only the continuous dynamics of such systems are extremely challenging. This remains so even for linear dynamical systems, such as linear hybrid automata and continuous-time Markov chains, whose evolution is detemined by linear differential equations. For example, one can ask to decide whether it is possible to escape a particular location of a linear hybrid automaton, given initial values of the continuous variables. Likewise one can ask whether a given set of probability distributions is reachable during the evolution of continuous-time Markov chain.

This talk focusses on reachability problems for solutions of linear differential equations. A central decision problem in this area is the Continuous Skolem Problem, which asks whether a real-valued function satisfying an ordinary linear differential equation has a zero. This can be seen as a continuous analog of the Skolem Problem for linear recurrence sequences, which asks whether the sequence satisfying a given recurrence has a zero term. For both the discrete and continuous versions of the Skolem Problem, decidability is open.

We show that the Continuous Skolem Problem lies at the heart of many natural verification questions on linear dynamical systems. We describe some recent work, done in collaboration with Chonev and Ouaknine, that uses results in transcendence theory and real algebraic geometry to obtain decidability for certain variants of the problem. In particular, we consider a bounded version of the Continuous Skolem Problem, corresponding to time-bounded reachability. We prove decidability of the bounded problem assuming Schanuel's conjecture, one of the main conjectures in transcendence theory. We describe some partial decidability results in the unbounded case and discuss mathematical obstacles to proving decidability of the Continuous Skolem Problem in full generality.

1998 ACM Subject Classification G.1.7 Ordinary Differential Equations

Keywords and phrases Linear differential Equations, Hybrid Automata, Schanuel's Conjecture

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.17

Category Invited Paper

* This work was partially supported by the EPSRC.

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 17–17

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

Notions of Conformance Testing for Cyber-Physical Systems: Overview and Roadmap*

Narges Khakpour[1] and Mohammad Reza Mousavi[2]

1 **Department of Computer Science**
Linnaeus University, Sweden
narges.khakpour@lnu.se

2 **Centre for Research on Embedded Systems (CERES)**
School of Information Technology
Halmstad University, Sweden
m.r.mousavi@hh.se

Abstract

We review and compare three notions of conformance testing for cyber-physical systems. We begin with a review of their underlying semantic models and present conformance-preserving translations between them. We identify the differences in the underlying semantic models and the various design decisions that lead to these substantially different notions of conformance testing. Learning from this exercise, we reflect upon the challenges in designing an "ideal" notion of conformance for cyber-physical systems and sketch a roadmap of future research in this domain.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Cyber-physical systems, hybrid systems, conformance testing, model-based testing, behavioral pre-orders, hybrid input-output conformance testing, (τ, ε) conformance, approximate simulation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.18

Category Invited Paper

1 Introduction

Cyber-physical systems (CPSs) are the result of the massive and tight interaction of computer systems with physical components and with each other. CPSs have gained importance due to the opportunities that they provide and the criticality of the applications in which they are used. Concerning this importance, model-based approaches are being studied and extended for cyber-physical systems, in order to lay a rigorous foundation for their design and engineering. One typical feature of models used for cyber-physical systems is the integration of discrete behavioral descriptions (often stemming from the "cyber" side), with continuous behavioral descriptions (often stemming from the "physical" side). This combination is well-known in the formal modeling literature and is often referred to as hybrid-systems modeling.[1]

* The work of M. R. Mousavi has been partially supported by the Swedish Research Council (Vetenskapsrådet) with award number 621-2014-5057 (Effective Model-Based Testing of Parallel Systems) and the Swedish Knowledge Foundation (Stiftelsen för Kunskaps- och Kompetensutveckling) in the context of the AUTO-CAAS project.

[1] Note that in addition to the interaction between the computer systems and the physical world, cyber-physical systems also feature complex patterns of communication and interaction among networked computer systems. This aspect is only mentioned in passing in the present paper.

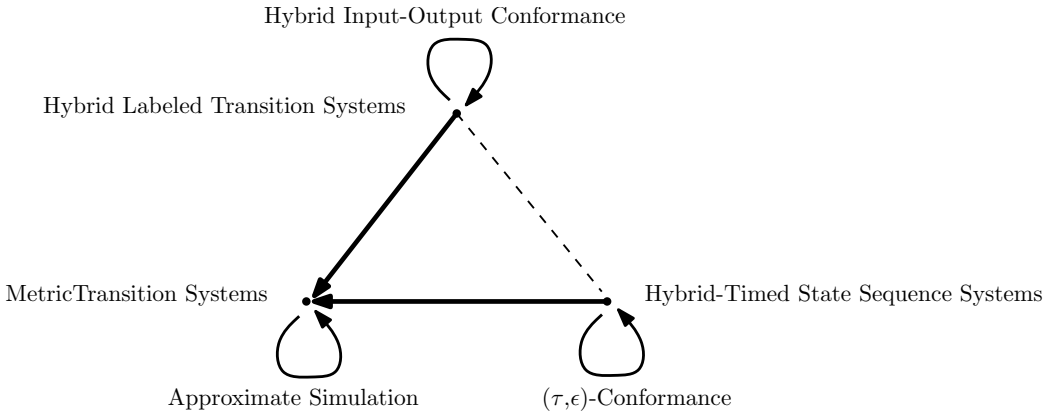


Figure 1 An overview of the semantic models and their mutual translations.

Myriad pieces of research work have been devoted to formal verification of cyber-physical systems and their hybrid-systems models; we refer to [8] for an overview. Model-based conformance testing is a lightweight verification technique, which aims at detecting faults or establishing a level of quality by generating test-cases from a model and applying it to the system under test [16, 38]. Conformance testing based on hybrid-systems models is a relatively recent subject matter [3, 4, 6, 7, 11, 14, 22, 23, 27, 32, 33, 34, 41, 59, 63, 64] and as we demonstrate in the remainder of this paper, still calls for more mature foundations and practical implementations.

A typical feature of hybrid systems research is the reliance on different approaches to system modeling and analysis that stem from different disciplines such as computer science (e.g., finite automata, process algebra, and Petri nets) and control theory (e.g., switching systems and bond graphs). Subsequently, existing approaches to conformance testing are based on different semantic models and assume different semantic properties of the model and the system under test. In this paper, we review the notions proposed in [3, 4, 22, 23, 33, 41, 59], which are, to our knowledge, the main existing proposals for a formal notion of conformance based on behavioral models. We compare these notions and reflect upon the results of this comparison. A summary of the semantic models studied in this paper, their corresponding notions of conformance and the devised translation relations are depicted in Figure 1. The three corners of the triangle denote the semantic models. The self-loops on the corners denote the notions of conformance. The solid lines on the sides of the triangle represent the translations that are presented in this paper. The dashed line is a missing translation; we refer to [46] for a related attempt in this context. We conclude the paper by describing some theoretical and practical challenges in model-based testing of cyber-physical systems.

Structure of the Paper

In Section 2, we give an overview of the three studied semantic models. In Section 3, we present mutual translations between the semantic models as outlined in Figure 1. In Section 4, we present the notions of conformance defined for these semantic models in the literature. Using these notions, in Section 5, we present full abstraction results regarding our translations; namely, we show that conforming models in the source semantic domain are projected into and reflected from conforming models in the target semantic domain. In Section 6, we reflect upon the results of our translations and comparisons and present several challenges for future research.

2 Semantic Models

Several behavioral semantic models exist for hybrid systems of which [18, Chapter2] and [19, 36] provide an overview. In this section, we review the following three of such fundamental semantic models:

- Hybrid labeled transition systems,
- Metric transition systems, and
- Hybrid-timed state sequence systems.

The choice of these models is motivated by the fact that they have been used in the context of the notions of conformance that are presented in Section 4.

2.1 Basic Definitions

The continuous behavior of a hybrid system is captured by the valuation of a set V of continuous variables. We assume that V is partitioned into disjoint sets of input variables, denoted by V_I, and output variables, denoted by V_O. A valuation of V is a function that assigns a value to each variable $v \in V$; here, only variables of type $\mathbb{R}$ (the set of real numbers) are considered. The set of all valuations of V is denoted by $Val(V)$ and is defined as the set of all functions $V \to \mathbb{R}$. To describe the piece-wise evolution of the system, we use the following notion of *trajectory* [43] (called activity in [9]).

▶ **Definition 1** (Trajectory). Let D be the set $\{(0,t] \mid t \in \mathbb{R}^{>0}\}$ of all left-open right-closed intervals.[2] A trajectory ϕ is a function of type $D \to Val(V)$, which maps each element in an interval in D to a valuation. The set of all trajectories for V is denoted by $Trajs(V)$.

Furthermore, we define the restriction operator on valuations and trajectories as follows.

▶ **Definition 2** (Valuation and Trajectory Restriction). Consider a valuation $val \in Val(V)$ and a set $V' \subseteq V$ of variables; the restriction of val to V', denoted by $val \downarrow V'$, is a valuation in $Val(V')$ such that for all $v' \in V'$, $(val \downarrow V')(v') = val(v')$.

Consider a trajectory $\phi : D \to Val(V)$; the restriction of ϕ to $V' \subseteq V$, denoted by (reusing the notation) $\phi \downarrow V'$, is the function of type $D \to Val(V')$, such that for each $d \in D$, $(\phi \downarrow V')(d) = \phi(d) \downarrow V'$.

2.2 Hybrid Labeled Transition System

A hybrid labeled transition system [19, 59] consists of a set of states with discrete (action) and continuous (trajectory) transitions between them. It is formally defined as follows:

▶ **Definition 3** (Hybrid Labeled Transition System (HLTS)). Let A be the union of disjoint sets of input actions A_I and output actions A_O. Assume that V is the union of disjoint sets of input variables V_I and output variables V_O. A hybrid labeled transition system $\mathcal{T}$ is a 5-tuple $(S, s_0, V, L, \to)$, where

- S is a (possibly infinite) set of states;
- $s_0 \in S$ is the initial state;
- V is a set of continuous variables;
- $L = A \uplus Trajs(V)$ is a set of (resp. action or trajectory) labels;

[2] The choice of "left-open right-closed" is arbitrary; we could just as well have chosen "left-closed right-open" intervals. The developments to come will be only slightly different in that case.

- $\rightarrow \subseteq S \times (L \cup \{\xi\}) \times S$ specifies the transition relation, where ξ denotes the internal action.[3]

We may write $s \xrightarrow{l} s'$ to mean $(s, l, s') \in \rightarrow$. We also write $s \xrightarrow{a}$ (respectively, $s \overset{\phi}{\rightsquigarrow}$) to mean that there exists an $s' \in S$ such that $s \xrightarrow{a} s'$ (and $\phi \in \mathit{Trajs}(V)$).

▶ **Definition 4** (Concrete HLTS). An HLTS is *concrete* if it does not have any ξ-labeled transition emanating from its reachable states.

Next, we define the notion of generalized transition relation for an HLTS, which allows us to "jump over" internal actions and to concatenate actions and trajectories to form traces.

▶ **Definition 5** (Generalized Transition Relation). Consider an HLTS $\mathcal{T} = (S, s_0, V, L, \rightarrow)$. The generalized transition relation for $\mathcal{T}$ is defined as the smallest relation $\Rightarrow \subseteq S \times L^* \times S$ where

- $s \overset{\epsilon}{\Rightarrow} s$, where ϵ denotes the empty sequence of labels;
- if $s \xrightarrow{\xi} s'$, then $s \overset{\epsilon}{\Rightarrow} s'$,
- $\forall l \in L$, if $s \xrightarrow{l} s'$, then $s \overset{l}{\Rightarrow} s'$;
- $\forall \alpha, \beta \in L^*$, if $s \overset{\alpha}{\Rightarrow} s''$ and $s'' \overset{\beta}{\Rightarrow} s'$, then $s \overset{\alpha\beta}{\Rightarrow} s'$;

We write $s \overset{\alpha}{\Rightarrow}$ to denote that there exists an $s' \in S$ such that $s \overset{\alpha}{\Rightarrow} s'$. The behavior of a system is specified by its set of traces, which are finite sequences of actions and trajectories.

▶ **Definition 6** (Trace). For HLTS $\mathcal{T}$, a trace is a finite sequence $\alpha \in L^*$ such that $s_0 \overset{\alpha}{\Rightarrow}$, where s_0 is the initial state of $\mathcal{T}$.

The length of a trace α is defined as the number of elements of the sequence and is represented by $|\alpha|$. We denote the set of all traces of $\mathcal{T}$ by $\mathit{Traces}(\mathcal{T})$. The restriction of a trace σ to a set $V' \subseteq V$ of variables, denoted by $\alpha \downarrow V'$, is defined by point-wise restriction of the trajectories in α while keeping the actions intact.

Let $\delta \in A_O$ be a special symbol to denote that a state has at least one emanating trajectory. We assume that HLTSs are normalized, i.e., for $s \in S$, if there exists $a \in \mathit{Trajs}(V)$ such that $s \overset{a}{\rightsquigarrow}$, then $s \xrightarrow{\delta} s$. This assumption is motivated by the notion of conformance on HLTSs.[4]

▶ **Definition 7** (Active Trajectory of a Trace). Consider an HLTS $\mathcal{T} = (S, s_0, V, L, \rightarrow)$, a trace $\alpha \in \mathit{Traces}(\mathcal{T})$, and a point $t \in \mathbb{R}^{>0}$ denoting time; the active trajectory of α at t, denoted by $\mathit{active}(\alpha, t)$ is defined to be a trajectory $\phi_i \in \mathit{Trajs}(V)$ when for each $j \leq i$ there exist $\phi_j \in \mathit{Trajs}(V)$, $\alpha_j \in A^*$, and $\alpha' \in (A \cup \mathit{Trajs}(V))^*$ such that the domain of each ϕ_j is $(0, t_j]$, $\alpha = \alpha_1\phi_1 ... \alpha_i\phi_i\alpha'$ and $\sum_{j=1}^{i-1} t_j < t \leq \sum_{j=1}^{i} t_j$. In that case the elapsed time of the active trajectory at t, denoted by $\mathit{elapsed}(\alpha, t)$, is defined as $t - \sum_{j=1}^{i-1} t_j$.

Moreover, we assume that HLTSs have the following three properties **A1**-**A3** [59]; we refer to [20] for some consequences of these assumptions:

- **A1** if $s \overset{\phi}{\rightsquigarrow} s'$ and $s \overset{\phi}{\rightsquigarrow} s''$, then $s' = s''$.
- **A2** if $s \overset{\phi' \frown \phi''}{\rightsquigarrow} s'$, then there exists s'' such that $s \overset{\phi'}{\rightsquigarrow} s''$ and $s'' \overset{\phi''}{\rightsquigarrow} s'$, where $\phi' \frown \phi''$ denotes concatenation of trajectories after shifting the domain of ϕ''.
- **A3** if $s \overset{\phi'}{\rightsquigarrow} s''$ and $s'' \overset{\phi''}{\rightsquigarrow} s'$, then $s \overset{\phi' \frown \phi''}{\rightsquigarrow} s'$.

[3] In the literature of concurrency theory, internal (unobservable) transitions are labeled by τ; in our context, however, τ denotes the conformance time bound and hence, we use ξ instead.

[4] We deviate from the notation commonly used in the literature in order to avoid clashing with the other notations used for conformance. In the literature, δ is used for lack of discrete output actions and ε for lack of trajectories.

Input-enabledness, defined below, is another constraint, which we in general do not require for all HLTSs. However, our full abstraction result does depend on input-enabledness, as we demonstrate in Section 5.1.

▶ **Definition 8** (Input-enabled HLTS). An HLTS $\mathcal{T} = (S, s_0, V, L, \rightarrow)$ is *input-enabled* if $\forall s \in S, \forall a \in A_I : s \stackrel{a}{\Rightarrow}$ and $\forall \phi \in \mathit{Trajs}(V_I) : \exists \phi' \in \mathit{Trajs}(V)$ such that $\phi' \downarrow V_I = \phi \wedge s \stackrel{\phi'}{\Rightarrow}$.

Finally, we define the notion of determinism, which again plays a role in our full abstraction results.

▶ **Definition 9** (Determinism). An HLTS $\mathcal{T} = (S, s_0, V, L, \rightarrow)$ is *deterministic* when $\forall s, s', s'' \in S$,

- $\forall a \in A$: if $s \stackrel{a}{\rightarrow} s'$ and $s \stackrel{a}{\rightarrow} s''$, then $s' = s''$, and
- $\forall \phi', \phi'' \in \mathit{Trajs}(V_I)$ if $\phi' \downarrow V_I = \phi'' \downarrow V_I$, if $s \stackrel{\phi'}{\rightarrow} s'$ and $s \stackrel{\phi'}{\rightarrow} s''$, then $\phi' = \phi''$ and $s' = s''$.

2.3 Metric Transition System

A metric is a function that defines the distance between different elements of a set:

▶ **Definition 10** (Metric). A metric on a set E is a function $d : E \times E \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$ such that for all $e_1, e_2, e_3 \in E$ the following properties hold:

- $d(e_1, e_3) \leq d(e_1, e_2) + d(e_2, e_3)$,
- $e_1 = e_2 \Leftrightarrow d(e_1, e_2) = 0$, and
- $d(e_1, e_2) = d(e_2, e_1)$.

Metric transition systems [31, 33, 56] are transition systems that are equipped with an observation function and one or more metrics. They are a generic formalism that can be used for specifying different sorts of dynamic behavior, such as discrete, continuous, and hybrid systems. In this paper, we use the metric transition systems whose metric is defined over observations on states [33], as quoted below.

▶ **Definition 11** (Metric Transition System (MTS)). A metric transition system $\mathcal{M}$ is a 7-tuple $(Q, Q_0, I, \rightarrow, O, \mathcal{B}, d)$, where:

- Q is a set of states,
- $Q_0 \subseteq Q$ is a set of initial states,
- I is the set of inputs,
- $\rightarrow \subseteq Q \times I \times Q$ is the transition relation,
- O is a set of outputs,
- $\mathcal{B} : Q \rightarrow O$ is an observation function, and
- $d = (d_1, d_2)$ is a pair of metrics where d_1 is defined over O and d_2 is defined over I.

Intuitively, the inputs in an MTS capture both continuous (system dynamics) and discrete behavior of the system.

2.4 Hybrid-Timed State Sequence System

Hybrid-timed state sequence systems (HSSs) [3, 4, 58] are another semantic model that assumes a discrete sampling of input and output variables.[5] The formal definition of HSS is quoted below.

[5] In [3, 4], Hybrid-Timed State Sequence Systems (HSSs) are simply called "hybrid systems"; we use the former term to be more specific and differentiate between different semantic models for hybrid systems.

▶ **Definition 12** (Hybrid-Timed State Sequence (TSS)). Consider $N \in \mathbb{N}$, $\mathbb{T} = \mathbb{R}_{\geq 0} \times \mathbb{N}$, and a set of variables V. A hybrid-timed state sequence (TSS) is defined as pair (x,σ), where $x \in (Val(V))^N$ and $\sigma \in \mathbb{T}^N$. The i'th element of a TSS (x,σ) is denoted by (x_i,σ_i), where $\sigma_i{=}(t_i, j_i) \in \mathbb{T}$.

We denote the set of all TSSs defined over the set of variables V, considering a specific sample size $N \in \mathbb{N}$, by $\mathbb{TSS}(V, N)$. The set of all TSSs over v regardless of sample size, denoted by $\mathbb{TSS}(V)$, is defined as $\bigcup_{N \in \in \mathbb{N}} \mathbb{TSS}(V, N)$.

The time domain $\mathbb{T} = \mathbb{R}_{\geq 0} \times \mathbb{N}$ is often called the "super-dense" time domain in the literature [44]. The ordering $<$ on the super-dense time domain is the lexicographical ordering by lifting the ordering on real and natural numbers.

Consider $x \in (Val(V))^N$ and $\sigma_x \in \mathbb{T}^N$; we refer to the i'th element of x and σ_x, respectively, by x_i and $\sigma_{x,i}$. Moreover for $t \in \mathbb{T}$, we write t^1 for the first (the real-valued) component of t and t^2 for the second (the natural-number-valued) component of t.

We assume for all $(x,\sigma) \in \mathbb{TSS}(V, N)$, $\sigma : \mathbb{N} \to \mathbb{T}$ is a strictly monotonic function with respect to the lexicographic ordering on $\mathbb{T}$. In other words, for each two consecutive points in σ, the real-valued component does not decrease and if the real-valued component remains the same, the natural-number-valued component increases.

▶ **Definition 13** (Hybrid-Timed State Sequence System (HSS)). A hybrid-timed state sequence system (HSS) $\mathcal{H}$ is a function $\mathcal{H} : H \times \mathbb{TSS}(V_I, N) \to \mathbb{TSS}(V_O, N)$, where $H \subseteq Val(V_I \cup V_O)$.

HSSs are assumed to be input enabled; this constraint is captured by the following formal definition:

▶ **Definition 14** (Input-enabled HSS). An HSS $\mathcal{H} : H \times \mathbb{TSS}(V_I, N) \to \mathbb{TSS}(V_O, N)$ is input-enabled, if it satisfies the following constraint:

$\forall h_0 \in H, \forall (x, \sigma_x) \in \mathbb{TSS}(V_I, N) : \exists (y, \sigma_y) \in \mathbb{TSS}(V_O, N)$ such that $(y, \sigma_y) = \mathcal{H}(h_0, (x, \sigma_x))$.

2.5 Informal Comparison of Semantic Models

The following list summarizes some of the fundamental differences among the three semantic models reviewed in this section.

- (Non-)Determinism: HLTSs and MTSs offer natural support for non-determinism. HSSs, to the contrary, are defined as a function and hence, do not provide support for non-determinism. Non-determinism is often useful in modeling abstraction from details that are either unavailable or irrelevant at the time of modeling.
- Dense-time or sampled trajectories: HLTSs explicitly assume dense-timed trajectories while HSSs assume a fixed sample size. MTSs are oblivious to this choice and can be used to model both dense- and sampled-time trajectories.
- Explicit discrete interactions: HLTSs provide an explicit means for modeling discrete observable actions that can be further used for synchronization between models and with the environment; these represent message passing synchronization among concurrent processes [10]. HSSs inherently lack this means; they could be exploited though to code discrete actions as changes in auxiliary variables' valuations. MTSs stay at a very abstract level and can be used to naturally model discrete actions as inputs.
- Input-enabledness: HLTSs and MTSs are not required to be input-enabled. HSSs as a semantic model may not be input-enabled, but the authors seem to have put it as a requirement on the semantic domain.

- Observations on states or transitions: HLTSs suggest that the observations should be placed on transitions and states do not carry any observable information. For HSSs the notions of states and transitions are a bit blurred; if one takes states to be valuations of variables and transitions to be labeled by the difference of super-dense time points, then HSSs carry observable information both on states and transitions [50]. Likewise, MTSs carry observable information both in states and transitions.
- Partial valuations: HLTSs assume that the valuation of all variables are defined by trajectories; HSSs, however, do not make such an assumption and allow for different sampling of input and output variable valuations. Similar to the previous cases MTSs are oblivious to this choice and can be used to model either of the two types of systems.

3 Translations among Semantic Models

In this section, we present translations between the semantic models introduced in Section 2, as illustrated in Figure 1.

3.1 From HLTS to MTS

The translation from HLTS to MTS defines as the states of the target MTS: the triples comprising (discrete) states, input trajectories leading to the state (or $\perp$ if none), and valuations of variables in the source HLTS. The transitions are then the union of discrete transitions and time transitions. Discrete transitions only update the discrete part of the state. In the case of trajectories, we update both the discrete state and the valuation with the state and the valuation at the upper bound of their domain.

▶ **Definition 15** (Translation from HLTS to MTS)**.** Consider an HLTS $\mathcal{T} = (S, s_0, V, L, \rightarrow)$; an MTS $\mathcal{M} = (Q, Q_0, I, \rightarrow, O, \mathcal{B}, d)$ is a translation of $\mathcal{T}$ when it satisfies the following constraints:

- $Q = S \times (\mathit{Trajs}(V_I) \cup \{\perp\}) \times \mathit{Val}(V)$,
- $Q_0 = \{\langle s_0, \phi, x\rangle \mid \phi \in \mathit{Trajs}(V_I) \cup \{\perp\}, x \in \mathit{Val}(V)\}$,
- $I = A \cup \mathbb{R}^{\geq 0}$,
- for each generalized transition $s \overset{\alpha}{\Rightarrow} s'$ of $\mathcal{T}$, and for all $x \in \mathit{Val}(V)$,
  - if $\alpha \in A$, then $\langle s, \phi, x\rangle \xrightarrow{\alpha} \langle s', \perp, x\rangle$,
  - if $\alpha \in \mathit{Trajs}(V)$, then $\langle s, \phi_\perp, x\rangle \xrightarrow{t} \langle s', \alpha \downarrow V_I, \alpha(t)\rangle$, where $\mathit{dom}(\alpha) = (0, t]$ and $\phi_\perp \in \mathit{Trajs}(V_I) \cup \{\perp\}$,
- $O \subseteq \mathit{Trajs}(V_I) \times \mathit{Val}(V)$,
- $\mathcal{B}(\langle s, \phi, x\rangle) = (\phi, x)$, for all $\langle s, \phi, x\rangle \in Q$,
- $d = (d_1, d_2)$, where

$$d_1((\phi_\perp, x), (\phi'_\perp, y)) = \begin{cases} ||x - y|| & \text{if } \phi_\perp = \phi'_\perp = \perp \text{ or} \\ & \quad \forall t \in \mathit{dom}(\phi_\perp) \cap \mathit{dom}(\phi'_\perp) : \\ & \quad\quad \phi_\perp(t) \downarrow V_I = \phi'_\perp(t) \downarrow V_I \\ \infty & \text{otherwise} \end{cases}$$

$$d_2(t, t') = |t - t'|$$

3.2 From HSS to MTS

In the translation from HSS to MTS, we define as the state of the target MTS: the initial condition, the input TSS, the output TSS and the current moment of time (in the super-dense time domain). (The initial state is an exception; it is denoted by $\perp$ and does not assume any initial state, input- or output TSS.) We define two types of transitions:

- initial transitions that define an initial condition and an input TSS for a state, and
- timed transitions that are labeled by relative time (a real number) denoting the amount of time need to reach a different output valuation in the past or in the future.

The metric on states compares the current valuations of output TSSs and the metric on transitions takes the difference in the amount of time passed.

▶ **Definition 16** (Translation from HSS to MTS). Consider HSS $\mathcal{H} : H_0 \times \mathbb{TSS}(V_I, N) \to \mathbb{TSS}(V_O, N)$; MTS $\mathcal{M} = (Q, Q_0, I, \to, O, \mathcal{B}, d)$ is a translation of $\mathcal{H}$ with respect to initial condition $h \in H_0$, if it satisfies the following constraints:

- $Q = \{\perp\} \cup$
  $\{\langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,i}\rangle \mid$
  $h_0 \in H_0,$
  $(x, \sigma(x)) \in \mathbb{TSS}(V_I, N),$
  $(y, \sigma(y)) \in \mathbb{TSS}(V_O, N),$
  $(y, \sigma_y) = \mathcal{H}(h_0, (x, \sigma_x)),$
  $i \in [1, N]\}$
- $Q_0 = \{\perp\}$,
- $I = \mathbb{R}$,
- $\perp \xrightarrow{0} \langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,1}\rangle$, for each $\langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,1}\rangle \in Q$, and

  $\langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,i}\rangle \xrightarrow{\sigma^1_{y,j} - \sigma^1_{y,i}} \langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,j}\rangle$.
  Moreover, the transition relation $\longrightarrow$ is closed under the following deduction rule:

$$\frac{q_0 \xrightarrow{t_0} q_1 \quad q_1 \xrightarrow{t_1} q_2}{q_0 \xrightarrow{t_0+t_1} q_2}$$

- $O = \{\perp\} \cup H \times \mathbb{TSS}(V_I, N) \times Val(V_O)$,
- $\mathcal{B}(\perp) = \perp$,
  $\mathcal{B}(\langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,i}\rangle) = (h_0, (x, \sigma_x), val_O)$ if and only if $y_i = val_O$,
- $d = (d_1, d_2)$, where

$$d_1(a, b) = \begin{cases} 0 & a = b = \perp \\ ||val_O - val'_O|| & a = (h_0, (x, \sigma_x), val_O) \\ & b = (h_0, (x, \sigma_x), val'_O) \\ \infty & \text{otherwise} \end{cases}$$

$$d_2(t, t') = |t - t'|$$

Next, we informally explain the definition of the metrics and the transition relation in the above-given translation.

Concerning metric d_1, the first case is self-explanatory. The second case compares the current valuations of output variables only in cases where the initial conditions and input

TSSs are identical. The last case covers both comparing the initial state with the rest and also comparing those states that have different initial conditions or input TSSs. Metric d_2 is also straightforward and only concerns the difference in relative time to reach a state.

Regarding the transition relation, the initial state is connected by a 0-labeled transition to all states in which a valid initial condition and an output produces an output TSS; moreover, in the target state the current time is set to the time of the first element in the output TSS. The time transitions are straightforward, once the initial condition and input and output TSSs are fixed, a time transition moves back and forth between the points of time (in the super-dense time domain) when outputs are produced. The last deduction rule in the definition of $\xrightarrow{t}$ is only needed to connect the initial state by a single transition to all reachable states. This is a technical requirement that pops up in the forthcoming proof of full abstraction.

4 Notions of Conformance

In this section, we review the different notions of conformance testing defined for the semantic models presented in Section 2.

4.1 Hybrid Input-Output Conformance (HIOCO)

We start with reviewing the notion of hybrid input-output conformance (*HIOCO*) [59, 60]. To this end, we first give some preliminary definitions. To avoid repetition, we assume in the remainder of this section that HLTS $\mathcal{T}$ is a 5-tuple $(S, s_0, V, L, \rightarrow)$.

▶ **Definition 17** (**after** Operator)**.** For HLTS $\mathcal{T}$ and trace $\alpha \in \mathit{Traces}(\mathcal{T})$, we define $\mathcal{T}$ **after** $\alpha = \left\{ s \,\middle|\, s_0 \stackrel{\alpha}{\Rightarrow} s \right\}$

▶ **Definition 18** (Trajectories of a State)**.** Consider HLTS $\mathcal{T}$ and state $s \in S$, then **trj**(s) is defined as $\mathbf{trj}(s) = \left\{ \sigma \in \mathit{Trajs}(V) \mid s \stackrel{\sigma}{\Rightarrow} \right\}$. This definition is extended to a set of states $S' \subseteq S$ as $\mathbf{trj}(S') = \bigcup_{s \in S'} \mathbf{trj}(s)$.

▶ **Definition 19** (Trajectory Infiltration)**.** Consider $\Sigma_{\mathcal{I}}, \Sigma_{\mathcal{S}} \subseteq \mathit{Trajs}(V)$ and $V_I \subseteq V$ as the input partition of V;

$$\mathbf{infilter}(\Sigma_{\mathcal{I}}, \Sigma_{\mathcal{S}}) = \{\sigma \in \Sigma_{\mathcal{I}} \mid \exists \sigma' \in \Sigma_{\mathcal{S}}.\ \sigma \downarrow V_I = \sigma' \downarrow V_I\}$$

Informally, $\mathbf{infilter}(\Sigma_{\mathcal{I}}, \Sigma_{\mathcal{S}})$ is the set of all trajectories $\sigma \in \Sigma_{\mathcal{I}}$ such that they find a counterpart in $\Sigma_{\mathcal{S}}$ that agrees with σ on input variables. In this definition $\mathcal{I}$ and $\mathcal{S}$ typically denote the implementation and the specification, respectively.

▶ **Definition 20** (State Output)**.** Consider $s \in S$; the output of s, denoted by $\mathbf{out}(s) \subseteq A_O$, is defined as $\mathbf{out}(s) = \{a \in A_O | s \stackrel{a}{\rightarrow}\}$, where A_O is assumed to contain δ. This definition is extended to a set of states $C \subseteq S$ as $\mathbf{out}(C) = \bigcup_{s \in C} \mathbf{out}(s)$.

Using the above-given definitions, we are now ready to define the notion of HIOCO.

▶ **Definition 21** (Hybrid I/O Conformance)**.** Consider an HLTS $\mathcal{S}$; an input-enabled HLTS $\mathcal{I}$ is said to be hybrid input-output conforming to $\mathcal{S}$, denoted by $\mathcal{I}$ *hioco* $\mathcal{S}$, if and only if for all traces $\alpha \in \mathit{Traces}(\mathcal{S})$:

$$\mathbf{out}\ (\mathcal{I}\ \mathbf{after}\ \alpha) \subseteq \mathbf{out}\ (\mathcal{S}\ \mathbf{after}\ \alpha) \wedge$$
$$\mathbf{infilter}\ (\mathbf{trj}(\mathcal{I}\ \mathbf{after}\ \alpha), \mathbf{trj}\ (\mathcal{S}\ \mathbf{after}\ \alpha)) \subseteq \mathbf{trj}\ (\mathcal{S}\ \mathbf{after}\ \alpha)$$

Informally, this notion states that for all traces α of specification $\mathcal{S}$, the discrete outputs of the implementation $\mathcal{I}$ after α must be a subset of that of the specification. Moreover after each trace α of specification $\mathcal{S}$, the set of trajectories in the implementation, with a corresponding trajectory in the specification (with equal input valuations), must be a subset of the trajectories of the specification.

4.2 Approximate Simulation

Approximate simulation [33, 32, 34, 52] is a notion of conformance that describes how well a system is approximated by another system in terms of its observable behavior. We assume below that MTSs $\mathcal{M}_i$ are defined as $(Q, Q_{0i}, I, \rightarrow, O, \mathcal{B}, d)$, i.e., have all components in common apart from the initial state.

▶ **Definition 22** (Approximate Simulation). Consider two MTSs $\mathcal{M}_1$ and $\mathcal{M}_2$; a relation $R_{\varepsilon,\tau} \subseteq Q \times Q$ is an *approximate simulation relation with precision* (ε, τ), for $\varepsilon \geq 0, \tau \geq 0$, when for all $(q_1, q_2) \in R_{\varepsilon,\tau}$,

- $d_1(\mathcal{B}(q_1), \mathcal{B}(q_2)) \leq \varepsilon$,
- if $q_1 \xrightarrow{i_1} q_1'$, then there exists a transition $q_2 \xrightarrow{i_2} q_2'$ such that $d_2(i_1, i_2) \leq \tau$ and $(q_1', q_2') \in R_{\varepsilon,\tau}$.

$\mathcal{M}_2$ *approximately simulates* $\mathcal{M}_1$ with precision (ε, τ), denoted by $\mathcal{M}_1 \preceq_{\varepsilon,\tau} \mathcal{M}_2$, when there exists a relation $R_{\varepsilon,\tau}$, such that for all $q_1 \in Q_{01}$, there exists $q_2 \in Q_{02}$ such that $(q_1, q_2) \in R_{\varepsilon,\tau}$.

$\mathcal{M}_1$ and $\mathcal{M}_2$ are *approximately bisimilar* with precision (ε, τ), denoted by $\mathcal{M}_1 \equiv_{\varepsilon,\tau} \mathcal{M}_2$, when there exists a symmetric simulation relation relating their sets of initial states.

4.3 (τ, ε)-Conformance

The notion of (τ, ε)-conformance relates two HSSs when for each sampled input, the sampled output of related pairs differ in time with the maximum value of τ and differ in valuations with the maximum value of ε.

▶ **Definition 23** ((τ, ε)-Conformance). Consider a test duration $T \in \mathbb{T}$ and $\tau, \varepsilon > 0$ and TSSs $(y, \sigma_y) \in \mathbb{TSS}(V, N)$ and $(y', \sigma_{y'}) \in \mathbb{TSS}(V, N')$; then (τ, ε)-*conforms to* $(y', \sigma_{y'})$, denoted by $(y, \sigma_y) \approx_{\tau,\varepsilon} (y', \sigma_{y'})$, if and only if

1. for all $i \in [1, N]$ such that $t_i \leq T$ there exists $k \in [1, N']$ such that $t_k \leq T$, $|t_i^1 - t_k'^1| \leq \tau$ and $||y_i - y_k'|| \leq \varepsilon$, and
2. for all $i \in [1, N']$ such that $t_i' \leq T$, there exists $k \in [1, N]$ such that $t_k \leq T$, $|t_i'^1 - t_k^1| \leq \tau$ and $||y_i' - y_k|| \leq \varepsilon$.

HSS $\mathcal{H}$ (τ, ε)-*conforms to* HSS $\mathcal{H}'$ (both with the same sets of input and output variables), denoted by $\mathcal{H} \approx_{\tau,\varepsilon} \mathcal{H}'$, when for each initial condition h_0 and each TSS (x, σ_x) on the common input variables V_I, $\mathcal{H}(h_0, (x, \sigma_x)) \approx_{\tau,\varepsilon,V_O} \mathcal{H}'(h_0, (x, \sigma_x))$.

We have made two slight modifications with respect to the original definition of (τ, ε)-conformance in [3, 4]: firstly, the original notion requires the number of discrete jumps (the natural-number-valued part of time) for related states to be identical. The main purpose for this choice is to be sensitive to Zeno behavior, but we consider this a too strong constraint for the purpose and hence, we removed this it. (We expect our results to hold in the original setting, as well.) Secondly, the original definition requires the difference of the timing and

valuations to be strictly less than the conformance bounds τ and ϵ, respectively. We have changed this to *less than or equal* to be consistent with the earlier definition of approximate simulation.

5 Full Abstraction for Translations

In this section, we show that the translations introduced in Section 3 preserve the notions of conformance for a designated subset of the source semantic domain. We also give some suggestions as to how to generalize these results to all semantic models.

5.1 Full Abstraction for the HLTS-MTS Translation

We start with proving that our translation from HLTS to MTS projects and reflects HIOCO-conforming pairs of models to approximately similar models with approximation bounds set to 0. To obtain this result, we need to focus on a very restricted subset of HLTSs, namely concrete input-enabled deterministic ones. After we present the proof, we sketch some initial ideas as to how to relax these restrictive assumptions. To avoid repetition, we assume that the HLTSs in the remainder of this section are of the form $(S_{\mathcal{T}}, s_{0\mathcal{T}}, V, L, \rightarrow)$, where $\mathcal{T}$ is the HLTS at hand. Note that we do not subscript the transition relation because it is always clear from the context and we do not subscript the set of variables and labels, because they are assumed to be the same throughout this section.

▶ **Theorem 24.** *Consider two concrete input-enabled HLTSs $\mathcal{I}$ and $\mathcal{S}$, where $\mathcal{S}$ is deterministic; assume that $trans(\mathcal{I})$ and $trans(\mathcal{S})$ denote the MTS translations of $\mathcal{I}$ and $\mathcal{S}$, respectively. Then, the following statement holds:*

$$\mathcal{I} \textbf{ hioco } \mathcal{S} \Rightarrow trans(\mathcal{I}) \preceq_{0,0} trans(\mathcal{S}).$$

Proof. We start with proving the following lemma. This lemma makes sure that the simulation relation between the states of $trans(\mathcal{I})$ and $trans(\mathcal{S})$ (given in the remainder of the proof) is well-defined.

▶ **Lemma 25.** *Consider concrete input-enabled and deterministic HLTSs $\mathcal{I}$ and $\mathcal{S}$ and an arbitrary trace $\alpha \in Traces(S)$; $\mathcal{I}$ **hioco** $\mathcal{S}$ implies that for all $s_1 \in \mathcal{I}$ **after** α, there exits $s_2 \in \mathcal{S}$ **after** α such that $\mathcal{I}_{s_1}$ **hioco** $\mathcal{S}_{s_2}$, where for an HLTS $\mathcal{T}$ and state s, $\mathcal{T}_s$ denotes the same HLTS with the initial state s.*

Proof. We prove the lemma by induction on the length of α.

Since both $\mathcal{I}$ and $\mathcal{S}$ are concrete (i.e., $s_{0\mathcal{I}}$ **after** $\epsilon = s_{0\mathcal{I}}$ and $s_{0\mathcal{S}}$ **after** $\epsilon = s_{0\mathcal{S}}$) and $\mathcal{I}_{s_{0\mathcal{I}}} = \mathcal{I}$ and $\mathcal{S}_{s_{0\mathcal{S}}} = \mathcal{S}$, the base case follows immediately.

Assume that for all $\alpha_n \in Traces(S)$ with $|\alpha_n| \leq n$ and all $s_1 \in \mathcal{I}$ **after** α_n, there exists $s_2 \in \mathcal{S}$ **after** α_n, such that $\mathcal{I}_{s_1}$ **hioco** $\mathcal{S}_{s_2}$.

Consider a trace $\alpha_{n+1} \in Traces(S)$; if $\mathcal{I}$ **after** $\alpha_{n+1} = \emptyset$, then the thesis follows vacuously.

For the case that $\mathcal{I}$ **after** $\alpha_{n+1} \neq \emptyset$, let l be the last element of α_{n+1}. Hence, there is an l-labeled transition $s_1 \xrightarrow{l} s_1'$, where $s_1 \in \mathcal{I}$ **after** α_n and $s_1' \in \mathcal{I}$ **after** α_{n+1}. It follows from the induction hypothesis that there exists a state $s_2 \in \mathcal{S}$ **after** α_n such that $\mathcal{I}_{s_1}$ **hioco** $\mathcal{S}_{s_2}$. If l is an input, since $\mathcal{S}$ is input-enabled, there exists a transition $s_2 \xrightarrow{l} s_2'$. If l is an output, it follows from the first condition of Definition 21 that $s_2 \xrightarrow{l} s_2'$. If l is a trajectory, it follows from input-enabledness of $\mathcal{S}$ that $\mathcal{I}$ affords an l' trajectory such that $l \downarrow V_I = l' \downarrow V_I$. Then, it follows from the second condition of Definition 21 that that $s_2 \xrightarrow{l} s_2'$.

Hence, it remains only to prove that $\mathcal{I}_{s'_1}$ **hioco** $\mathcal{S}_{s'_2}$; we proceed with a proof by contradiction. Suppose there exists $\alpha' \in \mathit{Traces}(S_{s'_2})$ such that one of the two conditions of Definition 21 does not hold for $\mathcal{I}_{s'_1}$ and $\mathcal{S}_{s'_2}$. It follows from $\alpha' \in \mathit{Traces}(S_{s'_2})$ that $\alpha_{n+1}\alpha' \in \mathit{Traces}(S)$ and hence, the same condition is violated for $\mathcal{I}$ and $\mathcal{S}$, which contradicts the assumption of the lemma. Hence, we conclude that $\mathcal{I}_{s'_1}$ **hioco** $\mathcal{S}_{s'_2}$. ◀

We define the following relation R and then prove that R is a witnessing approximate simulation relation for $\mathit{trans}(\mathcal{I}) \preceq_{0,0} \mathit{trans}(\mathcal{S})$:

$$R = \{(\langle s_1, \phi, x\rangle, \langle s_2, \phi, x\rangle) \mid \mathcal{I}_{s_1} \textbf{ hioco } \mathcal{S}_{s_2}, \phi \in \mathit{Trajs}(V_I) \cup \{\bot\}, x \in \mathit{Val}(V)\}$$

It follows from Definitions 10 and 15 (particularly, the definition of $\mathcal{B}$ in the latter) that for each two related states in R, $d_1(\mathcal{B}(\langle s_1, \phi, x\rangle), \mathcal{B}(\langle s_2, \phi, x\rangle)) = d_1((\phi, x), (\phi, x)) = 0$.

From $\mathcal{I}$ **hioco** $\mathcal{S}$, it follows that $(\langle s_{0,\mathcal{I}}, \bot, x\rangle, \langle s_{0,\mathcal{S}}, \bot, x\rangle) \in R$.

Hence, it only remains to show that if $\langle s_1, \phi, x\rangle \xrightarrow{\alpha} \langle s'_1, \phi', x'\rangle$, for some $\alpha \in L$, then there exists s'_2 such that $\langle s_2, \phi, x\rangle \xrightarrow{\alpha} \langle s'_2, \phi', x'\rangle$ and $(\langle s'_1, \phi', x'\rangle, \langle s'_2, \phi', x'\rangle) \in R$.

We distinguish the following cases based on whether α is an input action, output action, or a trajectory:

- $\alpha \in A_I$: Since $\mathcal{S}$ is input-enabled, $\alpha \in \mathit{Traces}(\mathcal{S}_{s_1})$. Hence, s_2 affords an α-labeled transition to some state s'_2. Since $\mathcal{S}$ is deterministic, this is the only α-labeled transition afforded by s_1. Hence, it follows from $\mathcal{I}_{s_1}$ **hioco** $\mathcal{S}_{s_2}$ and Lemma 25 that $\mathcal{I}_{s'_1}$ **hioco** $\mathcal{S}_{s'_2}$. Finally, it follows from the latter statement and the construction of R that $(\langle s'_1, \bot, x'\rangle, \langle s'_2, \bot, x'\rangle) \in R$, which was to be shown.
- $\alpha \in A_O$: It follows from $\mathcal{I}_{s_1}$ **hioco** $\mathcal{S}_{s_2}$ and Definition 21. Hence, s_2 affords an α-labeled transition to some state s'_2. With a similar reasoning as in the above item, we obtain that $(\langle s'_1, \bot, x'\rangle, \langle s'_2, \bot, x'\rangle) \in R$.
- $\alpha \in \mathit{Trajs}(V)$: Then, Since $\mathcal{S}$ is input-enabled, there exists some $\phi' \in \mathit{Traces}(\mathcal{S}_{s_1})$ such that $\phi' \downarrow V_I = \alpha \downarrow V_I$. It follows from $\mathcal{I}_{s_1}$ **hioco** $\mathcal{S}_{s_2}$ that s_2 affords an α' labeled trajectory to some state s'_2. With a similar reasoning as in the first item, we obtain that $(\langle s'_1, \phi' \downarrow V_I, x'\rangle, \langle s'_2, \phi' \downarrow V_I, x'\rangle) \in R$. ◀

The assumptions that both HLTSs are concrete and input-enabled and that the specification is deterministic are very restrictive. We envisage that concreteness and determinism assumptions can be relaxed. A possible proof technique could require a slightly modified translation that is reminiscent of the subset construction for transforming non-deterministic finite automata to determinstic ones; a similar transformation has been proposed in the setting of IOCO, see, e.g., [49, Definition 8]. For non-input-enabled HLTSs, we believe that a slightly different notion than approximate bisimulation needs to be employed; this notion should be similar to XY-bisimulation [2].

▶ **Theorem 26.** *Consider two concrete deterministic HLTSs $\mathcal{I}$ and $\mathcal{S}$, where $\mathcal{S}$ is also input-enabled, and assume that $\mathit{trans}(\mathcal{I})$ and $\mathit{trans}(\mathcal{S})$ denote their MTS translations, respectively. Then, the following statement holds:*

$$\mathit{trans}(\mathcal{I}) \preceq_{0,0} \mathit{trans}(\mathcal{S}) \Rightarrow \mathcal{I} \textbf{ hioco } \mathcal{S}$$

Proof. Consider the approximate bisimilarity relation $\preceq_{0,0}$ (i.e., the largest approximate bisimulation relation). We proceed with proving the following lemma.

▶ **Lemma 27.** *Consider two states* $\langle s_1, \phi_1, x_1\rangle$ *and* $\langle s_2, \phi_2, x_2\rangle$*, respectively, of trans*$(\mathcal{I})$ *and trans*$(\mathcal{S})$ *such that* $\langle s_1, \phi_1, x_2\rangle \preceq_{0,0} \langle s_2, \phi_2, x_2\rangle$ *; the following statements hold:*

$$\boldsymbol{out}\ (s_1) \subseteq \boldsymbol{out}\ (s_2) \tag{1}$$

and

$$\boldsymbol{trj}(s_1) \subseteq \boldsymbol{trj}\ (s_2) \tag{2}$$

Proof. We next proceed with the proofs of the two statements in the thesis:

- In order to prove that $\mathbf{out}\ (s_1) \subseteq \mathbf{out}\ (s_2)$, consider an arbitrary action $a \in \mathbf{out}\ (s_1)$. This can only be due to a transition $s_1 \xrightarrow{a} s_1'$ in $\mathcal{I}$. It follows from Definition 15 that $\langle s_1, \phi_1, x_1\rangle \xrightarrow{a} \langle s_1', \perp, x_2\rangle$. Due to $\langle s_1, \phi_1, x_2\rangle \preceq_{0,0} \langle s_2, \phi_2, x_2\rangle$, we obtain $\langle s_2, \phi_2, x_2\rangle \xrightarrow{a} \langle s_2', \phi_2, x_2\rangle$ for some $\langle s_2', \phi_2, x_2\rangle$ such that $\langle s_1', \perp, x_2\rangle \preceq_{0,0} \langle s_2', \phi_2, x_2\rangle$. It then follows from Definition 15 that $\phi_2 = \perp$ and $x_1 = x_2$. It also follows from Definition 15 that the transition of $\langle s_2, \phi_2, x_2\rangle$ is due to a transition $s_2 \xrightarrow{a} s_2'$ in $\mathcal{S}$. Hence, $a \in \mathbf{out}\ (s_2)$, which was to be shown.
- In order to prove that $\mathbf{trj}(s_1) \subseteq \mathbf{trj}\ (s_2)$ consider an arbitrary trajectory $\phi \in \mathbf{trj}(s_1)$. This statement can only be due to a transition $s_1 \xrightarrow{\phi} s_1'$ in $\mathcal{I}$. It follows from Definition 15 that $\langle s_1, \phi_1, x_1\rangle \xrightarrow{t} \langle s_1', \phi \downarrow V_I, \phi(t)\rangle$. Due to $\langle s_1, \phi_1, x_1\rangle \preceq_{0,0} \langle s_2, \phi_2, x_2\rangle$, we obtain $\langle s_2, \phi_2, x_2\rangle \xrightarrow{t} \langle s_2', \phi \downarrow V_I, \phi(t)\rangle$ for some s_2' such that $\langle s_1', \phi \downarrow V_I, \phi(t)\rangle \preceq_{0,0} \langle s_2', \phi \downarrow V_I, \phi(t)\rangle$. Hence, it follows from Definition 15 that s_2 affords a trajectory ϕ' such that $s_2 \xrightarrow{\phi'} s_2'$, $dom(\phi') = (0, t]$, $\phi'(t) = \phi(t)$, and $\phi' \downarrow V_I = \phi \downarrow V_I$. We claim that for each $t' \in (0, t]$ it holds that $\phi'(t') = \phi(t')$.

  Take an arbitrary $t' \in (0, t]$, it follows from assumption **A2** that $\langle s_1, \phi_1, x_1\rangle \xrightarrow{t'} \langle s_1'', \phi_3 \downarrow V_I, \phi(t')\rangle$ and $\langle s_1'', \phi_3 \downarrow V_I, \phi(t')\rangle \xrightarrow{t-t'} \langle s_1''', \phi_4 \downarrow V_I, \phi(t)\rangle$ for some $\phi_3, \phi_4 \in Traj(V)$ such that $\phi = \phi_3 \frown \phi_4$. Since $\phi = \phi_3 \frown \phi_4$, it follows from Definition 9 and assumption **A1** that $s_1''' = s_1'$.

  It follows from $\langle s_1, \phi_1, x_1\rangle \preceq_{0,0} \langle s_2, \phi_2, x_2\rangle$ that $\langle s_2, \phi_2, x_2\rangle \xrightarrow{t'} \langle s_2'', \phi_3' \downarrow V_I, \phi_3'(t')\rangle$ for some $\langle s_2'', \phi_3' \downarrow V_I, \phi_3'(t')\rangle$ such that $\langle s_1'', \phi_3 \downarrow V_I, \phi(t')\rangle \preceq_{0,0} \langle s_2'', \phi_3' \downarrow V_I, \phi_3'(t')\rangle$ and $\phi_3 \downarrow V_I = \phi_3' \downarrow V_I$. Also, from $\langle s_1'', \phi_3 \downarrow V_I, \phi(t')\rangle \preceq_{0,0} \langle s_2'', \phi_3' \downarrow V_I, \phi_3'(t')\rangle$, we obtain that $\phi(t') = \phi_3'(t')$. Likewise, we obtain from $\langle s_1'', \phi_3 \downarrow V_I, \phi(t')\rangle \preceq_{0,0} \langle s_2'', \phi_3' \downarrow V_I, \phi(t')\rangle$ that $\langle s_2'', \phi_3' \downarrow V_I, \phi(t')\rangle \xrightarrow{t-t'} \langle s_2''', \phi_4' \downarrow V_I, \phi(t)\rangle$ for some $\langle s_2''', \phi_4' \downarrow V_I, \phi(t)\rangle$ such that $\langle s_1''', \phi_4 \downarrow V_I, \phi(t)\rangle \preceq_{0,0} \langle s_2''', \phi_4' \downarrow V_I, \phi(t)\rangle$ and $\phi_4 \downarrow V_I = \phi_4' \downarrow V_I$. We have that $\phi = \phi_3 \frown \phi_4$, $\phi_3 \downarrow V_I = \phi_3' \downarrow V_I$, $\phi_4 \downarrow V_I = \phi_4' \downarrow V_I$ and $\phi \downarrow V_I = \phi' \downarrow V_I$; hence, due to **A1**, we obtain that $\phi' = \phi_3' \frown \phi_4'$. We already had that $\phi(t') = \phi_3'(t')$ and from $\phi' = \phi_3' \frown \phi_4'$, we obtain that $\phi(t') = \phi'(t')$, which was to be shown. ◀

In order to prove the theorem, we prove the following claim:

Consider a trace $\alpha_n \in Traces(\mathcal{S})$ with length n; for all $s_1 \in \mathcal{I}$ **after** α_n, there exists $s_2 \in \mathcal{S}$ **after** α_n such that $\langle s_1, \phi, x\rangle \preceq_{0,0} \langle s_2, \phi, x\rangle$ for all $\phi \in TrajV$, $x \in Val(V)$.

Once we prove the claim, by considering Lemma 27 the theorem follows.

We prove the claim by induction on n. For the base case, we have that $s_{0\mathcal{I}}\mathbf{after}\epsilon = s_{0\mathcal{I}}$ and $s_{0\mathcal{S}}\mathbf{after}\epsilon = s_{0\mathcal{S}}$ and it follows from Definition 15 and $\langle s_{0\mathcal{I}}, \phi, x\rangle \preceq_{0,0} \langle s_{0\mathcal{S}}, \perp, x\rangle$ and s_2 is $s_{0\mathcal{S}}$.

Assume that for all $k \leq n$, $\alpha_k \in Traces(\mathcal{S})$ and all $s_1 \in \mathcal{I}$ **after** α_k, there exists $s_2 \in \mathcal{S}$ **after** α_k such that $\langle s_1, \phi, x\rangle \preceq_{0,0} \langle s_2, \phi, x\rangle$.

Consider a trace $\alpha_{n+1} \in Traces(\mathcal{S})$; if $\mathcal{I}$ **after** $\alpha_{n+1} = \emptyset$, then the claim follows. For the case that $\mathcal{I}$ **after** $\alpha_{n+1} \neq \emptyset$, consider $l \in L$ to be the last element of α_{n+1} and assume that

$s_1 \in \mathcal{I}$ **after** α_n. According to the induction hypothesis, there exists $s_2 \in \mathcal{S}$ **after** α_n such that $\langle s_1, \phi, x\rangle \preceq_{0,0} \langle s_2, \phi, x\rangle$. We distinguish the following cases based on the type of the last element l in α_{n+1}:

- $l \in A$: Since $s_1 \xrightarrow{a} s_1'$, then $\langle s_1, \phi, x\rangle \xrightarrow{a} \langle s_1', \bot, x\rangle$ based on Definition 15. Considering $\langle s_1, \bot, x\rangle \preceq_{0,0} \langle s_2, \bot, x\rangle$, $\langle s_1, \phi, x\rangle \xrightarrow{a} \langle s_1', \bot, x\rangle$ and Definition 22, it follows that there exists a state s_2' such that $\langle s_2, \phi, x\rangle \xrightarrow{a} \langle s_2', \bot, x\rangle$ and $\langle s_1', \bot, x\rangle \preceq_{0,0} \langle s_2', \bot, x\rangle$.
  In order to show that $\langle s_1', \phi, x\rangle \preceq_{0,0} \langle s_2', \phi, x\rangle$, we prove the following more general claim:

▶ **Lemma 28.** *If* $\langle s_1', \phi, x\rangle \preceq_{0,0} \langle s_2', \phi, x\rangle$*, then* $\langle s_1', \phi', x'\rangle \preceq_{0,0} \langle s_2', \phi', x'\rangle$ *for any* $\phi' \in$ *Trajs*(V) *and* $x' \in$ *Val*(V)*.*

Proof. Consider the witnessing approximate simulation relation R such that $(\langle s_1', \phi, x\rangle\ , \langle s_2', \phi, x\rangle) \in R$; consider the extension R' of R with all pairs $(\langle s, \phi', x'\rangle, \langle s', \phi', x'\rangle)$ such that for some ϕ' and x', $(\langle s, \phi'', x''\rangle, \langle s', \phi'', x''\rangle) \in R$. It is straightforward to check that R' is an approximate simulation relation. ◀

Therefore, due to Lemma 28, $\langle s_2', \bot, x\rangle$ is the state that is approximately bisimilar to $\langle s_1', \bot, x\rangle$ where $s_2' \in \mathcal{S}$ **after** α_{n+1}, which was to be shown.

- $l \in$ *Trajs*(V): since $l \in$ **trj**(s_1) and $\langle s_1, \phi, x\rangle \preceq_{0,0} \langle s_2, \phi, x\rangle$, due to Lemma 27, we obtain that $l \in$ **trj**(s_2). It follows from Definition 15 that $\langle s_1, \phi, x\rangle \xrightarrow{t} \langle s_1', l \downarrow V_I, l(t)\rangle$, where $dom(l) = (0, t]$. Since $\mathcal{S}$ is deterministic and $l \in$ **trj**(s_2), the only possibility for $\langle s_2, \phi, x\rangle$ to mimic this transition is the transition: $\langle s_2, \phi, x\rangle \xrightarrow{t} \langle s_2', l \downarrow V_I, l(t)\rangle$, for some s_2' such that $\langle s_1', l \downarrow V_I, l(t)\rangle \preceq_{0,0} \langle s_2', l \downarrow V_I, l(t)\rangle$. It follows from Lemma 28 that $\langle s_1', \phi', x'\rangle \preceq_{0,0} \langle s_2', \phi', x'\rangle$, for each $\phi' \in$ *Trajs*(V) and $x' \in$ *Val*(V), which concludes the proof of this item and the theorem. ◀

Similar to Theorem 24, we conjecture that Theorem 26 also holds for non-deterministic HLTSs.

5.2 Full Abstraction for HSS-MTS Translation

In this section, we prove full abstraction results for the translation from HSS to MTS. In both the projection and the reflection theorems, we need to double the original conformance time bound τ. We need to stretch the conformance time bound, because the notion of (τ, ε)-conformance allows for matching valuations within τ time bounds both in the past and in the future, which is a neighborhood of width 2τ.

▶ **Theorem 29.** *Consider two HSSs* $\mathcal{I}$ *and* $\mathcal{S}$ *and* $\varepsilon > 0$*; assume that MTSs* $MTS(\mathcal{I})$ *and* $MTS(\mathcal{S})$ *are translations of* $\mathcal{I}$ *and* $\mathcal{S}$*, respectively. Then, the following statement holds:*

$$\mathcal{I} \approx_{\tau,\varepsilon} \mathcal{S} \Rightarrow MTS(\mathcal{I}) \equiv_{\varepsilon,2\tau} MTS(\mathcal{S}).$$

Proof. Consider the following relation between the states of $MTS(\mathcal{I})$ and $MTS(\mathcal{S})$:

$$\begin{aligned} R_{\varepsilon,2\tau} \;=\; & \{(\bot,\bot),\\ & (\langle h_0, (x,\sigma_x), (y,\sigma_y), \sigma_{y,i}\rangle, \langle h_0, (x,\sigma_x), (y',\sigma_{y'}), \sigma_{y',j}\rangle \mid \\ & \quad (y,\sigma_y) \approx_{\tau,\varepsilon} (y',\sigma_{y'}) \wedge \\ & \quad |\sigma^1_{y,i} - \sigma^1_{y',j}| \le \tau \\ & \quad ||y_i - y'_j|| \le \varepsilon\} \end{aligned}$$

The relation is symmetric and hence, once we prove that this relation is an approximate bisimulation relation, the theorem follows, because the initial states of $MTS(\mathcal{I})$ and $MTS(\mathcal{S})$ are related by $R_{\varepsilon,2\tau}$.

The fact that for each $(q_0, q_1) \in R_{\varepsilon,2\tau}$, $d_1(\mathcal{B}(q_0), \mathcal{B}(q_1)) \leq \varepsilon$ follows from the construction of $R_{\varepsilon,2\tau}$ and Definition 16.

It remains to prove the transfer conditions of Definition 22.

Regarding the transfer condition for $\perp$, assume that in $MTS(\mathcal{I})$, $\perp \xrightarrow{t} \langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,i}\rangle$; we immediately have that $t = \sigma_{y,i} - \sigma_{y,1}$. It follows from $\mathcal{I} \approx_{\tau,\varepsilon} \mathcal{S}$ that there exists $(y', \sigma_{y'}) = \mathcal{S}(h_0, (x, \sigma_x))$ such that $(y, \sigma_y) \approx_{\tau,\varepsilon} (y', \sigma_{y'})$. In particular, it holds that there exists a $\sigma_{y',j}$ such that $|\sigma^1_{y,i} - \sigma^1_{y',j}| \leq \tau$ and $||y_i - y'_j|| \leq \varepsilon$. It hence, follows from Definition 16 that in $MTS(\mathcal{S})$, $\perp \xrightarrow{t'} \langle h_0, (x, \sigma_x), (y', \sigma_{y'}), \sigma_{y',j}\rangle$, where $t' = \sigma_{y',j} - \sigma_{y',1}$. We note that $|\sigma_{y,1} - \sigma_{y',1}| \leq \tau$, because otherwise, one of the two points does not find a counter part (within the time range of τ) in the other sequence. Hence, we obtain that and we have that $d_2(t, t') = |(\sigma_{y,i} - \sigma_{y',j}) - (\sigma_{y,1} - \sigma_{y',1})| \leq 2\tau$. We also have that $(\langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,i}\rangle, \langle h_0, (x, \sigma_x), (y', \sigma_{y'}), \sigma_{y',j}\rangle) \in R_{\varepsilon,2\tau}$ and this concludes the transfer condition for $\perp$.

Take arbitrary states $(\langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,i}\rangle, \langle h_0, (x, \sigma_x), (y', \sigma_{y'}), \sigma_{y',j}\rangle) \in R_{\varepsilon,2\tau}$. Consider a transition $\langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,i}\rangle \xrightarrow{t} \langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,k}\rangle$; we have that $t = \sigma_{y,k} - \sigma_{y,i}$. It follows from $(y, \sigma_y) \approx_{\tau,\varepsilon} (y', \sigma_{y'})$ that there exists $m \in \mathbb{N}$ such that $|\sigma^1_{y,k} - \sigma^1_{y',m}| \leq \tau$ and $||y_k - y'_m|| \leq \varepsilon$. Due to Definition 16, we have that $\langle h_0, (x, \sigma_x), (y', \sigma_{y'}), \sigma_{y',j}\rangle \xrightarrow{t'} \langle h_0, (x, \sigma_x), (y', \sigma_{y'}), \sigma_{y',m}\rangle$ and $t' = \sigma^1_{y',j} - \sigma^1_{y',m}$. It also follows from $|\sigma^1_{y,i} - \sigma^1_{y',j}| \leq \tau$ and $|\sigma^1_{y,k} - \sigma^1_{y',m}| \leq \tau$ that $d_2(t, t') = |(\sigma_{y,i} - \sigma_{y',j}) - (\sigma_{y,1} - \sigma_{y',1})| \leq 2\tau$. We also have that $(\langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,k}\rangle, \langle h_0, (x, \sigma_x), (y', \sigma_{y'}), \sigma_{y',m}\rangle) \in R_{\varepsilon,2\tau}$ and this concludes the only remaining part of the transfer condition and the proof. ◀

▶ **Theorem 30.** *Consider two HSSs $\mathcal{I}$ and $\mathcal{S}$, and two approximation bounds $\tau \geq 0$ and $\varepsilon \geq 0$; assume that MTSs $MTS(\mathcal{I})$ and $MTS(\mathcal{S})$ are translations of $\mathcal{I}$ and $\mathcal{S}$, respectively. Then, the following statement holds:*

$$MTS(\mathcal{I}) \equiv_{\tau,\varepsilon} MTS(\mathcal{S}) \Rightarrow \mathcal{I} \approx_{\varepsilon,2\tau} \mathcal{S}.$$

Proof. In the proof to follow, we assume that $\mathcal{I}$ and $\mathcal{S}$ are of the form $\mathcal{I} : H \times \mathbb{TSS}(V_I, N) \to \mathbb{TSS}(V_O, N_{\mathcal{I}})$ and $\mathcal{S} : H \times \mathbb{TSS}(V_I, N) \to \mathbb{TSS}(V_O, N_{\mathcal{S}})$, respectively.

Consider $h_0 \in H_0$, $(x, \sigma_x) \in \mathbb{TSS}(V_I, N)$, and $(y, \sigma_y) \in \mathbb{TSS}(V_O, N)$ such that $(y, \sigma_y) = \mathcal{I}(h_0, (x, \sigma_x))$; in order to prove the theorem, we need to find a TSS $(y', \sigma_{y'}) \in \mathbb{TSS}(V_O, N_{\mathcal{I}})$ such that $(y', \sigma_{y'}) = \mathcal{I}(h_0, (x, \sigma_x))$ and $(y, \sigma_y) \approx_{\varepsilon,2\tau} (y', \sigma_{y'})$.

It follows from Definition 16 that in $MTS(\mathcal{I})$, we have $\perp \xrightarrow{t} \langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,i}\rangle$, for each $i < N_{\mathcal{I}}$. It also follows from $MTS(\mathcal{I}) \equiv_{\tau,\varepsilon} MTS(\mathcal{S})$ that there exists an approximate bisimulation relation $R_{\varepsilon,\tau}$ such that $(\{\perp\}, \{\perp\}) \in R_{\varepsilon,\tau}$. Due to the latter statement and the t-labeled transition of $\perp$ in $MTS(\mathcal{I})$, we obtain in $MTS(\mathcal{S})$ that for some $(y', \sigma_{y'}) \in \mathbb{TSS}(V_O, N_{\mathcal{S}})$ and $j \leq N_{\mathcal{S}}$, $\perp \xrightarrow{t'} \langle h_0, (x, \sigma_x), (y', \sigma'_y), \sigma_{y',j}\rangle$ such that $|\sigma^1_{y,i} - \sigma^1_{y',j}| \leq \tau$, $||y_i - y'_j|| \leq \varepsilon$, and $(\langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,i}\rangle, \langle h_0, (x, \sigma_x), (y', \sigma_{y'}), \sigma_{y',j}\rangle) \in R_{\varepsilon,\tau}$. In the remainder of the proof, without loss of generality, we assume that $\sigma^1_{y,i} - \sigma^1_{y',j}$ is non-negative; the proof for the case where $\sigma^1_{y,i} - \sigma^1_{y',j}$ is negative is symmetric and is hence, dispensed with. We claim that for any such (y', σ'_y), it holds that $(y, \sigma_y) \approx_{\varepsilon,\tau} (y', \sigma_{y'})$.

To see why this claim holds, take an arbitrary point $\sigma_{y',k}$ for $k < N_{\mathcal{S}}$. We have in $MTS(\mathcal{S})$ that $\langle h_0, (x, \sigma_x), (y', \sigma'_y), \sigma_{y',j}\rangle \xrightarrow{\sigma^1_{y',k} - \sigma^1_{y',j}} \langle h_0, (x, \sigma_x), (y', \sigma'_y), \sigma_{y',k}\rangle$. We have that $(\langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,i}\rangle, \langle h_0, (x, \sigma_x), (y', \sigma_{y'}), \sigma_{y',j}\rangle) \in R_{\varepsilon,\tau}$ and hence, we obtain in

$MTS(\mathcal{I})$ that for some $m \leq N_{\mathcal{I}}$, $\langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,i}\rangle \overset{\sigma^1_{y,m}-\sigma^1_{y,i}}{\longrightarrow} \langle h_0, (x, \sigma_x), (y, \sigma_y), \sigma_{y,m}\rangle$ such that $|(\sigma^1_{y,m} - \sigma^1_{y,i}) - (\sigma^1_{y',k} - \sigma^1_{y',j})| \leq \tau$ and $||y_k - y'_m|| \leq \varepsilon$. We only need to show that $|\sigma^1_{y,k} - \sigma^1_{y',m}| \leq 2tau$. We distinguish the following two cases based on whether $\sigma^1_{y,m} - \sigma^1_{y,i}$ is negative or not.

- $(\sigma^1_{y,m} - \sigma^1_{y,i}) - (\sigma^1_{y',k} - \sigma^1_{y',j}) \leq 0$, then we have:

$$\begin{array}{ll} (\sigma^1_{y,m} - \sigma^1_{y',k}) + \tau & \leq \\ (\sigma^1_{y,m} - \sigma^1_{y',k}) + (\sigma^1_{y',j} - \sigma^1_{y,i}) & = \\ (\sigma^1_{y,m} - \sigma^1_{y,i}) - (\sigma^1_{y',k} - \sigma^1_{y',j}) & \leq \\ 0 & \end{array}$$

  To conclude $(\sigma^1_{y,m} - \sigma^1_{y',k}) + \tau < 0$, that is $|\sigma^1_{y,m} - \sigma^1_{y',k}| < \tau < 2tau$, which was to be shown.
- $(\sigma^1_{y,m} - \sigma^1_{y,i}) - (\sigma^1_{y',k} - \sigma^1_{y',j}) > 0$, then we have:

$$\begin{array}{ll} (\sigma^1_{y,m} - \sigma^1_{y',k}) & = \\ (\sigma^1_{y,m} - \sigma^1_{y',k}) - (\sigma^1_{y',j} - \sigma^1_{y,i}) + (\sigma^1_{y',j} - \sigma^1_{y,i}) & = \\ ((\sigma^1_{y,m} - \sigma^1_{y,i}) - (\sigma^1_{y',k} - \sigma^1_{y',j})) + (\sigma^1_{y',j} - \sigma^1_{y,i}) & \leq \\ \tau + tau & = \\ 2\tau & \end{array}$$

  To summarize, $(\sigma^1_{y,m} - \sigma^1_{y',k}) \leq 2\tau$, and this concludes the proof. ◀

6 Challenges

6.1 Notions of Conformance

Partial models

Practical models are typically incomplete: firstly, they do not specify what the consequence of applying each and every intput at each and every state is (i.e., they may not be input-enabled). Secondly, they may not specify / allow to inspect the values of all variables at every state. Also, another aspects of partiality that may come in handy is to allow for different sets of variables in the specification and implementation: implementations tend to be defined in terms of far more variables than the specification and an abstraction mechanism should relate the implementation variables to the abstract variables in the specification.

HIOCO supports partiality in terms of non-input-enabled specifications, but other aspects of partiality are still missing in the theory. The notion of (τ, ε)-conformance does not support partial models and hence, its extension in this direction can be useful. Establishing a model of partiality for MTSs requires a generic framework for defining observations and metrics.

Partial models pose a further theoretical challenge for defining a notion of conformance that is a pre-order and pre-congruence; these aspects are discussed in the remainder of this section.

Non-determinism

At a high level of abstraction, many choices cannot be made or are irrelevant. Hence, non-deterministic models are natural phenomena at high levels of abstraction. It turns out that

testing based on non-deterministic models is significantly more complex and computationally more demanding than testing based on deterministic models. There are several proposals to allow for a restricted form of non-determinism [47, 62], which could be employed to reconcile non-determinism, e.g., with (τ, ε)-conformance.

It should also be noted that abstraction from internal actions is not supported by the current definitions of (τ, ε)-conformance and approximate simulation, which is useful when abstraction into internal actions is introduced in modeling and the design trajectory.

Pre-order (or Equivalence)

The notions of HIOCO and (τ, ε)-conformance are known not to be transitive for different reasons. Namely, HIOCO allows for partial models and hence, the set of traces tested in two pairs of conforming models may be different. (This is a straightforward extension of the original counter-example of transitivity for ioco [53].) For (τ, ε)-conformance the "conformance bounds" (τ and ε) add up when comparing two pairs of related models and hence, transitivity breaks. Finding reasonable conditions for establishing a pre-order conformance relation for partial models may help in exploiting such a notion in top-down design trajectory. Also regarding conformance bounds, calculating new bounds on the compositions of conformance relations is an interesting research direction.

Pre-congruence (compositionality)

Coming up with a notion of conformance testing that supports partial models and / or approximate comparison of behaviors and is also compositional is another non-trivial challenge. In addition to [57], there seems to be renewed interest in addressing this challenge in the context of IOCO [12, 21]. Also the authors of [13] suggest that it may be possible to achieve new results through the meta-theory of structural operational semantics.

Logical characterization

The notions of testing developed in concurrency theory are often motivated by a notion observer that can perform certain interactions with the system [5, 26]. This type of theoretical connection between the extensional and intensional notions seems to be open for the notion of conformance studied in this paper. Along the same lines, it is unclear what the logical characterization of these notions are, i.e., what class of properties (e.g., in extensions of modal μ-calculus or temporal logic [24, 25, 29]) are preserved under these notion of conformance. We refer to [4, 28] for some related results in this direction.

Distributed systems / testers

Conformance testing of asynchronous and distributed systems has been extensively studied; examples of recent work in adapting different notions of conformance to asynchronous and distributed settings are [37, 48, 51, 55, 61]. Distributed testing introduces several issues regarding nondeterminism and controllability in test-case generation, as well as reconstructing observation from partial distributed observations when checking conformance. This combination is known to be notoriously difficult and becomes even more challenging in the setting of cyber-physical systems.

Quiescence, agility, and Zeno behavior

Observing quiescence, i.e., lack of outputs and internal actions, has been a key aspect of the original IOCO theory [53], but it has been left out of HIOCO. In HIOCO, quiescence has been replaced with the agility observation which serves a different purpose, namely to note when time may or may not pass. A careful analysis of these two observations (i.e., quiescence and agility) and the different possible combinations of choices concerning them remains to be done.

Zeno behavior and chattering [42, 65] are modeling phenomenon, which bear resemblance to the divergence issue in the traditional notions of conformance testing. An ideal notion of conformance should be sensitive to these issues and distinguish systems featuring and lacking Zeno behavior. The notion of (τ, ε)-conformance counts the number of discrete steps in order to distinguish Zeno- and non-Zeno behavior, which is in our view somewhat ad-hoc. It is also contrary to the intuition that discrete jumps should be visible only through their observable effects. We believe, hence, that a more thorough treatment of Zeno behavior in the notions of conformance for cyber-physical systems is due.

6.2 Test-Case Generation

Robustness

In [35], a notion of robustness in test-case generation has been introduced for (τ, ε)-conformance. Robustness ensures the generated test-cases remain within the (τ, ε) boundaries of guard conditions (for transitions) and invariants of state space. This ensures that off-line test case can always be applied to the implementation without forcing it into an unintended state or violating the invariant. However, this turns out to be very challenging for generic hybrid systems, which requires solutions to Ordinary Differential Equations (ODEs) or Differential Algebraic Equations (DAEs). Finding a practical subset of hybrid system models for which robustness can be decided efficiently and building a test-case generation algorithm around it is a practical challenge.

Currently the notion of (τ, ε)-conformance assumes a single conformance bound for all variables; this needs to be further detailed, perhaps into a vector-valued conformance bound. Moreover, the useful notion of "conformance degree" [4], which determines the least conformance bounds can be adapted to this vector-valued setting, allowing for a Pareto analysis of conformance.

Coverage

Defining a model-based notion of coverage is still a relevant research question. In [15, 17, 40], some recent attempts have been made in consolidating traditional notions of coverage such as regularity and uniformity [30]; these notions are formalizations of category-partition techniques. Regarding continuous dynamics, various notions of coverage have been studied in [22, 23, 27, 41]. A combination of these ideas need to be considered for defining a satisfactory notion of coverage for hybrid systems.

Sampling

The choice of sampling in [4] is left unspecified and is mentioned as future work. This is a non-trivial problem and an appropriate choice of sampling requires knowing the solutions to the systems dynamics equations. Coverage-based choice of sampling is a possible solution to this issue [22, 23, 27, 41], which requires further investigation in the case of (τ, ε)-conformance.

6.3 Practical challenges

Tooling

We are aware of only few academic tools for conformance testing of cyber-physical systems, namely, HTG [22] and S-TaLiRo [39]. Also in [35] a prototype Matlab-based tool is reported which implements a test-case generation and conformance checking algorithm based on the notion of (τ, ε)-conformance. Developing tools and benchmarks for conformance testing of cyber-physical systems is certainly a challenge for the years to come.

Rigorous Models

Formal models are the starting point of model-based testing and yet, they often do not exist for industrial systems. This makes bars wide application of model-based testing in industrial practice. We see two possible solutions to this problem. In some domains, (domain-specific) languages are common in the design trajectory. For example, Matlab Simulink models are common, among others, in the automotive domain. Hence, building model-based testing tools that use such domain-specific models as their input language can be very beneficial. (Reactis and S-TaLiRo, respectively, are examples of commercial and academic model-based testing tools with such an input language.)

Model-learning techniques (see, e.g., [1, 2]) (also called learning-based testing [45] and test-based modeling [54]) may provide an alternative path to building or updating models for model-based testing.

Online testing

Applying on-line testing techniques to cyber-physical systems requires predictable real-time performance of test-case generation, test-case execution and conformance checking algorithms, none of which is trivial. Moreover, in the presence of non-determinism, conformance checking requires calculating the set of current states in a huge state-space, which becomes intractable in concurrent systems; employing on-the-fly reduction techniques in such a setting is inevitable.

Acknowledgments. We thank Pieter Cuijpers, Eugenio Moggi, Wojciech Mostowski, Michel A. Reniers, and Walid Taha for providing insightful comments on earlier drafts of this paper.

References

1 Fides Aarts, Bengt Jonsson, Johan Uijen, and Frits W. Vaandrager. Generating models of infinite-state communication protocols using regular inference with abstraction. *Formal Methods in System Design*, 46(1):1–41, 2015.

2 Fides Aarts and Frits W. Vaandrager. Learning I/O automata. In *Proceedings of the 21th International Conference on Concurrency Theory (CONCUR 2010)*, volume 6269 of *Lecture Notes in Computer Science*, pages 71–85. Springer, 2010.

3 Houssam Abbas, Bardh Hoxha, Georgios E. Fainekos, J. V. Deshmukh, James Kapinski, and Koichi Ueda. WiP abstract: Conformance testing as falsification for cyber-physical systems. In *Proceedings of the ACM/IEEE 5th International Conference on Cyber-Physical Systems (ICCPS 2014)*, page 211. IEEE CS, 2014. Available online: http://arxiv.org/abs/1401.5200.

4 Houssam Abbas, Hans Mittelmann, and Georgios E. Fainekos. Formal property verification in a conformance testing framework. In *12th ACM-IEEE International Conference on*

Formal Methods and Models for System Design (MEMOCODE 2014), pages 155–164. IEEE, 2014.

5 Samson Abramsky. Observation equivalence as a testing equivalence. *Theoretical Computer Science*, 53(2-3):225–241, 1987.

6 Bernhard K. Aichernig, Harald Brandl, Elisabeth Jöbstl, and Willibald Krenn. Model-based mutation testing of hybrid systems. In *Revised Selected Papers from the 8th International Symposium on Formal Methods for Components and Objects (FMCO 2009)*, volume 6286 of *Lecture Notes in Computer Science*, pages 228–249. Springer, 2010.

7 Bernhard K. Aichernig, Harald Brandl, and Franz Wotawa. Conformance testing of hybrid systems with qualitative reasoning models. *Electronic Notes in Theoretical Computer Science*, 253(2):53–69, 2009. Proceedings of Fifth Workshop on Model Based Testing (MBT 2009).

8 Rajeev Alur. Formal verification of hybrid systems. In *Proceedings of the 11th International Conference on Embedded Software (EMSOFT 2011)*, pages 273–278. ACM, 2011.

9 Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

10 Jos C. M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335(2-3):131–146, 2005.

11 Klaus Bender, Manfred Broy, István Péter, Alexander Pretschner, and Thomas Stauner. Model based development of hybrid systems: Specification, simulation, test case generation. In *Modelling, Analysis, and Design of Hybrid Systems*, volume 279 of *Lecture Notes in Control and Information Sciences*, pages 37–51. Springer, 2002.

12 Nikola Benes, Przemyslaw Daca, Thomas A. Henzinger, Jan Kretínský, and Dejan Nickovic. Complete composition operators for IOCO-testing theory. In *Proceedings of the 18th International ACM SIGSOFT Symposium on Component-Based Software Engineering (CBSE 2015)*, pages 101–110. ACM, 2015.

13 Harsh Beohar and Mohammad Reza Mousavi. A pre-congruence format for XY-simulation. In *Proceedings of the 6th International Conference on Fundamentals of Software Engineering (FSEN 2015)*, Lecture Notes in Computer Science, 2015.

14 Harald Brandl, Martin Weiglhofer, and Bernhard K. Aichernig. Automated conformance verification of hybrid systems. In *Proceedings of the 10th International Conference on Quality Software (QSIC 2010)*, pages 3–12. IEEE CS, 2010.

15 Cécile Braunstein, Anne Elisabeth Haxthausen, Wen-ling Huang, Felix Hübner, Jan Peleska, Uwe Schulze, and Linh Vu Hong. Complete model-based equivalence class testing for the ETCS ceiling speed monitor. In *Proceedings of the 16th International Conference on Formal Engineering Methods on Formal Methods and Software Engineering (ICFEM 2014)*, volume 8829 of *Lecture Notes in Computer Science*, pages 380–395. Springer, 2014.

16 Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner. *Model-Based Testing of Reactive Systems: Advanced Lectures*, volume 3472 of *Lecture Notes in Computer Science*. Springer, 2005.

17 Ana Cavalcanti and Marie-Claude Gaudel. Data flow coverage for circus-based testing. In *Proceedings 17th International Conference of the Fundamental Approaches to Software Engineering (FASE 2014)*, volume 8411 of *Lecture Notes in Computer Science*, pages 415–429. Springer, 2014.

18 Pieter Cuijpers. *Hybrid Process Algebra*. PhD thesis, Department of Computer Science, Eindhoven University of Technology, 2004.

19 Pieter Cuijpers, Michel Reniers, and Maurice Heemels. Hybrid transition systems. Technical Report CSR-02-12, Department of Computer Science, Eindhoven University of Technology, 2002.

20 Pieter J. L. Cuijpers and Michel A. Reniers. Lost in translation: Hybrid-time flows vs. real-time transitions. In *Proceedings of the 11th International Workshop on Hybrid Systems: Computation and Control (HSCC 2008)*, volume 4981 of *Lecture Notes in Computer Science*, pages 116–129. Springer, 2008.

21 Przemyslaw Daca, Thomas A. Henzinger, Willibald Krenn, and Dejan Nickovic. Compositional specifications for ioco testing. In *Proceedings of the 7th IEEE International Conference on Software Testing, Verification and Validation (ICST 2014)*, pages 373–382. IEEE CS, 2014.

22 Thao Dang. Model-based testing of hybrid systems. In Justyna Zander, Ina Schieferdecker, and Pieter J. Mosterman, editors, *Model-based Testing for Embedded Systems*, pages 383–424. CRC Press, 2011.

23 Thao Dang and Tarik Nahhal. Coverage-guided test generation for continuous and hybrid systems. *Formal Methods in System Design*, 34(2):183–213, 2009.

24 Jennifer M. Davoren. On hybrid systems and the modal μ-calculus. In *Hybrid Systems V*, volume 1567 of *Lecture Notes in Computer Science*, pages 38–69. Springer, 1999.

25 Jennifer M. Davoren, Vaughan Coulthard, Nicolas Markey, and Thomas Moor. Non-deterministic temporal logics for general flow systems. In *Proceedings of the 7th International Workshop on Hybrid Systems: Computation and Control (HSCC 2004)*, volume 2993 of *Lecture Notes in Computer Science*, pages 280–295. Springer, 2004.

26 Rocco De Nicola and Matthew Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

27 Tommaso Dreossi, Thao Dang, Alexandre Donzé, James Kapinski, Xiaoqing Jin, and Jyotirmoy V. Deshmukh. Efficient guiding strategies for testing of temporal properties of hybrid systems. In *Proceedings of the 7th International NASA Formal Methods Symposium (NFM 2015)*, volume 9058 of *Lecture Notes in Computer Science*, pages 127–142. Springer, 2015.

28 Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

29 Carlo A. Furia and Matteo Rossi. On the expressiveness of MTL variants over dense time. In *Proceedings of the 5th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2007)*, volume 4763 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 2007.

30 Marie-Claude Gaudel. Testing can be formal, too. In *Proceedings of the 6th International Joint Conference on Theory and Practice of Software Development (TAPSOFT 1995)*, volume 915 of *Lecture Notes in Computer Science*, pages 82–96. Springer, 1995.

31 Alessandro Giacalone, Chi-Chang Jou, and Scott A Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of the IFIP TC2 Working Conference on Programming Concepts and Methods (PROCOMET 1990)*, pages 443–458. North-Holland, 1990.

32 Antoine Girard, A. Agung Julius, and George J. Pappas. Approximate simulation relations for hybrid systems. *Discrete Event Dynamic Systems*, 18(2):163–179, 2008.

33 Antoine Girard and George J. Pappas. Approximation metrics for discrete and continuous systems. Technical Report MS-CIS-05-10, Dept. of CIS, University of Pennsylvania, 2005.

34 Antoine Girard and George J. Pappas. Approximate bisimulation: A bridge between computer science and control theory. *European Journal of Control*, 17(5-6):568–578, 2011.

35 Antoine Girard and George J. Pappas. A tool prototype for model-based testing of cyber-physical systems. Technical Report CST 2015.090, Contol Systems Group, Dept. of Mechanical Engineering, Eindhoven University of Technology, 2015.

36 Rafal Goebel, Ricardo G. Sanfelice, and Andrew R. Teel. Hybrid dynamical systems. *IEEE Control Systems*, 29(2):28–93, 2009.

37 Robert M. Hierons. Generating complete controllable test suites for distributed testing. *IEEE Trans. Software Eng.*, 41(3):279–293, 2015.

38 Robert M. Hierons, Kirill Bogdanov, Jonathan P. Bowen, Rance Cleaveland, John Derrick, Jeremy Dick, Marian Gheorghe, Mark Harman, Kalpesh Kapoor, Paul J. Krause, Gerald Lüttgen, Anthony J. H. Simons, Sergiy A. Vilkomir, Martin R. Woodward, and Hussein Zedan. Using formal specifications to support testing. *ACM Computing Surveys*, 41(2):9:1–9:76, 2009.

39 Bardh Hoxha, Houssam Abbas, and Georgios E. Fainekos. Using S-TaLiRo on industrial size automotive models. In *Proceedings of the Applied Verification for Continuous and Hybrid Systems (ARCH 2014)*, 2014.

40 Wen-ling Huang and Jan Peleska. Exhaustive model-based equivalence class testing. In *Proceedings of the 25th IFIP WG 6.1 International Conference of Testing Software and Systems (ICTSS 2013)*, volume 8254 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2013.

41 A. Agung Julius, Georgios E. Fainekos, Madhukar Anand, Insup Lee, and George J. Pappas. Robust test generation and coverage for hybrid systems. In *Proceedings of the 10th International Workshop on Hybrid Systems: Computation and Control (HSCC 2007)*, volume 4416 of *Lecture Notes in Computer Science*, pages 329–342. Springer, 2007.

42 Daniel Liberzon. *Switching in Systems and Control*. Systems & Control: Foundations and Application. Birkhäuser, 2003.

43 Nancy Lynch, Roberto Segala, and Frits Vaandrager. Hybrid I/O automata. *Information and Computation*, 185(1):105–157, 2003.

44 Oded Maler, Zohar Manna, and Amir Pnueli. From timed to hybrid systems. In *Proceedings of the REX Workshop on Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 447–484. Springer, 1992.

45 Karl Meinke, Fei Niu, and Muddassar A. Sindhu. Learning-based software testing: A tutorial. In *International Workshops on Leveraging Applications of Formal Methods, Verification, and Validation*, volume 336 of *Communications in Computer and Information Science*, pages 200–219. Springer, 2012.

46 Morteza Mohaqeqi, Mohammad Reza Mousavi, and Walid Taha. Conformance testing of cyber-physical systems: A comparative study. In *Proceedings of the 14th International Workshop on Automated Verification of Critical Systems (AVOCS 2014)*, volume 70 of *Electronic Communications of the EASST*, 2014.

47 Neda Noroozi. *Improving Theories of Input Output Conformance Testing*. PhD thesis, Eindhoven University of Technology, The Netherlands, 2015.

48 Neda Noroozi, Ramtin Khosravi, Mohammad Reza Mousavi, and Tim A. C. Willemse. Synchrony and asynchrony in conformance testing. *Software and System Modeling*, 14(1):149–172, 2015.

49 Neda Noroozi, Mohammad Reza Mousavi, and Tim A. C. Willemse. Decomposability in input output conformance testing. In *Proceedings of the 8th Workshop on Model-Based Testing (MBT 2013)*, volume 111 of *Electronic Proceedings in Theoretical Computer Science*, pages 51–66, 2013.

50 Jan Willem Polderman and Jan C. Willems. *Introduction to Mathematical Systems Theory: A Behavioral Approach*, volume 26 of *Texts in Applied Mathematics*. Springer, 1998.

51 Adenilso Simao and Alexandre Petrenko. From test purposes to asynchronous test cases. In *Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW 2010)*, pages 1–10. IEEE CS, 2010.

52 Paulo Tabuada. Approximate simulation relations and finite abstractions of quantized control systems. In *Proceedings of the 10th International Workshop on Hybrid Systems:*

Computation and Control (HSCC 2007), volume 4416 of *Lecture Notes in Computer Science*, pages 529–542. Springer, 2007.

53 Jan Tretmans. *A formal Approach to conformance testing.* PhD thesis, University of Twente, The Netherlands, 1992.

54 Jan Tretmans. Model-based testing and some steps towards test-based modelling. In *Advanced Lectures of the 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM 2011)*, volume 6659 of *Lecture Notes in Computer Science*, pages 297–326. Springer, 2011.

55 Jan Tretmans and Louis Verhaard. A queue model relating synchronous and asynchronous communication. In *Proceedings of the IFIP Symposium on Protocol Specification, Testing and Verification XII*, pages 131–145, Amsterdam, The Netherlands, The Netherlands, 1992. North-Holland Publishing Co.

56 Franck van Breugel, Claudio Hermida, Michael Makkai, and James Worrell. An accessible approach to behavioural pseudometrics. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005)*, volume 3580 of *Lecture Notes in Computer Science*, pages 1018–1030. Springer, 2005.

57 Machiel van der Bijl, Arend Rensink, and Jan Tretmans. Compositional testing with IOCO. In *Proceedings of the 3rd International Workshop on Formal Approaches to Testing of Software (FATES 2003)*, volume 2931 of *Lecture Notes in Computer Science*, pages 86–100. Springer, 2004.

58 Arjan van der Schaft and Hans Schumacher. *An Introduction to Hybrid Dynamical Systems*, volume 251 of *Lecture Notes in Control and Information Sciences*. Springer, 2000.

59 Michiel van Osch. Hybrid input-output conformance and test generation. In *Formal Approaches to Software Testing and Runtime Verification*, volume 4262 of *Lecture Notes in Computer Science*, pages 70–84. Springer, 2006.

60 Michiel van Osch. *Automated Model-based Testing of Hybrid Systems.* PhD thesis, Eindhoven University of Technology, The Netherlands, 2009.

61 Louis Verhaard, Jan Tretmans, Pim Pars, and Ed Brinksma. On asynchronous testign. In *Protocol Test Systems*, volume C-11 of *IFIP Transaction*, pages 55–66, 1992.

62 Martin Weiglhofer. *Automated Software Conformance Testing.* PhD thesis, Graz University of Technology, Austria, 2009.

63 Matthias Woehrle, Kai Lampka, and Lothar Thiele. Segmented state space traversal for conformance testing of cyber-physical systems. In *Proceedings of the 9th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS 2011)*, volume 6919 of *Lecture Notes in Computer Science*, pages 193–208. Springer, 2011.

64 Matthias Woehrle, Kai Lampka, and Lothar Thiele. Conformance testing for cyber-physical systems. *ACM Transactions on Embedded Computing Systems*, 11(4):84:1–84:23, 2013.

65 Jun Zhang, Karl Henrik Johansson, John Lygeros, and Shankar Sastry. Zeno hybrid systems. *International Journal of Robust and Nonlinear Control*, 11(5):435–451, 2001.

Behavioural Equivalences for Co-operating Transactions

Matthew Hennessy

Trinity College Dublin, Ireland
matthew.hennessy@scss.tcd.ie

Abstract

Relaxing the isolation requirements on transactions leads to systems in which transactions can now co-operate to achieve distributed goals. However in the absence of isolation it is not easy to understand the desired behaviour of transactional systems, or the extent to which the other standard ACID properties of transactions can be maintained: atomicity, consistency and durability. In this talk I will give an overview of some recent work in this area, outlining semantic theories for a process calculus which has been augmented by a new construct for co-operating transactions.

1998 ACM Subject Classification C.2.4 Distributed Systems

Keywords and phrases Behavioural equivalences, transactional systems

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.41

Category Invited Paper

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 41–41

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

Applications of Automata and Concurrency Theory in Networks

Alexandra Silva*

Radboud University, The Netherlands
alexandra@cs.ru.nl

Abstract

Networks have received widespread attention in recent years as a target for domain-specific language design. The emergence of *software-defined networking* (SDN) as a popular paradigm for network programming has led to the appearance of a number of SDN programming languages seeking to provide high-level abstractions to simplify the task of specifying the packet-processing behavior of a network.

Previous work by Anderson et al. [1] introduced NetKAT, a language and logic for specifying and verifying the packet-processing behavior of networks. NetKAT provides general-purpose programming constructs such as parallel and sequential composition, conditional tests, and iteration, as well as special-purpose primitives for querying and modifying packet headers and encoding network topologies. In contrast to competing approaches, NetKAT has a formal mathematical semantics and an equational deductive system that is sound and complete over that semantics, as well as a PSPACE decision procedure. It is based on Kleene algebra with tests (KAT), an algebraic system for propositional program verification that has been extensively studied for nearly two decades [3]. Several practical applications of NetKAT have been developed, including algorithms for testing reachability and non-interference and a syntactic correctness proof for a compiler that translates programs to hardware instructions for SDN switches.

In a follow-up paper [2], the coalgebraic theory of NetKAT was developed and a bisimulation-based algorithm for deciding equivalence was devised. The new algorithm was shown to be significantly more efficient than the previous naive algorithm [1], which was PSPACE in the best case and the worst case, as it was based on the determinization of a nondeterministic algorithm. Along with the coalgebraic model of NetKAT, the authors presented a specialized version of the Brzozowski derivative in both semantic and syntactic forms. They also also proved a version of Kleene's theorem for NetKAT that shows that the coalgebraic model is equivalent to the standard packet-processing and language models introduced previously [1]. They demonstrated the real-world applicability of the tool by using it to decide common network verification questions such as all-pairs connectivity, loop-freedom, and translation validation – all pressing questions in modern networks.

This talk will survey applications of automata theory, concurrency theory and coalgebra to problems in networking. We will suggest directions for exploring the bridge between the two communities and ways to deliver new synergies. On the one hand, this will lead to new insights and techniques that will enable the development of rigorous semantic foundations for networks. On the other hand, the idysiocransies of networks will provide new challenges for the automata and concurrency community.

1998 ACM Subject Classification C.2.4 Distributed Systems, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Automata, network programming, coalgebra

* Based on joint work with Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mathew Milano, Mark Reitblatt, and Laure Thompson.

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 42–43

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

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.42

Category Invited Paper

References

1 Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. Netkat: semantic foundations for networks. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'14, San Diego, CA, USA, January 20–21, 2014*, pages 113–126. ACM, 2014.

2 Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. A coalgebraic decision procedure for netkat. In Sriram K. Rajamani and David Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'15, Mumbai, India, January 15–17, 2015*, pages 343–355. ACM, 2015.

3 Dexter Kozen. Kleene algebra with tests. *ACM Trans. Program. Lang. Syst.*, 19(3):427–443, 1997.

Distributed Local Strategies in Broadcast Networks*

Nathalie Bertrand[1], Paulin Fournier[2], and Arnaud Sangnier[3]

1 **Inria Rennes Bretagne Atlantique, France**
2 **ENS Rennes, Université Rennes 1, France**
3 **LIAFA, Université Paris Diderot, Sorbonne Paris Cité, CNRS, France**

Abstract

We study the problems of reaching a specific control state, or converging to a set of target states, in networks with a parameterized number of identical processes communicating via broadcast. To reflect the distributed aspect of such networks, we restrict our attention to executions in which all the processes must follow the same *local strategy* that, given their past performed actions and received messages, provides the next action to be performed. We show that the reachability and target problems under such local strategies are NP-complete, assuming that the set of receivers is chosen non-deterministically at each step. On the other hand, these problems become undecidable when the communication topology is a clique. However, decidability can be regained for reachability under the additional assumption that all processes are bound to receive the broadcast messages.

1998 ACM Subject Classification F.3 Logics and Meanings of Programs, F.1.1 Models of Computation

Keywords and phrases Broadcast Networks, Parameterized Verification, Local strategies

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.44

1 Introduction

Parameterized models for distributed systems. Distributed systems are nowadays ubiquitous and distribution is one of the main paradigms in the conception of computing systems. Conceiving, analyzing, debugging and verifying such systems are tedious tasks which lately received an increased interest from the formal methods community. Considering parametric models with an unknown number of identical processes is a possible approach to tame distributed systems in which all processes share the same code. It has the advantages to allow one to establish the correctness of a system independently of the number of participants, and to ease bugs detection by the possibility to adapt the number of processes on demand.

In their seminal paper on distributed models with many identical entities [14], German and Sistla represent the behavior of a network by finite state machines interacting via 'rendezvous' communications. Variants have then been proposed, to handle different communication means, like broadcast communication [11], token-passing [6, 2], message passing [5] or shared memory [12]. In his nice survey on such parameterized models [10], Esparza shows that minor changes, such as the presence or absence of a controller in the system, can drastically modify the complexity of the verification problems. Another perspective for parametric systems has been proposed by Bollig who studied their expressive power with respect to logics over Message Sequence Charts [4].

* This work is partially supported by the ANR national research program ANR-14-CE28-0002 PACS.

Broadcast protocols. Among the various parametric models of networks, broadcast protocols, originally studied by Esparza *et al.* [11], have later been analyzed under a new viewpoint, leading to new insights on the verification problems. Specifically, a low level model to represent the main characteristics of ad-hoc networks has been proposed [8]: the network is equipped with a communication topology and processes communicate via broadcast to their neighbors. It was shown that, given a protocol represented by a finite state machine performing internal actions, broadcasts and receptions of messages, the problem of deciding whether there exists an initial communication topology from which one of the processes can reach a specific control state is undecidable. The same holds for the target problem, which asks whether all processes can converge to a set of target states. For both the reachability and the target problems, decidability can however be regained, by considering communication topologies that can change non-deterministically at any moment [7]. Another option to recover decidability of the reachability problem is to restrict the topologies to clique graphs [9], yielding a model equivalent to broadcast protocols.

Local distributed strategies. In this paper, we consider the reachability and target problems under a new perspective, which we believe could also be interesting for other 'many identical processes' models. In such models, the protocol executed by each process is often described by a finite state machine that can be non-deterministic. Therefore it may happen that two processes behave differently, even if they have the same information on what has happened so far in an execution. To forbid such non-truly distributed behaviors, we constrain processes to take the same decisions in case they fired the same sequence of transitions so far. We thus study the reachability and target problems in broadcast protocols restricted to *local strategies*. Interestingly, the notably difficult distributed controller synthesis problem [15] is relatively close to the problem of existence of a local strategy. Indeed a local strategy corresponds to a local controller for the processes executing the protocol and whose role is to resolve the non-deterministic choices.

Our contributions. First we show that the reachability and target problems under local strategies in reconfigurable broadcast networks are NP-complete. To obtain the upper bound, we prove that local strategies can be succinctly represented by a finite tree of polynomial size in the size of the input protocol. This result is particularly interesting, because deciding the existence of a local strategy is intrinsically difficult. Indeed, even with a fixed number of processes, the locality constraint cannot be simply tested on the induced transition system, and *a priori* local strategies may need unbounded memory. From our decidability proofs, we derive an upper bound on the memory needed to implement the local strategies. We also give cutoffs, *i.e.* upper bounds on the minimal number of processes needed to reach or converge to target states. Second we show the two problems to be undecidable when the communication topology is a clique. Moreover, the undecidability proof of the target problem holds even if the locality assumption is dropped. However, the reachability problem under local strategies in clique is decidable (yet non-primitive recursive) for complete protocols, *i.e.* when receptions are always possible from every state.

Due to lack of space, omitted details and proofs can be found in the companion research report [3].

2 Networks of reconfigurable broadcast protocols

In this paper, given $i, j \in \mathbb{N}$ such that $i \leq j$, we let $[i..j] = \{k \mid i \leq k \leq j\}$. For a set E and a natural $\ell > 0$, let E^ℓ be the set of vectors $\mathbf{v}$ of size ℓ over E. For a vector $\mathbf{v} \in E^\ell$ and $i \in [1..\ell]$, $\mathbf{v}[i]$ is the i-th component of $\mathbf{v}$ and $|\mathbf{v}| = \ell$ its size. The notation $\mathcal{V}_E$ stands for the

infinite set $\bigcup_{\ell \in \mathbb{N} \setminus \{0\}} E^\ell$ of all vectors over E. We will use the notation $\mathcal{M}(E)$ to denote the set of multi-sets over E.

2.1 Syntax and semantics

We begin by presenting our model for networks of broadcast protocols. Following [8, 9, 7], we assume that each process in the network executes the same (non-deterministic) broadcast protocol given by a finite state machine where the actions are of three kinds: broadcast of a message m (denoted by $!!m$), reception of a message m (denoted by $??m$) and internal action (denoted by ε).

▶ **Definition 1.** A *broadcast protocol* is a tuple $\mathcal{P} = (Q, q_0, \Sigma, \Delta)$ with Q a finite set of control states; $q_0 \in Q$ the initial control state; Σ a finite message alphabet and $\Delta \subseteq Q \times (\{!!m, ??m \mid m \in \Sigma\} \cup \{\varepsilon\}) \times Q$ a finite set of edges.

We denote by $A(q)$ the set $\{(q, \varepsilon, q') \in \Delta\} \cup \{(q, !!m, q') \in \Delta\}$ containing broadcasts and internal actions (called *active actions*) of $\mathcal{P}$ that start from state q. Furthermore, for each message $m \in \Sigma$, we denote by $R_m(q)$ the set $\{(q, ??m, q') \in \Delta\}$ containing the edges that start in state q and can be taken on reception of message m. We say that a broadcast protocol is *complete* if for every $q \in Q$ and every $m \in \Sigma$, $R_m(q) \neq \emptyset$. Whether protocols are complete or not may change the decidability status of the problems we consider (see Section 4).

We now define the semantics associated with such a protocol. It is common to represent the network topology by an undirected graph describing the communication links [7]. Since the topology may change at any time (such an operation is called reconfiguration), we decide here to simplify the notations by specifying, for each broadcast, a set of possible receivers that is chosen non-deterministically. The semantics of a network built over a broadcast protocol $\mathcal{P} = (Q, q_0, \Sigma, \Delta)$ is given by a transition system $\mathcal{T}_\mathcal{P} = (\Gamma, \Gamma_0, \to)$ where $\Gamma = \mathcal{V}_Q$ is the set of configurations (represented by vectors over Q); $\Gamma_0 = \mathcal{V}_{\{q_0\}}$ is the set of initial configurations and $\to \subseteq \Gamma \times \mathbb{N} \times \Delta \times 2^\mathbb{N} \times \Gamma$ is the transition relation defined as follows: $(\gamma, p, \delta, R, \gamma') \in \to$ (also denoted by $\gamma \xrightarrow{p,\delta,R} \gamma'$) iff $|\gamma| = |\gamma'|$ and $p \in [1..|\gamma|]$ and $R \subseteq [1..|\gamma|] \setminus \{p\}$ and one of the following conditions holds:

Internal action: $\delta = (\gamma[p], \varepsilon, \gamma'[p])$ and $\gamma'[p'] = \gamma[p']$ for all $p' \in [1..|\gamma|] \setminus \{p\}$ (*the p-th process performs an internal action*).

Communication: $\delta = (\gamma[p], !!m, \gamma'[p])$ and $(\gamma[p'], ??m, \gamma'[p']) \in \Delta$ for all $p' \in R$ such that $R_m(\gamma[p']) \neq \emptyset$, and $\gamma'[p''] = \gamma[p'']$ for all $p'' \in [1..|\gamma|] \setminus (R \cup \{p\})$ and for all $p'' \in R$ such that $R_m(\gamma[p'']) = \emptyset$ (*the p-th process broadcasts m to all the processes in the reception set R*).

Obviously, when an internal action is performed, the reception set R is not taken into account. We point out the fact that the hypothesis $|\gamma| = |\gamma'|$ implies that the number of processes remains constant during an execution (there is no creation or deletion of processes). Yet, $\mathcal{T}_\mathcal{P}$ is an infinite state transition system since the number of possible initial configurations is infinite. An *execution* of $\mathcal{P}$ is then a finite sequence of consecutive transitions in $\mathcal{T}_\mathcal{P}$ of the form $\theta = \gamma_0 \xrightarrow{p_0,\delta_0,R_0} \gamma_1 \ldots \xrightarrow{p_\ell,\delta_\ell,R_\ell} \gamma_{\ell+1}$ and we denote by $\Theta[\mathcal{P}]$ (or simply Θ when $\mathcal{P}$ is clear from context) the set of all executions of $\mathcal{P}$. Furthermore, we use $nbproc(\theta) = |\gamma_0|$ to represent the number of processes involved in the execution θ.

2.2 Local strategies and clique executions

Our goal is to analyze executions of broadcast protocols under *local strategies*, where each process performs the same choices of edges according to its past history (*i.e.* according to the edges of the protocol it has fired so far).

A *finite path* in $\mathcal{P}$ is either the empty path, denoted by ϵ, or a non-empty finite sequence of edges $\delta_0 \cdots \delta_\ell$ such that δ_0 starts in q_0 and for all $i \in [1..\ell]$, δ_i starts in the state in which δ_{i-1} ends. For convenience, we say that ϵ ends in state q_0. We write $\mathsf{Path}(\mathcal{P})$ for the set of all finite paths in $\mathcal{P}$.

For an execution $\theta \in \Theta[\mathcal{P}]$, we define, for every $p \in [1..nbproc(\theta)]$, the *past* of process p in θ (also referred to as its *history*), written $\pi_p(\theta)$, as the finite path in $\mathcal{P}$ that stores the sequences of edges of $\mathcal{P}$ taken by p along θ. We can now define local strategies which allow us to focus on the executions in which each process performs the same choice according to its past. A *local strategy* σ for $\mathcal{P}$ is a pair (σ_a, σ_r) of functions specifying, given a history, the next active action to be taken, and the reception edge to choose when receiving a message, respectively. Formally $\sigma_a : \mathsf{Path}(\mathcal{P}) \to (Q \times (\{!!m \mid m \in \Sigma\} \cup \{\varepsilon\}) \times Q)$ satisfies, for every $\rho \in \mathsf{Path}(\mathcal{P})$ ending in $q \in Q$, either $A(q) = \emptyset$ or $\sigma_a(\rho) \in A(q)$. Whereas $\sigma_r : \mathsf{Path}(\mathcal{P}) \times \Sigma \to (Q \times \{??m \mid m \in \Sigma\} \times Q)$ satisfies, for every $\rho \in \mathsf{Path}(\mathcal{P})$ ending in $q \in Q$ and every $m \in \Sigma$, either $R_m(q) = \emptyset$ or $\sigma_r(\rho, m) \in R_m(q)$.

Since our aim is to analyze executions where each process behaves according to the same local strategy, we now provide the formal definition of such executions. Given a local strategy σ, we say that a path $\delta_0 \cdots \delta_\ell$ *respects* σ if for all $i \in [0..\ell-1]$, we have $\delta_{i+1} = \sigma_a(\delta_0 \dots \delta_i)$ or $\delta_{i+1} = \sigma_r(\delta_0 \cdots \delta_i, m)$ for some $m \in \Sigma$. Following this, an execution θ respects σ if for all $p \in [1..nbproc(\theta)]$, we have that $\pi_p(\theta)$ respects σ (i.e. we have that each process behaves as dictated by σ). Finally we define $\Theta_{\mathcal{L}} \subseteq \Theta$ as the set of *local executions* (also called local semantics), that is executions θ respecting a local strategy.

We also consider another set of executions where we assume that every message is broadcast to all the processes of the network (apart from the emitter). Formally, an execution $\theta = \gamma_0 \xrightarrow{p_0,\delta_0,R_0} \dots \xrightarrow{p_\ell,\delta_\ell,R_\ell} \gamma_{\ell+1}$ is said to be a *clique execution* if $R_k = [1, \dots, nbproc(\theta)] \setminus \{p_k\}$ for every $k \in [0..\ell]$. We denote by $\Theta_{\mathcal{C}}$ the set of clique executions (also called clique semantics). Note that clique executions of broadcast networks have been studied in [9] and that such networks correspond to broadcast protocols with no rendez-vous [11]. We will also consider the intersection of these subsets of executions and write $\Theta_{\mathcal{LC}}$ for the set $\Theta_{\mathcal{L}} \cap \Theta_{\mathcal{C}}$ of clique executions which respect a local strategy.

2.3 Verification problems

In this work we study the parameterized verification of the reachability and target properties for broadcast protocols restricted to local strategies. The first one asks whether there exists an execution respecting some local strategy and that eventually reaches a configuration where a given control state appears, whereas the latter problem seeks for an execution respecting some local strategy and that ends in a configuration where all the control states belong to a given target set. We consider several variants of these problems depending on whether we restrict to clique executions or not and to complete protocols or not.

For an execution $\theta = \gamma_0 \xrightarrow{p_0,\delta_0,R_0} \gamma_1 \dots \xrightarrow{p_\ell,\delta_\ell,R_\ell} \gamma_{\ell+1}$, we denote by $\mathsf{End}(\theta) = \{\gamma_{\ell+1}[p] \mid p \in [1..nbproc(\theta)]\}$ the set of states that appear in the last configuration of θ. Reach$[\mathcal{S}]$, the parameterized reachability problem for executions restricted to $\mathcal{S} \in \{\mathcal{L}, \mathcal{C}, \mathcal{LC}\}$ is defined as follows:

Input: A broadcast protocol $\mathcal{P} = (Q, q_0, \Sigma, \Delta)$ and a control state $q_F \in Q$.
Output: Does there exist an execution $\theta \in \Theta_{\mathcal{S}}$ such that $q_F \in \mathsf{End}(\theta)$?

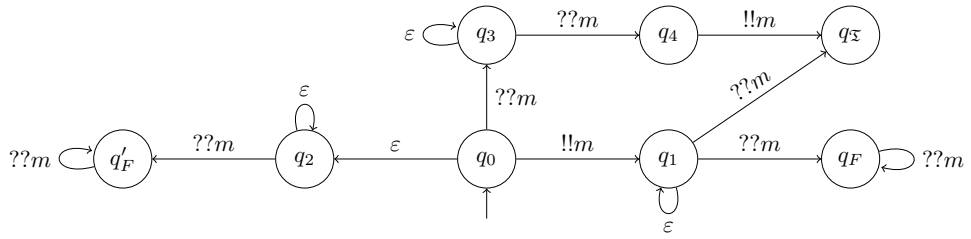


Figure 1 Example of a broadcast protocol.

In previous works, the parameterized reachability problem has been studied without the restriction to local strategies; in particular the reachability problem on unconstrained executions is in PTIME [7] and REACH[$\mathcal{C}$] is decidable and Non-Primitive Recursive (NPR) [9, 11] (it is in fact Ackermann-complete [16]).

TARGET[$\mathcal{S}$], the parameterized target problem for executions restricted to $\mathcal{S} \in \{\mathcal{L}, \mathcal{C}, \mathcal{LC}\}$ is defined as follows:

Input: A broadcast protocol $\mathcal{P} = (Q, q_0, \Sigma, \Delta)$ and a set of control states $\mathfrak{T} \subseteq Q$.

Output: Does there exist an execution $\theta \in \Theta_{\mathcal{S}}$ such that $\mathsf{End}(\theta) \subseteq \mathfrak{T}$?

It has been shown that a generalization of the target problem, without restriction to local strategies, can be solved in NP [7]. In this work, we focus on executions under local strategies and we obtain the results presented in the following table:

REACH[$\mathcal{L}$]	REACH[$\mathcal{LC}$]	TARGET[$\mathcal{L}$]	TARGET[$\mathcal{LC}$]
NP-complete [Thm. 3]	Undecidable [Thm. 5] Decidable and NPR for complete protocols [Thm. 7]	NP-complete [Thm. 4]	Undecidable [Thm. 5]

Most of the problems listed in the above table are monotone: if, in a network of a given size, an execution satisfying the reachability or target property exists, then, in any bigger network, there also exists an execution satisfying the same property. Let θ be an execution in $\Theta_{\mathcal{L}}$ [resp. $\Theta_{\mathcal{LC}}$]. For every $N \geq nbproc(\theta)$, there exists θ' in $\Theta_{\mathcal{L}}$ [resp. $\Theta_{\mathcal{LC}}$] such that $nbproc(\theta') = N$ and $\mathsf{End}(\theta) = \mathsf{End}(\theta')$ [resp. $\mathsf{End}(\theta) \subseteq \mathsf{End}(\theta')$]. This monotonicity property allows us to look for cutoffs, *i.e.* minimal number of processes such that a local execution with a given property exists. In this work, we provide upper-bounds on these cutoffs for REACH[$\mathcal{L}$] (Proposition 3.1) and TARGET[$\mathcal{L}$] (Theorem 4.2). For REACH[$\mathcal{LC}$] restricted to complete protocols, given the complexity of the problem, such an upper-bound would be non-primitive recursive and thus would not be of any practical use.

2.4 Illustrative example

To illustrate the notions of local strategies and clique executions, we provide an example of a broadcast protocol in Fig. 1. On this protocol no clique execution can reach state q_F: as soon as a process in q_0 sends message m, all the other processes in q_0 receive this message, and move to q_3, because of the clique topology. An example of a clique execution is: $(q_0, q_0, q_0, q_0) \to (q_1, q_3, q_3, q_3)$ (where we omit the labels over $\to$). However, there exists a local execution reaching q_F: $(q_0, q_0) \to (q_1, q_0) \to (q_F, q_1)$. This execution respects a local strategy since, from q_0 with empty past, the first process chooses the edge broadcasting m with empty reception set and in the next step the second process, also with empty past, performs the same action, broadcasting the message m to the first process. On the other hand, no local strategy permits to reach q'_F. Indeed, intuitively, to reach q'_F, in state q_0

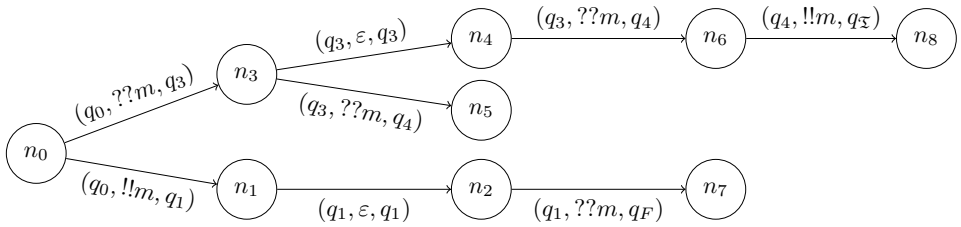


Figure 2 A strategy pattern for the broadcast protocol depicted Fig. 1.

one process with empty past needs to go to q_1 and another one to q_2, which is forbidden by locality. Finally $(q_0, q_0, q_0) \to (q_1, q_0, q_3) \to (q_1, q_1, q_4) \to (q_{\mathfrak{T}}, q_{\mathfrak{T}}, q_{\mathfrak{T}})$ is a local execution that targets the set $\mathfrak{T} = \{q_{\mathfrak{T}}\}$.

3 Verification problems for local executions

We begin with studying the parameterized reachability and target problems under local executions, *i.e.* we seek for a local strategy ensuring either to reach a specific control state, or to reach a configuration in which all the control states belong to a given set.

3.1 Solving Reach[$\mathcal{L}$]

To obtain an NP-algorithm for Reach[$\mathcal{L}$], we prove that there exists a local strategy to reach a specific control state if and only if there is a local strategy which can be represented thanks to a finite tree of polynomial size; the idea behind such a tree being that the paths in the tree represent past histories and the edges outgoing a specific node represent the decisions of the local strategy. The NP-algorithm will then consist in guessing such finite tree of polynomial size and verifying if it satisfies some conditions needed to reach the specified control state.

Representing strategies with trees. We now define our tree representation of strategies called strategy patterns, which are standard labelled trees with labels on the edges. Intuitively a strategy pattern defines, for some of the paths in the associated protocol, the active action and receptions to perform.

A *strategy pattern* for a broadcast protocol $\mathcal{P} = (Q, q_0, \Sigma, \Delta)$ is a labelled tree $T = (N, n_0, E, \Delta, \mathsf{lab})$ with N a finite set of nodes, $n_0 \in N$ the root, $E \subseteq N \times N$ the edge relation and $\mathsf{lab} : E \to \Delta$ the edge-labelling function. Moreover T is such that if $e_1 \cdots e_\ell$ is a path in T, then $\mathsf{lab}(e_1) \cdots \mathsf{lab}(e_\ell) \in \mathsf{Path}(\mathcal{P})$, and for every node $n \in N$: there is at most one edge $e = (n, n') \in E$ such that $\mathsf{lab}(e)$ is an active action; and, for each message m, there is at most one edge $e = (n, n') \in E$ such that $\mathsf{lab}(e)$ is a reception of m.

Since all labels of edges outgoing a node share a common source state (due to the hypothesis on labelling of paths), the labelling function lab can be consistently extended to nodes by letting $\mathsf{lab}(n_0) = q_0$ and $\mathsf{lab}(n) = q$ for any $(n', n) \in E$ with $\mathsf{lab}((n', n)) = (q', a, q)$.

The strategy pattern represented in Fig. 2, for the broadcast protocol from Fig. 1, illustrates that strategy patterns somehow correspond to under-specified local strategies. For example, from node n_1 (labelled by q_1) no reception of message m is specified, and from node n_5 (labelled by q_4) no reception and no active action are specified.

More generally, given $\mathcal{P}$ a broadcast protocol, and T a strategy pattern for $\mathcal{P}$ with edge-labelling function lab, a local strategy $\sigma = (\sigma_a, \sigma_r)$ for $\mathcal{P}$ is said to *follow* T if for every

path $e_1 \cdots e_\ell$ in T, the path $\rho = \mathsf{lab}(e_1) \cdots \mathsf{lab}(e_\ell)$ in $\mathcal{P}$ respects σ. Notice that any strategy pattern admits at least one local strategy that follows it.

Reasoning on strategy patterns. We now show that one can test directly on a strategy pattern whether the local strategies following it can yield an execution reaching a specific control state. An *admissible strategy pattern* for $\mathcal{P} = (Q, q_0, \Sigma, \Delta)$ is a pair $(T, \prec)$ where $T = (N, n_0, E, \Delta, \mathsf{lab})$ is a strategy pattern for $\mathcal{P}$ and $\prec \subseteq N \times N$ is a strict total order on the nodes of T such that:

(1) for all $(n, n') \in E$ we have $n \prec n'$;

(2) for all $e = (n, n') \in E$, if $\mathsf{lab}(e) = (\mathsf{lab}(n), ??m, \mathsf{lab}(n'))$ for some $m \in \Sigma$, then there exists $e_1 = (n_1, n_1')$ in E such that $n_1' \prec n'$ and $\mathsf{lab}(e_1) = (\mathsf{lab}(n_1), !!m, \mathsf{lab}(n_1'))$.

In words, (1) states that $\prec$ respects the natural order on the tree and (2) that every node corresponding to a reception of m should be preceded by a node corresponding to a broadcast of m.

The example of strategy pattern on Fig. 2 is admissible with the order $n_i \prec n_j$ if $i < j$, whereas for any order including $n_3 \prec n_1$ it is not admissible (a broadcast of m should precede n_3). In general, given a strategy pattern T and a strict total order $\prec$, checking whether $(T, \prec)$ is admissible can be done in polynomial time (in the size of the pattern).

In order to state the relation between admissible strategy patterns and local strategies, we define $\mathsf{lab}(T) = \{\mathsf{lab}(n) \mid n \in N\}$ as the set of control states labelling nodes of T and $\mathsf{Occur}(\theta) = \{\gamma_i[p] \mid i \in [0..\ell+1] \text{ and } p \in [1..nbproc(\theta)]\}$ as the set of states that appear along an execution $\theta = \gamma_0 \to \cdots \to \gamma_{\ell+1}$. The next proposition tells us that admissible strategy patterns are necessary and sufficient to represent the sets of states that can be reached under local strategies. For all $Q' \subseteq Q$, there exists an admissible strategy pattern $(T, \prec)$ such that $\mathsf{lab}(T) = Q'$ iff there exists a local strategy σ and an execution θ such that θ respects σ and $Q' = \mathsf{Occur}(\theta)$, furthermore σ follows T.

Minimizing admissible strategy patterns. For $(T, \prec)$ an admissible strategy pattern, we denote by $\mathsf{last}(T, \prec)$ the maximal node w.r.t. $\prec$ and we say that $(T, \prec)$ is q_F*-admissible* if $\mathsf{lab}(\mathsf{last}(T, \prec)) = q_F$. We now show that there exist polynomial size witnesses of q_F-admissible strategy patterns. The idea is to keep only relevant edges that either lead to a node labelled by q_F or that permit a broadcast of a new message. Intuitively, a *minimal* strategy pattern guarantees that (1) there is a unique node labelled with q_F, (2) in every subtree there is either a node labelled by q_F or a broadcast of a new message (*i.e.* a broadcast of a message that has not been seen previously with respect to the order $\prec$), and (3) a path starting and ending in two different nodes labelled by the same state, cannot be compressed without losing a new broadcast or a path towards q_F (by compressing we mean replacing the first node on the path by the last one). These hypotheses allow us to seek only for q_F-admissible strategy patterns of polynomial size.

If there exists a q_F-admissible strategy pattern for $\mathcal{P}$, then there is one of size at most $(2|\Sigma| + 1) \cdot (|Q| - 1)$ and of height at most $(|\Sigma| + 1) \cdot |Q|$.

By Proposition 3.1, there exists an execution $\theta \in \Theta_{\mathcal{L}}$ such that $q_F \in \mathsf{Occur}(\theta)$ iff there exists a q_F-admissible strategy pattern and thanks to Proposition 3.1 it suffices to look only for q_F-admissible strategy patterns of size polynomial in the size of the broadcast protocol. A non-deterministic polynomial time algorithm for Reach[$\mathcal{L}$] consists then in guessing a strategy pattern of polynomial size and an order and then verifying whether it is q_F-admissible.

▶ **Theorem 2.** Reach[$\mathcal{L}$] *is in* NP.

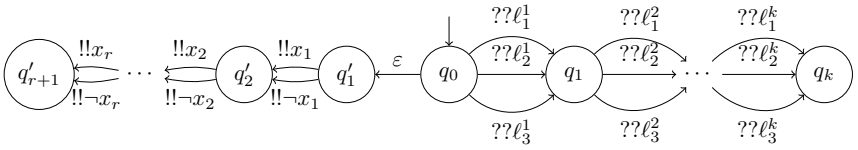


Figure 3 Encoding a 3-SAT formula into a broadcast protocol.

We can furthermore provide bounds on the minimal number of processes and on the memory needed to implement local strategies. Given a q_F-admissible strategy pattern one can define an execution following the pattern such that each reception edge of the pattern is taken exactly once and active actions may be taken multiple times but in a row. Such an execution needs at most one process per reception edge. Together with the bound on the size of the minimal strategy patterns (see Proposition 3.1), this yields a cutoff property on the minimal size of network to reach the final state. Moreover the past history of every process in this execution is bounded by the depth of the tree, hence we obtain an upper bound on the size of the memory needed by each process for Reach[$\mathcal{L}$].

If there exists an execution $\theta \in \Theta_{\mathcal{L}}$ such that $q_F \in \mathsf{Occur}(\theta)$, then there exists an execution $\theta' \in \Theta_{\mathcal{L}}$ such that $q_F \in \mathsf{Occur}(\theta')$ and $nbproc(\theta') \leq (2|\Sigma|+1)\cdot(|Q|-1)$ and $|\pi_p(\theta')| \leq (|\Sigma|+1)\cdot|Q|$ for every $p \in [1..nbproc(\theta')]$.

By reducing 3-SAT, one can furthermore show Reach[$\mathcal{L}$] to be NP-hard. Let $\phi = \bigwedge_{1\leq i\leq k}(\ell_1^i \vee \ell_2^i \vee \ell_3^i)$ be a 3-SAT formula such that $\ell_j^i \in \{x_1, \neg x_1, \ldots, x_r, \neg x_r\}$ for all $i \in [1..k]$ and $j \in \{1,2,3\}$. We build from ϕ the broadcast protocol $\mathcal{P}$ depicted at Fig. 3. Under this construction, ϕ is satisfiable iff there is an execution $\theta \in \Theta_{\mathcal{L}}$ such that $q_k \in \mathsf{Occur}(\theta)$. The local strategy hypothesis ensures that even if several processes broadcast a message corresponding to the same variable, all of them must take the same decision so that there cannot be any execution during which both x_i and $\neg x_i$ are broadcast. It is then clear that control state q_k can be reached if and only if each clause is satisfied by the set of broadcast messages. Together with Theorem 2, we obtain the precise complexity of Reach[$\mathcal{L}$].

▶ **Theorem 3.** Reach[$\mathcal{L}$] *is* NP*-complete.*

3.2 Solving Target[$\mathcal{L}$]

Admissible strategy patterns can also be used to obtain an NP-algorithm for Target[$\mathcal{L}$]. As we have seen, given an admissible strategy pattern, one can build an execution where the processes visit all the control states present in the pattern. When considering the target problem, one also needs to ensure that the processes can afterwards be directed to the target set. To guarantee this, it is possible to extend admissible strategy patterns with another order on the nodes which ensures that (a) from any node there exists a path leading to the target set and (b) whenever on this path a reception is performed, the corresponding message can be broadcast by a process that will only later on be able to reach the target.

We formalize now this idea. For $\mathfrak{T} \subseteq Q$ a set of states, a $\mathfrak{T}$*-coadmissible strategy pattern* for $\mathcal{P} = (Q, q_0, \Sigma, \Delta)$ is a pair $(T, \lhd)$ where $T = (N, n_0, E, \Delta, \mathsf{lab})$ is a strategy pattern for $\mathcal{P}$ and $\lhd \subseteq N \times N$ is a strict total order on the nodes T such that for every node $n \in N$ with $\mathsf{lab}(n) \notin \mathfrak{T}$ there exists an edge $e = (n, n') \in E$ with $n \lhd n'$ and either:

- $\mathsf{lab}(e) = (\mathsf{lab}(n), \varepsilon, \mathsf{lab}(n'))$ or,
- $\mathsf{lab}(e) = (\mathsf{lab}(n), !!m, \mathsf{lab}(n'))$ or,
- $\mathsf{lab}(e) = (\mathsf{lab}(n), ??m, \mathsf{lab}(n'))$ and there exists an edge $e_1 = (n_1, n'_1) \in E$ such that $n \lhd n_1$, $n \lhd n'_1$ and $\mathsf{lab}(e_1) = (q_1, !!m, q'_1)$.

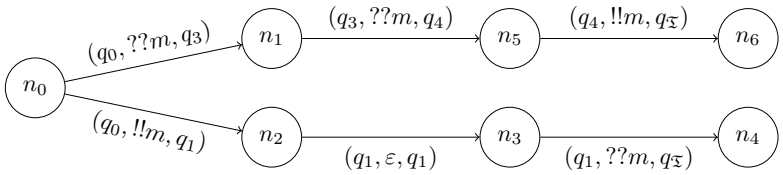


Figure 4 A $\mathfrak{T}$-coadmissible strategy pattern on the example protocol of Fig. 1.

Intuitively the order $\lhd$ in a $\mathfrak{T}$-coadmissible strategy pattern corresponds to the order in which processes must move along the tree towards the target; the conditions express that any node with label not in $\mathfrak{T}$ has an outgoing edge that is feasible. In particular, a reception of m is only feasible before all edges carrying the corresponding broadcast are disabled.

A strategy pattern T equipped with two orderings $\prec$ and $\lhd$ is said to be $\mathfrak{T}$-*biadmissible* whenever $(T, \prec)$ is admissible and $(T, \lhd)$ is $\mathfrak{T}$-coadmissible.

To illustrate the construction of $\mathfrak{T}$-coadmissble patterns, we give in Fig. 4 an example pattern, that, equipped with the natural order $n_i \lhd n_j$ iff $i < j$, is $\mathfrak{T}$-coadmissible for $\mathfrak{T} = \{q_\mathfrak{T}\}$. Indeed all leaves are labelled with a target state, and the broadcast edge $n_5 \xrightarrow{(q_4, !!m, q_\mathfrak{T})} n_6$ allows all processes to take the corresponding reception edges. This $\mathfrak{T}$-coadmissible pattern is in particular obtained from the execution $(q_0, q_0, q_0) \to (q_1, q_3, q_0) \to (q_1, q_3, q_0) \to (q_\mathfrak{T}, q_4, q_1) \to (q_\mathfrak{T}, q_4, q_1) \to (q_\mathfrak{T}, q_\mathfrak{T}, q_\mathfrak{T})$. Notice that $\lhd$ is not an admissible order, because $n_1 \lhd n_2$, however there are admissible orders for this pattern, for example the order $n_0 \prec n_2 \prec n_3 \prec n_4 \prec n_1 \prec n_5 \prec n_6$.

As for Reach[$\mathcal{L}$], one can show polynomial size witnesses of $\mathfrak{T}$-*biadmissible* strategy patterns exist, yielding an NP-algorithm for Target[$\mathcal{L}$]. Also, the size of minimal $\mathfrak{T}$-*biadmissible* strategy patterns gives here also a cutoff on the number of processes needed to satisfy the target condition, as well as an upper bound on the memory size.

▶ **Theorem 4.**

1. Target[$\mathcal{L}$] *is* NP*-complete.*
2. *If there exists an execution* $\theta \in \Theta_\mathcal{L}$ *such that* $\mathsf{End}(\theta) \subseteq \mathfrak{T}$*, then there exists an execution* $\theta' \in \Theta_\mathcal{L}$ *such that* $\mathsf{End}(\theta') \subseteq \mathfrak{T}$ *and* $nbproc(\theta') \leq 16|\Sigma| \cdot |Q| + 4|\Sigma| \cdot (|Q| - |\mathfrak{T}| + 1)$ *and* $|\pi_p(\theta')| \leq 4|\Sigma| \cdot |Q| + 2(|Q| - |\mathfrak{T}|) + 1$ *for every* $p \leq nbproc(\theta')$*.*

▶ Remark. The NP-hardness derives from the fact that the target problem is harder than the reachability problem. To reduce Reach[$\mathcal{L}$] to Target[$\mathcal{L}$], one can add the broadcast of a new message from q_F, and its reception from any state to q_F.

Another consequence of this simple reduction is that Target[$\mathcal{L}$] in NP yields another proof that Reach[$\mathcal{L}$] is in NP, yet the two proofs of NP-membership allowed us to give an incremental presentation, starting with admissible strategy patterns, and proceeding with co-admissible strategy patterns.

4 Verification problems for local clique executions

4.1 Undecidability of Reach[$\mathcal{LC}$] and Target[$\mathcal{LC}$]

Reach[$\mathcal{LC}$] and Target[$\mathcal{LC}$] happen to be undecidable and for the latter, even in the case of complete protocols. The proofs of these two results are based on a reduction from the halting problem of a two counter Minsky machine (a finite program equipped with two integer variables which can be incremented, decremented and tested to zero). The main idea

consists in both cases in isolating some processes to simulate the behavior of the machine while the other processes encode the values of the counters.

Thanks to the clique semantics we can in fact isolate one process. This is achieved by setting the first transition to be the broadcast of a message *start* whose reception makes all the other process change their state. Hence, thanks to the clique semantics, there is only one process that sends the message *start*, such process, called the controller, will be in charge of simulating the transitions of the Minsky machine. The clique semantics is also used to correctly simulate the increment and decrement of counters. For instance to increment a counter, the controller asks whether a process simulating the counter can be moved from state 0 to state 1 and if it is possible, relying on the clique topology only one such process changes its state (the value of the counter is then the number of processes in state 1). In fact, all the processes will receive the request, but the first one answering it, will force the other processes to come back to their original state, ensuring that only one process will move from state 0 to 1.

The main difficulty is that broadcast protocols (even under the clique semantics) cannot test the absence of processes in a certain state (which would be needed to simulate a test to 0 of one of the counters). Here is how we overcome this issue for Target[$\mathcal{LC}$]: the controller, when simulating a zero-test, sends all the processes with value 1 into a sink error state and the target problem allows to check for the reachability of a configuration with no process in this error state (and thus to test whether the controller has 'cheated', *i.e.* has taken a zero-test transition whereas the value of the associated counter was not 0). We point out that in this case, restricting to local executions is not necessary, we get in fact as well that Target[$\mathcal{C}$] is undecidable.

For Reach[$\mathcal{LC}$], the reduction is more tricky since we cannot rely on a target set of states to check that zero-test were faithfully simulated. Here in fact we will use two controllers. Basically, before sending a *start* message, some processes will be able to go to a waiting state (thanks to an internal transition) from which they can become controller and in which they will not receive any messages (this is where the protocol needs to be incomplete). Then we will use the locality hypothesis to ensure that two different controllers will simulate exactly the same run of the Minsky machine twice and with exactly the same number of processes encoding the counters. Restricting to local strategies guarantees the two runs to be identical, and the correctness derives from the fact that if in the first simulation the controller 'cheats' while performing a zero-test (and sending as before some processes encoding a counter value into a sink state), then in the second simulation, the number of processes encoding the counters will be smaller (due to the processes blocked in the sink state), so that the simulation will fail (because there will not be enough processes to simulate faithfully the counter values).

▶ **Theorem 5.** Reach*[$\mathcal{LC}$] is undecidable and* Target*[$\mathcal{LC}$] restricted to complete protocol is undecidable.*

The undecidability proof for Reach[$\mathcal{LC}$] strongly relies on the protocol being incomplete. Indeed, in the absence of specified receptions, the processes ignore broadcast messages and keep the same history, thus allowing to perform twice the same simulation of the run. In contrast, for complete protocols, all the processes are aware of all broadcast messages, therefore one cannot force the two runs to be identical. In fact, the reachability problem is decidable for complete protocols, as we shall see in the next section.

4.2 Decidability of Reach[$\mathcal{LC}$] for complete protocols

To prove the decidability of Reach[$\mathcal{LC}$] for complete protocols, we abstract the behavior of a protocol under local clique semantics by counting the possible number of different histories

in each control state.

We identify two cases when the history of processes can differ (under local clique semantics):

(1) When a process p performs a broadcast, its history is unique for ever (since all the other processes must receive the emitted message);

(2) A set of processes sharing the same history can be split when some of them perform a sequence of internal actions and the others perform only a prefix of that sequence.

From a complete broadcast protocol $\mathcal{P} = (Q, q_0, \Sigma, \Delta)$ we build an abstract transition system $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}} = (\Lambda, \lambda_0, \Rightarrow)$ where configurations count the number of different histories in each control state. More precisely the set of abstract configurations is $\Lambda = \mathcal{M}(Q \times \{\mathbf{m}, \mathbf{s}\} \times \{!!_{ok}, !!_{no}\}) \times \{\varepsilon, !!\}$. Abstract configurations are thus pairs where the first element is a multiset and the second element is a flag in $\{\varepsilon, !!\}$. The latter indicates the type of the next actions to be simulated (sequence of internal actions or broadcast): it prevents to simulate consecutively two incoherent sequences of internal actions (with respect to the local strategy hypothesis). For the former, an element $(q, \mathbf{s}, !!_{ok})$ in the multiset represents a single process (flag $\mathbf{s}$) in state q with a unique history which is allowed to perform a broadcast (flag $!!_{ok}$). An element $(q, \mathbf{m}, !!_{no})$ represents many processes (flag $\mathbf{m}$) in state q, all sharing the same unique history and none of them is allowed to perform a broadcast (flag $!!_{no}$). The initial abstract configuration λ_0 is then $(\{\{(q_0, \mathbf{m}, !!_{ok})\}\}, \varepsilon)$. In the sequel we will write $\mathbf{HM}$ for the set $\mathcal{M}(Q \times \{\mathbf{m}, \mathbf{s}\} \times \{!!_{ok}, !!_{no}\})$ of history multisets, so that $\Lambda = \mathbf{HM} \times \{\varepsilon, !!\}$, and typical elements of $\mathbf{HM}$ are denoted $\mathbb{M}$, $\mathbb{M}'$, etc.

In order to provide the definition of the abstract transition relation $\Rightarrow$, we need to introduce new notions, and notations. An ε-path ρ in $\mathcal{P}$ from q to q' is either the empty path (and in that case $q = q'$) or it is a non-empty finite path $\delta_0 \cdots \delta_n$ that starts in q, ends in q' and such that all the δ_i's are internal transitions.

An ε-path ρ in $\mathcal{P}$ is said to be a *prefix* of an ε-path ρ' if $\rho \neq \rho'$ and either ρ is the empty path or $\rho = \delta_0 \cdots \delta_n$ and $\rho' = \delta_0 \cdots \delta_n \delta_{n+1} \ldots \delta_{n+m}$ for some $m > 0$. Since we will handle multisets, let us give some convenient notations. Given E a set, and $\mathbb{M}$ a multiset over E, we write $\mathbb{M}(e)$ for the number of occurrences of element $e \in E$ in $\mathbb{M}$. Moreover, $card(\mathbb{M})$ stands for the cardinality of $\mathbb{M}$: $card(\mathbb{M}) = \sum_{e \in E} \mathbb{M}(e)$. Last, we will write $\oplus$ for the addition on multisets: $\mathbb{M} \oplus \mathbb{M}'$ is such that for all $e \in E$, $(\mathbb{M} \oplus \mathbb{M}')(e) = \mathbb{M}(e) + \mathbb{M}'(e)$.

The abstract transition relation $\Rightarrow \in \Lambda \times \Lambda$ is composed of two transitions relations: one simulates the broadcast of messages and the other one sequences of internal transitions. This will guarantee an alternation between abstract configurations flagged with ε and the ones flagged with $!!$. Let us first define $\Rightarrow_{!!} \subseteq (\mathbf{HM} \times \{!!\}) \times (\mathbf{HM} \times \{\varepsilon\})$ which simulates a broadcast. We have $(\mathbb{M}, !!) \Rightarrow_{!!} (\mathbb{M}', \varepsilon)$ iff there exists $(q_1, !!m, q_2) \in \Delta$ and $\mathit{fl}_1 \in \{\mathbf{s}, \mathbf{m}\}$ such that

1. $\mathbb{M}(q_1, \mathit{fl}_1, !!_{ok}) > 0$

2. there exists a family of functions G indexed by $(q, \mathit{fl}, b) \in Q \times \{\mathbf{m}, \mathbf{s}\} \times \{!!_{ok}, !!_{no}\}$, such that $G_{(q,\mathit{fl},b)} : [1..\mathbb{M}(q, \mathit{fl}, b)] \to \mathbf{HM}$, and:

$$\mathbb{M}' = \{\{q_2, \mathbf{s}, !!_{ok}\}\} \oplus \bigoplus_{\{(q,\mathit{fl},b) \mid \mathbb{M}(q,\mathit{fl},b) \neq 0\}} \bigoplus_{i \in [1..\mathbb{M}(q,\mathit{fl},b)]} G_{(q,\mathit{fl},b)}(i)$$

and such that for each (q, fl, b) verifying $\mathbb{M}(q, \mathit{fl}, b) \neq 0$, for all $i \in [1..\mathbb{M}(q, \mathit{fl}, b)]$, the following conditions are satisfied:

a. if $\mathit{fl}_1 = \mathbf{s}$, $card(G_{(q_1,\mathit{fl}_1,!!_{ok})}(1)) = 0$ and if $\mathit{fl}_1 = \mathbf{m}$, then there exists $q' \in Q$ such that $G_{(q_1,\mathit{fl}_1,!!_{ok})}(1) = \{\{(q', \mathit{fl}_1, !!_{ok})\}\}$ and such that $(q, ??m, q') \in \Delta$;

b. if $(q, \mathit{fl}, b) \neq (q_1, \mathit{fl}_1, !!_{ok})$ or $i \neq 1$, then there exists $q' \in Q$ such that $G_{(q,\mathit{fl},b)}(i) = \{\{(q', \mathit{fl}, !!_{ok})\}\}$ and such that $(q, ??m, q') \in \Delta$.

Intuitively to provide the broadcast, we need to find a process which is 'allowed' to perform a broadcast and which is hence associated with an element $(q_1, \mathit{fl}_1, !!_{ok})$ in $\mathbb{M}$. The transition $(q_1, !!m, q_2)$ tells us which broadcast is simulated. Then the functions $G_{(q,\mathit{fl},b)}$ associate with each element of the multiset $\mathbb{M}$ of the form (q, fl, b) a single element which can be reached thanks to a reception of the message m. Of course this might not hold for an element of the shape $(q_1, \mathbf{s}, !!_{ok})$ if it is the one chosen to do the broadcast since it represents a single process, and hence this element moves to q_2. Note however that if $\mathit{fl}_1 = \mathbf{m}$, then $(q_1, \mathbf{m}, !!_{ok})$ represents many processes, hence the one which performs the broadcast is isolated, but the many other ones have to be treated for reception of the message. Note also that we use here the fact that since an element $(q, \mathbf{m}, b)$ represents many processes with the same history, all these processes will behave the same way on reception of the message m.

We now define $\Rightarrow_\varepsilon \subseteq (\mathbf{HM} \times \{\varepsilon\}) \times (\mathbf{HM} \times \{!!\})$ which simulates the firing of sequences of ε-transitions. We have $(\mathbb{M}, \varepsilon) \Rightarrow_\varepsilon (\mathbb{M}', !!)$ iff there exists a family of functions F indexed by $(q, \mathit{fl}, b) \in Q \times \{\mathbf{m}, \mathbf{s}\} \times \{!!_{ok}, !!_{no}\}$, such that $F_{(q,\mathit{fl},b)} : [1..\mathbb{M}(q, \mathit{fl}, b)] \to \mathbf{HM}$, and

$$\mathbb{M}' = \bigoplus_{\{(q,\mathit{fl},b) \mid \mathbb{M}(q,\mathit{fl},b) \neq 0\}} \; \bigoplus_{i \in [1..\mathbb{M}(q,\mathit{fl},b)]} F_{(q,\mathit{fl},b)}(i)$$

and such that for each (q, fl, b) verifying $\mathbb{M}(q, \mathit{fl}, b) \neq 0$, for all $i \in [1..\mathbb{M}(q, \mathit{fl}, b)]$, we have:

1. $\mathit{card}(F_{(q,\mathit{fl},b)}(i)) \geq 1$ and if $\mathit{fl} = \mathbf{s}$, $\mathit{card}(F_{(q,\mathit{fl},b)}(i)) = 1$;
2. If $F_{(q,\mathit{fl},b)}(i)(q', \mathit{fl}', b') \neq 0$, then $\mathit{fl}' = \mathit{fl}$;
3. There exists a pair $(q_{!!}, \mathit{fl}_{!!}) \in Q \times \{\mathbf{m}, \mathbf{s}\}$ such that:
   - $F_{(q,\mathit{fl},b)}(i)(q_{!!}, \mathit{fl}_{!!}, !!_{ok}) = 1$
   - for all $(q', \mathit{fl}') \neq (q_{!!}, \mathit{fl}_{!!})$ $F_{(q,\mathit{fl},b)}(i)(q', \mathit{fl}', !!_{ok}) = 0$;
   - There exists a ε-path $\rho_{!!}$ from q to $q_{!!}$.
4. For all (q', fl') such that $F_{(q,\mathit{fl},b)}(i)(q', \mathit{fl}', !!_{no}) = k > 0$, there exists k different ε-paths (strict) prefix of $\rho_{!!}$ from q to q'.

Intuitively the functions $F_{(q,\mathit{fl},b)}$ associate with each element (q, fl, b) of the multiset $\mathbb{M}$ a set of elements that can be reached via internal transitions. We recall that each such element represents a set (or a singleton if $\mathit{fl} = \mathbf{s}$) of processes sharing the same history. Condition 1. states that if there are multiple processes ($\mathit{fl} = \mathbf{m}$) then they can be matched to more states in the protocol, but if it is single ($\mathit{fl} = \mathbf{s}$) it should be matched by an unique state. Condition 2. expresses that if an element in $\mathbb{M}$ represents many processes, then all its images represent as well many processes. Conditions 3. and 4. deal with the locality assumption. Precisely, condition 3. states that among all the elements of $\mathbb{M}'$ associated with an element of $\mathbb{M}$, one and only one should be at the end of a ε-path, and only one process associated with this element will be allowed to perform a broadcast. This justifies the use of the flag $!!_{ok}$. Last, condition 4. concerns all the other elements associated to this element of $\mathbb{M}$: their flag is set to $!!_{no}$ (they cannot perform a broadcast, because the local strategy will force them to take an internal transition), and their state should be on the previously mentioned ε-path.

As announced, we define the abstract transitive relation by $\Rightarrow = \Rightarrow_\varepsilon \cup \Rightarrow_{!!}$. Note that by definition we have a strict alternation of transitions of the type $\Rightarrow_\varepsilon$ and of the type $\Rightarrow_{!!}$. An *abstract local clique execution* of $\mathcal{P}$ is then a finite sequence of consecutive transitions in $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}}$ of the shape $\xi = \lambda_0 \Rightarrow \lambda_1 \cdots \Rightarrow \lambda_{\ell+1}$. As for concrete executions, if $\lambda_{\ell+1} = (\mathbb{M}_{\ell+1}, t_{\ell+1})$ we denote by $\mathsf{End}(\xi) = \{q \mid \exists \mathit{fl} \in \{\mathbf{m}, \mathbf{s}\}. \exists b \in \{!!_{ok}, !!_{no}\}. \mathbb{M}_{\ell+1}(q, \mathit{fl}, b) > 0\}$ the set of states that appear in the end configuration of ξ.

As an example, a possible abstract execution of the broadcast protocol from Fig. 1 is: $(\{\{(q_0, \mathbf{m}, !!_{ok})\}\}, \varepsilon) \Rightarrow (\{\{(q_0, \mathbf{m}, !!_{no}), (q_2, \mathbf{m}, !!_{no}), (q_2, \mathbf{m}, !!_{ok})\}\}, !!)$. This single-step

execution represents that among the processes in q_0, some processes will take an internal action to q_2 and loop there with another internal action (they are represented by the element $(q_2, \mathbf{m}, !!_{ok})$), others will only move to q_2 taking a single internal action (they are represented by $(q_2, \mathbf{m}, !!_{no})$), and finally some processes will stay in q_0 (they are represented by $(q_0, \mathbf{m}, !!_{no})$); note that these processes cannot perform a broadcast, because due to the local strategy hypothesis, they committed to firing the internal action leading to q_2.

Another example of an abstract execution is: $(\{\{(q_0, \mathbf{m}, !!_{ok})\}\}, \varepsilon) \Rightarrow (\{\{(q_0, \mathbf{m}, !!_{ok})\}\}, !!)$ $\Rightarrow (\{\{(q_1, \mathbf{s}, !!_{ok}), (q_3, \mathbf{m}, !!_{ok})\}\}, \varepsilon) \Rightarrow (\{\{(q_1, \mathbf{s}, !!_{ok}), (q_3, \mathbf{m}, !!_{no}), (q_3, \mathbf{m}, !!_{ok})\}\}, \varepsilon)$. Here in the first step, no process performs internal actions, in the second step one of the processes in q_0 broadcasts m, moves to q_1 and we know that no other process will ever share the same history, it is hence represented by $(q_1, \mathbf{s}, !!_{ok})$; then all the other processes with the same history represented by $(q_0, \mathbf{m}, !!_{ok})$ must receive m and move to q_3, they are hence represented by $(q_3, \mathbf{m}, !!_{ok})$. The last step represents that some processes perform the internal action loop on q_3.

The definition of the abstract transition system $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}}$ ensures a correspondence between abstract local clique executions and local clique executions in $\mathcal{P}$. Formally:

▶ **Lemma 6.** *Let $q_F \in Q$. There exists an abstract local clique execution ξ of $\mathcal{P}$ such that $q_F \in \mathsf{End}(\xi)$ iff there exists a local clique execution $\theta \in \Theta_{\mathcal{LC}}$ such that $q_F \in \mathsf{End}(\theta)$.*

Given the abstract transition system $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}}$, in order to show that Reach$[\mathcal{LC}]$ is decidable, we then rely on the theory of well-structured transition systems [1, 13]. Indeed, the natural order on abstract configurations is a well-quasi-order compatible with the transition relation $\Rightarrow$ of $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}}$ (bigger abstract configurations simulate smaller ones) and one can compute predecessors of upward-closed sets of configurations. This allows us to conclude that, in $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}}$, the set of all predecessors of a configuration where q_F appears is effectively computable, so that we can decide whether q_F is reachable in $\mathcal{T}_{\mathcal{P}}^{\mathcal{LC}}$, hence, thanks to the previous lemma, in $\mathcal{P}$.

We also show that Reach$[\mathcal{LC}]$ is non-primitive recursive thanks to a PTIME reduction from Reach$[\mathcal{C}]$ (which is Ackermann-complete [16]) to Reach$[\mathcal{LC}]$. We exploit the fact that the only difference between the semantics $\mathcal{C}$ and $\mathcal{LC}$ is that in the latter, processes with the same history take the same decision. We simulate this in $\mathcal{C}$ with a gadget which assigns a different history to each individual process at the beginning of the protocol making hence the reachability problem for $\mathcal{C}$ equivalent to the one with $\mathcal{LC}$ semantics.

▶ **Theorem 7.** Reach*$[\mathcal{LC}]$ restricted to complete protocols is decidable and* NPR.

5 Conclusion

We considered reconfigurable broadcast networks under local strategies that rule out executions in which processes with identical local history behave differently. Under this natural assumption for distributed protocols, the reachability and target problems are NP-complete. Moreover, we gave polynomial bounds on the cutoff and on the memory needed by strategies. When the communication topology is a clique, both problems become undecidable. Decidability is recovered for reachability if we further assume that protocols are complete.

To the best of our knowledge, this is the first attempt to take into account the local viewpoint of the processes in parameterized distributed systems. It could be interesting to study how the method we propose in this work can be adapted to parameterized networks equipped with other means of communication (such as rendez-vous [14] or shared memory

[12]). In the future we also plan to deal with properties beyond simple reachability objectives, as for example linear or branching time properties.

References

1 Parosh A. Abdulla, Karlis Cerans, Bengt Jonsson, and Yih-Kuen Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Inf. Comput.*, 160(1-2):109–127, 2000.

2 Benjamin Aminof, Swen Jacobs, Ayrat Khalimov, and Sasha Rubin. Parameterized model checking of token-passing systems. In *Proc. of VMCAI'14*, volume 8318 of *LNCS*, pages 262–281, 2014.

3 Nathalie Bertrand, Paulin Fournier, and Arnaud Sangnier. Distributed local strategies in broadcast networks. Research report, Inria Rennes, July 2015. https://hal.inria.fr/hal-01170796.

4 Benedikt Bollig. Logic for communicating automata with parameterized topology. In *Proc. of CSL-LICS'14*, page 18. ACM, 2014.

5 Benedikt Bollig, Paul Gastin, and Jana Schubert. Parameterized verification of communicating automata under context bounds. In *Proc. of RP'14*, volume 8762 of *LNCS*, pages 45–57, 2014.

6 Edmund M. Clarke, Muralidhar Talupur, Tayssir Touili, and Helmut Veith. Verification by network decomposition. In *Proc. of CONCUR'04*, volume 3170 of *LNCS*, pages 276–291, 2004.

7 Giorgio Delzanno, Arnaud Sangnier, Riccardo Traverso, and Gianluigi Zavattaro. On the complexity of parameterized reachability in reconfigurable broadcast networks. In *Proc. of FSTTCS'12*, volume 18 of *LIPIcs*, pages 289–300. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2012.

8 Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized verification of ad hoc networks. In *Proc. of CONCUR'10*, volume 6269 of *LNCS*, pages 313–327. Springer, 2010.

9 Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. On the power of cliques in the parameterized verification of ad hoc networks. In *Proc. of FoSSaCS'11*, volume 6604 of *LNCS*, pages 441–455. Springer, 2011.

10 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In *Proc. of STACS'14*, volume 25 of *LIPIcs*, pages 1–10. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2014.

11 Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *Proc. of LICS'99*, pages 352–359. IEEE Computer Society, 1999.

12 Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. In *Proc. of CAV'13*, volume 8044 of *LNCS*, pages 124–140, 2013.

13 Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1-2):63–92, 2001.

14 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992.

15 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *Proc. of FOCS'90*, pages 746–757. IEEE Computer Society, 1990.

16 Sylvain Schmitz and Philippe Schnoebelen. The power of well-structured systems. In *Proc. of CONCUR'13*, volume 8052 of *LNCS*, pages 5–24. Springer, 2013.

A Framework for Transactional Consistency Models with Atomic Visibility

Andrea Cerone, Giovanni Bernardi, and Alexey Gotsman

IMDEA Software Institute, Madrid, Spain

Abstract

Modern distributed systems often rely on databases that achieve scalability by providing only weak guarantees about the consistency of distributed transaction processing. The semantics of programs interacting with such a database depends on its consistency model, defining these guarantees. Unfortunately, consistency models are usually stated informally or using disparate formalisms, often tied to the database internals. To deal with this problem, we propose a framework for specifying a variety of consistency models for transactions uniformly and declaratively. Our specifications are given in the style of weak memory models, using structures of events and relations on them. The specifications are particularly concise because they exploit the property of atomic visibility guaranteed by many consistency models: either all or none of the updates by a transaction can be visible to another one. This allows the specifications to abstract from individual events inside transactions. We illustrate the use of our framework by specifying several existing consistency models. To validate our specifications, we prove that they are equivalent to alternative operational ones, given as algorithms closer to actual implementations. Our work provides a rigorous foundation for developing the metatheory of the novel form of concurrency arising in weakly consistent large-scale databases.

1998 ACM Subject Classification C.2.4 Distributed Systems

Keywords and phrases Replication, Consistency models, Transactions

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.58

1 Introduction

To achieve availability and scalability, modern distributed systems often rely on *replicated databases*, which maintain multiple *replicas* of shared data. The database clients can execute transactions on the data at any of the replicas, which communicate changes to each other using message passing. For example, large-scale Internet services use data replicas in geographically distinct locations, and applications for mobile devices keep replicas locally as well as in the cloud to support offline use. Ideally, we want the concurrent and distributed processing in a replicated database to be transparent, as formalised by the classical notion of serialisability [20]: the database behaves as if it executed transactions serially on a non-replicated copy of the data. However, achieving this ideal requires extensive coordination between replicas, which slows down the database and even makes it unavailable if network connections between replicas fail [1]. For this reason, nowadays replicated databases often provide weaker consistency guarantees, which allow non-serialisable behaviours, called *anomalies*. For example, consider the following program issuing transactions concurrently:

$$\begin{array}{c}\mathsf{txn}\,\{\,x.\mathtt{write}(\mathit{post});\ y.\mathtt{write}(\mathit{empty})\,\}\ \ \|\ \ \mathsf{txn}\,\{\,u = x.\mathtt{read}();\ \ y.\mathtt{write}(\mathit{comment})\,\}\\ \|\ \ \mathsf{txn}\,\{\,v = x.\mathtt{read}();\ \ w = y.\mathtt{read}()\,\}\end{array} \tag{1}$$

where x, y are database objects and u, v, w local variables. In some databases the above program can execute so that the last transaction observes the *comment*, but not the *post*:

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 58–71

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

$u = post$, $v = empty$, $w = comment$. This result cannot be obtained by executing the three transactions in any sequence and, hence, is not serialisable. In an implementation it may arise if the first two transactions are executed at a replica r, and the third one at another replica r', and the messages carrying the updates by the first two transactions arrive to r' out of order.

The semantics of programs interacting with a replicated database thus depends on its *consistency model*, restricting the anomalies it can exhibit and, as a consequence, the possible performance optimisations in its implementation. Recent years have seen a plethora of proposals of consistency models for replicated databases [6, 19, 12, 24, 21, 13, 4] that make different trade-offs between consistency and performance. Unfortunately, these subtle models are usually specified informally or using disparate formalisms, often tied to database internals. Whereas some progress in formalising the consistency models has been recently made for replicated databases without transactions [11, 12], the situation is worse for databases providing these. The lack of a uniform specification formalism represents a major hurdle in developing the metatheory of the novel form of concurrency arising in weakly consistent replicated databases and, in particular, methods for formal reasoning about application programs using them.

To deal with this problem, we propose a framework to uniformly specify a variety of modern transactional consistency models. We apply the framework to specify six existing consistency models for replicated databases; the results are summarised in Figure 1, page 63. Specifications in our framework are declarative, i.e., they do not refer to the database internals and thus allow reasoning about the database behaviour at a higher abstraction level. To achieve this, we take an *axiomatic* approach similar to the one used to define the semantics of weak memory models of multiprocessors and shared-memory programming languages [3]: our specifications model database computations by *abstract executions*, which are structures of events and relations on them, reminiscent of event structures [26]. For example, Figure 3(a), page 63 gives an execution that could arise from the program (1). The boxes named T_1, T_2 and T_3 depict transactions, which are sequences of events ordered by the *program order* po, reflecting the program syntax. The *visibility* edges $T_1 \xrightarrow{\mathsf{VIS}} T_2$ and $T_2 \xrightarrow{\mathsf{VIS}} T_3$ mean that the transaction T_2 (T_3) is aware of the updates made by T_1 (T_2). Consistency models are specified by *consistency axioms*, constraining abstract executions; e.g., a consistency axiom may require the visibility relation to be transitive and thereby disallow the execution in Figure 3(a).

The key observation we exploit in our framework is that modern consistency models for replicated databases usually guarantee *atomic visibility*: either all or none of the events in a transaction can be visible to another transaction; it is the flexibility in *when* a transaction becomes visible that leads to anomalies. Thanks to atomic visibility, in abstract executions we can use relations on whole transactions (such as VIS in Figure 3(a)), rather than on separate events inside them, thereby achieving particularly concise specifications. We further illustrate the benefits of this form of the specifications by exploiting it to obtain sufficient and necessary conditions for *observational refinement* [16] between transactions. This allows replacing a transaction in an abstract execution by another one without invalidating the consistency axioms of a given model. One can think of our conditions as characterising the optimisations that the database can soundly perform inside a transaction due to its atomic visibility.

To ensure that our declarative axiomatic specifications indeed faithfully describe the database behaviour, we prove that they are equivalent to alternative *operational* ones, given as algorithms closer to actual implementations (Theorem 6, §4). This correspondence also highlights implementation features that motivate the form of the consistency axioms.

Our work systematises the knowledge about consistency models of replicated databases and provides insights into relationships between them (§3). The proposed specification framework also gives a basis to develop methods for reasoning about application programs using weakly consistent databases. Finally, our framework is an effective tool for exploring the space of consistency models, because their concise axiomatic specifications allow easily experimenting with alternative designs. In particular, our formalisation naturally suggests a new consistency model (§3).

2 Abstract Executions

We consider a database storing *objects* $\mathsf{Obj} = \{x, y, \ldots\}$, which for simplicity we assume to be integer-valued. Clients interact with the database by issuing `read` and `write` operations on the objects, grouped into *transactions*. We let $\mathsf{Op} = \{\mathtt{read}(x, n), \mathtt{write}(x, n) \mid x \in \mathsf{Obj}, n \in \mathbb{Z}\}$ describe the possible operation invocations: reading a value n from an object x or writing n to x.

To specify a consistency model, we need to define the set of all client-database interactions that it allows. We start by introducing structures for recording such interactions in a single database computation, called *histories*. In these, we denote operation invocations using *history events* of the form (ι, o), where ι is an identifier from a countably infinite set $\mathsf{EventId}$ and $o \in \mathsf{Op}$. We use e, f, g to range over history events. We let $\mathsf{WEvent}_x = \{(\iota, \mathtt{write}(x, n)) \mid \iota \in \mathsf{EventId}, n \in \mathbb{Z}\}$, define the set REvent_x of read events similarly, and let $\mathsf{HEvent}_x = \mathsf{REvent}_x \cup \mathsf{WEvent}_x$. A relation is a *total order* if it is transitive, irreflexive, and relates every two distinct elements one way or another.

▶ **Definition 1.** A *transaction* $T, S, \ldots$ is a pair (E, po), where $E \subseteq \mathsf{HEvent}$ is a finite, non-empty set of events with distinct identifiers, and the *program order* po is a total order over E. A *history* $\mathcal{H}$ is a (finite or infinite) set of transactions with disjoint sets of event identifiers.

All transactions in a history are assumed to be committed: to simplify presentation, our specifications do not constrain values read inside aborted or ongoing transactions.

To define the set of histories allowed by a given consistency model, we introduce *abstract executions*, which enrich histories with certain relations on transactions, declaratively describing how the database processes them. Consistency models are then defined by constraining these relations. We call a relation *prefix-finite*, if every element has finitely many predecessors in the transitive closure of the relation.

▶ **Definition 2.** An *abstract execution* is a triple $\mathcal{A} = (\mathcal{H}, \mathsf{VIS}, \mathsf{AR})$ where:
- *visibility* $\mathsf{VIS} \subseteq \mathcal{H} \times \mathcal{H}$ is a prefix-finite, acyclic relation; and
- *arbitration* $\mathsf{AR} \subseteq \mathcal{H} \times \mathcal{H}$ is a prefix-finite, total order such that $\mathsf{AR} \supseteq \mathsf{VIS}$.

We often write $T \xrightarrow{\mathsf{VIS}} S$ in lieu of $(T, S) \in \mathsf{VIS}$, and similarly for AR. Figure 3(a) gives an execution corresponding to the anomaly explained in §1. Informally, $T \xrightarrow{\mathsf{VIS}} S$ means that S is aware of T, and thus T's effects can influence the results of operations in S. In implementation terms, this may be the case if the updates performed by T have been delivered to the replica performing S; the prefix-finiteness requirement ensures that there may only be finitely many such transactions T. We call transactions unrelated by visibility *concurrent*. The relationship $T \xrightarrow{\mathsf{AR}} S$ means that the versions of objects written by S supersede those written by T; e.g., *comment* supersedes *empty* in Figure 3(a). The constraint $\mathsf{AR} \supseteq \mathsf{VIS}$ ensures that writes by a transaction T supersede those that T is aware of; thus AR essentially

orders writes only by concurrent transactions. In an implementation, arbitration can be established by assigning timestamps to transactions.

A consistency model specification is a set of *consistency axioms* Φ constraining executions. The model allows those histories for which there exists an execution that satisfies the axioms:

$$\mathsf{Hist}_\Phi = \{\mathcal{H} \mid \exists \mathsf{VIS}, \mathsf{AR}.\, (\mathcal{H}, \mathsf{VIS}, \mathsf{AR}) \models \Phi\}. \tag{2}$$

Our consistency axioms do not restrict the operations done by the database clients. We can obtain the set of histories produced by a particular program interacting with the database, such as (1), by restricting the above set, as is standard in weak memory model definitions [7].

3 Specifying Transactional Consistency Models

We now apply the concepts introduced to define several existing consistency models; see Figures 1–3. For a total order R and a set A, we let $\max_{\mathsf{R}}(A)$ be the element $u \in A$ such that $\forall v \in A.\, v = u \vee (v, u) \in \mathsf{R}$; if $A = \emptyset$, then $\max_{\mathsf{R}}(A)$ is undefined. In the following, the use of $\max_{\mathsf{R}}(A)$ in an expression implicitly assumes that it is defined. For a relation $\mathsf{R} \subseteq A \times A$ and an element $u \in A$, we let $\mathsf{R}^{-1}(u) = \{v \mid (v, u) \in \mathsf{R}\}$. We denote the sequential composition of relations R_1 and R_2 by $\mathsf{R}_1; \mathsf{R}_2$. We write _ for a value that is irrelevant and implicitly existentially quantified.

Baseline consistency model: Read Atomic. The weakest consistency model we consider, Read Atomic (Figure 1), is defined by the axioms Int and Ext (Figure 2), which determine the outcomes of reads in terms of the visibility and arbitration relations. Consistency models stronger than Read Atomic are defined by adding axioms that constrain these relations. The *internal consistency axiom* Int ensures that, within a transaction, the database provides sequential semantics: a read from an object returns the same value as the last write to or read from this object in the transaction. In particular, Int guarantees that, if a transaction writes to an object and then reads the object, then it will observe its last write. The axiom also disallows so-called *unrepeatable reads*: if a transaction reads an object twice without writing to it in-between, it will read the same value in both cases.

If a read is not preceded in the program order by an operation on the same object, then its value is determined in terms of writes by other transactions using the *external consistency axiom* Ext. The formulation of Ext relies on the following notation, defining certain attributes of a transaction $T = (E, \mathsf{po})$. We let $T \vdash \mathsf{Write}\ x : n$ if T writes to x and the last value written is n: $\max_{\mathsf{po}}(E \cap \mathsf{WEvent}_x) = (\_, \mathtt{write}(x, n))$. We let $T \vdash \mathsf{Read}\ x : n$ if T makes an *external* read from x, i.e., one before writing to x, and n is the value returned by the first such read: $\min_{\mathsf{po}}(E \cap \mathsf{HEvent}_x) = (\_, \mathtt{read}(x, n))$. In this case, Int ensures that n will be the result of all external reads from x in T. According to Ext, the value returned by an external read in T is determined by the transactions VIS-preceding T that write to x: if there are no such transactions, then T reads the initial value 0; otherwise it reads the final value written by the last such transaction in AR. (In examples we sometimes use initial values other than 0.) For example, the execution in Figure 3(a) satisfies Ext; if it included the edge $T_1 \xrightarrow{\mathsf{VIS}} T_3$, then Ext would force the read from x in T_3 to return *post*. The axiom Ext implies the absence of so-called *dirty reads*: a committed transaction cannot read a value written by an aborted or an ongoing transaction (which are not present in abstract executions), and a transaction cannot read a value that was overwritten by the transaction that wrote it (ensured by the definition of $T \vdash \mathsf{Write}\ x : n$). Finally, Ext guarantees *atomic visibility* of a transaction: either all or none of its writes can be visible to another transaction.

For example, Ext disallows the execution in Figure 3(b) and, in fact, any execution with the same history. This illustrates a *fractured reads* anomaly: T_1 makes *Alice* and *Bob* friends, but T_2 observes only one direction of the friendship relationship. Thus, the consistency guarantees provided by Read Atomic are useful because they allow maintaining integrity invariants, such as the symmetry of the friendship relation.

Stronger consistency models. Even though Read Atomic ensures that all writes by a transaction become visible together, it does not constrain *when* this happens. This leads to a number of anomalies, including the causality violation shown in Figure 3(a). We now consider stronger consistency models that provide additional guarantees about the visibility of transactions. We specify the first model of *causal consistency* by requiring VIS to be transitive (TransVis). This implies that transactions ordered by VIS (such as T_1 and T_2 in Figure 3(a)), are observed by others (such as T_3) in this order. Hence, the axiom TransVis disallows the anomaly in Figure 3(a).

Both Read Atomic and causal consistency can be implemented without requiring any coordination among replicas [6, 19]: a replica can decide to commit a transaction without consulting other replicas. This allows the database to stay available even during network failures. However, the above consistency models allow the *lost update* anomaly illustrated by the execution in Figure 3(c), which satisfies the axioms of causal consistency. This execution could arise from the code, also shown in the figure, that uses transactions T_1 and T_2 to make deposits into an account. The two transactions read the initial balance of the account and concurrently modify it, resulting in one deposit getting lost. The next consistency model we consider, parallel snapshot isolation, prohibits such anomalies in exchange for requiring replica coordination in its implementations [24]. We specify it by strengthening causal consistency with the axiom NoConflict, which does not allow transactions writing to the same object to be concurrent. This rules out any execution with the history in Figure 3(c): it forces T_1 and T_2 to be ordered by VIS, so that they cannot both read 0 from `acct`.

The axiom TransVis in causal consistency and parallel snapshot isolation guarantees that VIS-ordered transactions are observed by others in this order (cf. Figure 3(a)). However, the axiom allows two *concurrent* transactions to be observed in different orders, as illustrated by the *long fork* anomaly in Figure 3(d), allowed by both models. Concurrent transactions T_1 and T_2 write to x and y, respectively. A transaction T_3 observes the write to x, but not y, and a transaction T_4 observes the write to y, but not x. Thus, from the perspectives of T_3 and T_4, the writes of T_1 and T_2 happen in different orders.

The next pair of consistency models that we consider disallow this anomaly. We specify prefix consistency and snapshot isolation by strengthening causal consistency, respectively, parallel snapshot isolation, with the requirement that all transactions become visible throughout the system in the same order given by AR. This is formalised by the axiom Prefix: if T observes S, then it also observes all AR-predecessors of S. Since $\mathsf{AR} \supseteq \mathsf{VIS}$, Prefix implies TransVis. The axiom Prefix disallows any execution with the history in Figure 3(d): T_1 and T_2 have to be related by AR one way or another; but then by Prefix, either T_4 has to observe *post1* or T_3 has to observe *post2*.

Even though consistent prefix and snapshot isolation ensure that transactions become visible to others in the same order, they allow this to happen with a delay, caused by asynchronous propagation of updates in implementations. This leads to the *write skew* anomaly shown in Figure 3(e). Here each of T_1 and T_2 checks that the combined balance of two accounts exceeds 100 and, if so, withdraws 100 from one of them. Both transactions pass the checks and make the withdrawals from different accounts, resulting in the combined

Φ	Consistency model	Axioms (Figure 2)	Fractured reads	Causality violation	Lost update	Long fork	Write skew
RA	Read Atomic [6]	INT, EXT	✘	✓	✓	✓	✓
CC	Causal consistency [19, 12]	INT, EXT, TRANSVIS	✘	✘	✓	✓	✓
PSI	Parallel snapshot isolation [24, 21]	INT, EXT, TRANSVIS, NOCONFLICT	✘	✘	✘	✓	✓
PC	Prefix consistency [13]	INT, EXT, PREFIX	✘	✘	✓	✘	✓
SI	Snapshot isolation [8]	INT, EXT, PREFIX, NOCONFLICT	✘	✘	✘	✘	✓
SER	Serialisability [20]	INT, EXT, TOTALVIS	✘	✘	✘	✘	✘

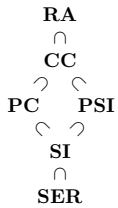


Figure 1 Consistency model definitions, anomalies and relationships.

$$\forall (E, \mathsf{po}) \in \mathcal{H}.\, \forall e \in E.\, \forall x, n.\, (e = (\_, \mathtt{read}(x, n)) \wedge (\mathsf{po}^{-1}(e) \cap \mathsf{HEvent}_x \neq \emptyset)) \implies \max_{\mathsf{po}}(\mathsf{po}^{-1}(e) \cap \mathsf{HEvent}_x) = (\_, \_(x, n)) \quad \text{(INT)}$$

$$\forall T \in \mathcal{H}.\, \forall x, n.\, T \vdash \mathsf{Read}\ x : n \implies ((\mathsf{VIS}^{-1}(T) \cap \{S \mid S \vdash \mathsf{Write}\ x : \_\} = \emptyset \wedge n = 0) \vee \max_{\mathsf{AR}}(\mathsf{VIS}^{-1}(T) \cap \{S \mid S \vdash \mathsf{Write}\ x : \_\}) \vdash \mathsf{Write}\ x : n) \quad \text{(EXT)}$$

VIS is transitive (TRANSVIS)	$\mathsf{AR}; \mathsf{VIS} \subseteq \mathsf{VIS}$ (PREFIX)	VIS is total (TOTALVIS)

$$\forall T, S \in \mathcal{H}.\, (T \neq S \wedge T \vdash \mathsf{Write}\ x : \_ \wedge S \vdash \mathsf{Write}\ x : \_) \implies (T \xrightarrow{\mathsf{VIS}} S \vee S \xrightarrow{\mathsf{VIS}} T) \quad \text{(NOCONFLICT)}$$

Figure 2 Consistency axioms, constraining an execution $(\mathcal{H}, \mathsf{VIS}, \mathsf{AR})$.

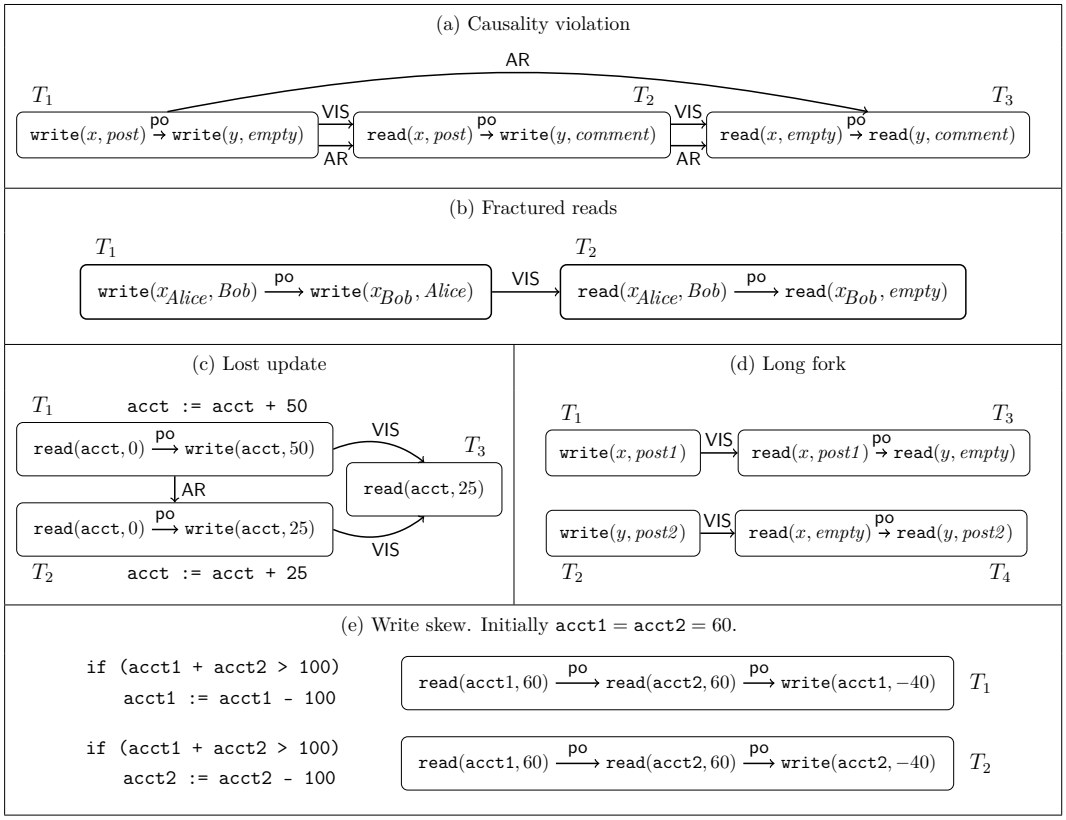


Figure 3 Executions illustrating anomalies allowed by different consistency models. The boxes group events into transactions. We sometimes omit irrelevant AR edges.

balance going negative. NoConflict allows the transactions to be concurrent, because they write to different objects.

Write skew and all the other anomalies mentioned above are disallowed by the classical consistency model of serialisability. Informally, a history is serialisable if the results of operations in it could be obtained by executing its (committed) transactions in some total order according to the usual sequential semantics. We formalise this in our framework by the axiom TotalVis, which requires the visibility relation VIS to be total. Since we always have $\mathsf{AR} \supseteq \mathsf{VIS}$, it is easy to see that there is no execution with the history in Figure 3(e) and a total VIS that would satisfy Ext.

Ramifications. The above specifications demonstrate the benefits of using our framework. First, the specifications are *declarative*, since they state constraints on database processing in terms of VIS and AR relations, rather than the database internals. The specifications thus allow checking whether a consistency model admits a given history solely in terms of these relations, as per (2).

The declarative nature of our specifications also provides a *better understanding* of consistency models. In particular, it makes apparent the relationships between different models and highlights the main mechanism of strengthening consistency—mandating that more edges be included into visibility.

▶ **Proposition 3.** *The strict inclusions between the consistency models in Figure 1 hold.*

The strictness of the inclusions in Figure 1 follows from the examples of histories in Figure 3.

Axiomatic specifications also provide an effective tool for *designing new consistency models*. For example, the existing consistency models do not include a counterpart of Read Atomic obtained by adding the NoConflict axiom. Such an "Update Atomic" consistency model would prevent lost update anomalies without having to enforce causal consistency (as in parallel snapshot isolation), which incurs performance overheads [5]. Update Atomic could be particularly useful when *mixed* with Read Atomic, so that the NoConflict axiom apply only to some transactions specified by the programmer. This provides a lightweight way of strengthening consistency where necessary.

Atomic visibility and observational refinement. Our specifications are particularly concise because they are tailored to consistency models providing atomic visibility. With axioms Int and Ext establishing this property, additional guarantees can be specified while abstracting from the internal events in transactions: solely in terms of VIS and AR relations on whole transactions and transaction attributes given by the $\vdash$-judgements. To further illustrate the benefits of this way of specification, we now exploit it to establish sufficient and necessary conditions for when one transaction *observationally refines* another, i.e., we can replace it in an execution without invalidating the consistency axioms. This notion is inspired by that of testing preorders in process algebras [16]. We can think of it as characterising the optimisations that the database can soundly perform inside the transaction due to its atomic visibility. As it happens, the conditions we establish differ subtly depending on the consistency model.

To formulate observational refinement, we introduce ***contexts*** $\mathcal{X}$—abstract executions with a hole $[\,]$ that represents a transaction with an unspecified behaviour: $\mathcal{X} = (\mathcal{H} \cup \{[\,]\}, \mathsf{VIS}, \mathsf{AR})$, where $\mathsf{VIS}, \mathsf{AR} \subseteq (\mathcal{H} \cup \{[\,]\}) \times (\mathcal{H} \cup \{[\,]\})$ satisfy the conditions in Definition 2. We can fill in the hole in the above context $\mathcal{X}$ by a transaction T, provided that the sets of event identifiers appearing in T and $\mathcal{H}$ are disjoint. This yields the abstract execution

$\mathcal{X}[T] = (\mathcal{H} \cup \{T\}, \mathsf{VIS}[[\,] \mapsto T], \mathsf{AR}[[\,] \mapsto T])$, where $\mathsf{VIS}[[\,] \mapsto T]$ treats T in the same way as VIS treats $[\,]$ and similarly for $\mathsf{AR}[[\,] \mapsto T]$ (we omit the formal definition to conserve space). We say that a transaction T_1 ***observationally refines*** a transaction T_2 on the consistency model Φ, written $T_1 \sqsubseteq_\Phi T_2$, if $\forall \mathcal{X}.\, \mathcal{X}[T_1] \models \Phi \implies \mathcal{X}[T_2] \models \Phi$.

▶ **Theorem 4.** *Let T_1, T_2 be such that $(\{T_1, T_2\}, \emptyset, \emptyset) \models \textsc{Int}$. We have $T_1 \sqsubseteq_{\mathbf{RA}} T_2$ if and only if for all x, n:*

$$\big(\neg(T_1 \vdash \mathsf{Read}\ x : n) \implies \neg(T_2 \vdash \mathsf{Read}\ x : n)\big) \wedge \big(T_1 \vdash \mathsf{Write}\ x : n \iff T_2 \vdash \mathsf{Write}\ x : n\big).$$

For $\Phi \in \{\mathbf{CC}, \mathbf{PC}, \mathbf{SER}\}$ we have $T_1 \sqsubseteq_\Phi T_2$ if and only if for all x, n, m, l:

$$\big(\neg(T_1 \vdash \mathsf{Read}\ x : n) \implies (\neg(T_2 \vdash \mathsf{Read}\ x : n) \wedge (T_1 \vdash \mathsf{Write}\ x : n \iff T_2 \vdash \mathsf{Write}\ x : n))\big) \wedge$$
$$\big((T_1 \vdash \mathsf{Read}\ x : n \wedge (T_1 \vdash \mathsf{Write}\ x : m \implies m = n)) \implies (T_2 \vdash \mathsf{Write}\ x : l \implies l = n)\big).$$

For $\Phi \in \{\mathbf{SI}, \mathbf{PSI}\}$ we have $T_1 \sqsubseteq_\Phi T_2$ if and only if $T_1 \sqsubseteq_{\mathbf{CC}} T_2$ and for all x, n:

$$\neg(T_1 \vdash \mathsf{Write}\ x : n) \implies \neg(T_2 \vdash \mathsf{Write}\ x : n).$$

We prove the theorem in [14, §A]. In the case of $\Phi = \mathbf{RA}$, we prohibit T_2 from reading more objects than T_1 or changing the values read by T_1; however, it is safe for T_2 to read less than T_1. We also require T_1 and T_2 to have the same sets of final writes. The case of $\Phi \in \{\mathbf{CC}, \mathbf{PC}\}$ introduces two exception to the latter requirement. One exception is when T_1 reads an object and writes the same value to it. Then T_2 may not change the value written, but may omit the write. Another exception is when T_1 reads an object, but does not write to it. Then T_2 can write the value read without invalidating the reads in the context. This is disallowed when $\Phi \in \{\mathbf{SI}, \mathbf{PSI}\}$.

4 Operational Specifications

To justify that our axiomatic specifications of weak consistency models indeed faithfully describe the intended database behaviour, we now prove that they are equivalent to alternative *operational* ones. These are given as algorithms that are close to actual implementations [6, 19, 13, 24], yet abstract from some of the more low-level features that such implementations have. We start by giving an operational specification of the weakest consistency model we consider, Read Atomic. We then specify other models weaker than serialisability by assuming additional guarantees about the communication between replicas in this algorithm.

4.1 Operational Specification of Read Atomic

Informally, the idealised algorithm for Read Atomic operates as follows. The database consists of a set of *replicas*, identified by $\mathsf{RId} = \{r_0, r_1, \ldots\}$, each maintaining a copy of all objects. The set RId is infinite, to model dynamic replica creation. We assume that the system is fully connected: each replica can broadcast messages to all others. All client operations within a given transaction are initially executed at a single replica (though operations in *different* transactions can be executed at different replicas). For simplicity, we assume that every transaction eventually terminates. When this happens, the replica decides whether to commit or abort it. In the former case, the replica sends a message to all other replicas containing the *transaction log*, which describes the updates done by the transaction. The replicas incorporate the updates into their state upon receiving the message. A transaction log has the form $t : \rho$, where $\rho \in \{\mathtt{write}(x, n) \mid x \in \mathsf{Obj}, n \in \mathbb{N}\}^* \triangleq \mathsf{UpdateList}$. This gives

the sequence of values written to objects and the unique *timestamp* $t \in \mathbb{N}$ of the transaction, which is used to determine the precedence of different object versions (and thus implements the AR relation in abstract executions). We denote the set of all sets of logs with distinct timestamps by LogSet.

Every replica processes transactions locally without interleaving. This idealisation does not limit generality, since all anomalies that would result from concurrent execution of transactions at a single replica arise anyway because of the asynchronous propagation of updates between replicas. The above assumption allows us to maintain the state of a replica r in the algorithm by a pair $(D, l) \in \mathsf{RState} \triangleq \mathsf{LogSet} \times (\mathsf{UpdateList} \uplus \{\mathsf{idle}\})$, where:

- l is either the sequence of updates done so far by the (single) transaction currently executing at r, or idle, signifying that no transaction is currently executing; and
- D is the database copy of r, represented by the set of logs of transactions that have committed at r or have been received from other replicas.

Then a *configuration* of the whole system $(R, M) \in \mathsf{Config} \triangleq (\mathsf{RId} \to \mathsf{RState}) \times \mathsf{LogSet}$ is described by the state $R(r)$ of every replica r and the pool of messages M in transit among replicas.

Formally, our algorithm is defined using the transition relation $\twoheadrightarrow$: $\mathsf{Config} \times \mathsf{LEvent} \times \mathsf{Config}$ in Figure 4, which describes how system configurations change in response to *low-level events* from a set LEvent, describing actions by clients and message receipts by replicas. The set LEvent consists of triples of the form $(\iota, r, \mathbf{o})$, where $\iota \in \mathsf{EventId}$ is the event identifier, $r \in \mathsf{RId}$ is the replica the event occurs at, and $\mathbf{o}$ is a *low-level operation* from the set

$$\mathsf{COp} = \{\mathtt{start}, \mathtt{read}(x, n), \mathtt{write}(x, n), \mathtt{commit}(t), \mathtt{abort}, \mathtt{receive}(t : \rho) \mid x \in \mathsf{Obj}, n \in \mathbb{Z}, t \in \mathbb{N}, \rho \in \mathsf{UpdateList}\}.$$

We use $\mathbf{e}, \mathbf{f}, \mathbf{g}$ to range over low-level events.

According to $\twoheadrightarrow$, when a client starts a transaction at a replica r (Start), the database initialises the current sequence of updates to signify that a transaction is in progress. Since a replica processes transactions serially, a transaction can start only if r is not already executing a transaction. When a client writes n to an object x at a replica r (Write), the corresponding record $\mathtt{write}(x, n)$ is appended to the current sequence of updates. This rule can be applied only when r is not executing a transaction. A read of an object x at r (Read) returns the value determined by a lastval function based on the transactions in r's database copy and the current transaction. For $D' \in \mathsf{LogSet}$ we define $\mathsf{lastval}(x, D')$ as the last value written to x by the transaction with the highest timestamp among those in D', or 0 if x is not mentioned in D'. Since the timestamps of transactions in D' are distinct, this defines $\mathsf{lastval}(x, D')$ uniquely. For brevity, we omit its formal definition. Note that (Read) implies that a transaction always reads from its own writes and a snapshot of the database the replica had at its start; the transaction is not affected by writes concurrently executing at other replicas, thus ensuring the absence of unrepeatable reads (§3).

If a transaction aborts at a replica r (Abort), the current sequence of updates of r is cleared. If the transaction commits (Commit), it gets assigned a timestamp t, and its log is added to the message pool, as well as to r's database copy. The timestamp t is chosen to be greater than the timestamps of all the transactions in r's database copy, which validates the condition $\mathsf{AR} \supseteq \mathsf{VIS}$ in Definition 2. The timestamp t also has to be distinct from any timestamp assigned previously in the execution. The fact that (Commit) sends all updates by a transaction in a single message ensures atomic visibility. Note that, in Read Atomic, a transaction can always commit; as we explain in the following, this is not the case for some

$$\text{(Start)}\quad \frac{\mathbf{e} = (\_, r, \texttt{start})}{(R[r \mapsto (D, \mathsf{idle})], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D, \varepsilon)], M)}$$

$$\text{(Write)}\quad \frac{\mathbf{e} = (\_, r, \texttt{write}(x, n))}{(R[r \mapsto (D, \rho)], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D, \rho \cdot \texttt{write}(x, n))], M)}$$

$$\text{(Read)}\quad \frac{\mathbf{e} = (\_, r, \texttt{read}(x, n)) \qquad n = \mathsf{lastval}(x, D \cup \{\infty : \rho\})}{(R[r \mapsto (D, \rho)], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D, \rho)], M)}$$

$$\text{(Abort)}\quad \frac{\mathbf{e} = (\_, r, \texttt{abort})}{(R[r \mapsto (D, \rho)], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D, \mathsf{idle})], M)}$$

$$\text{(Commit)}\quad \frac{\begin{array}{l}\mathbf{e} = (\_, r, \texttt{commit}(t)) \\ (\forall r', D'.\, R(r') = (D', \_) \implies (t : \_) \notin D') \qquad (\forall t'.\, (t' : \_) \in D \implies t > t')\end{array}}{(R[r \mapsto (D, \rho)], M) \xrightarrow{\mathbf{e}} (R[r \mapsto (D \cup \{t : \rho\}, \mathsf{idle})], M \cup \{t : \rho\})}$$

$$\text{(Receive)}\quad \frac{\mathbf{e} = (\_, r, \texttt{receive}(t : \rho))}{(R[r \mapsto (D, \mathsf{idle})], M \cup \{(t : \rho)\}) \xrightarrow{\mathbf{e}} (R[r \mapsto (D \cup \{(t : \rho)\}, \mathsf{idle})], M \cup \{t : \rho\})}$$

Figure 4 Transition relation $\rightarrow$: $\mathsf{Config} \times \mathsf{LEvent} \times \mathsf{Config}$ for defining the operational specification. We let $R[r \mapsto u]$ be the function that has the same value as R everywhere except r, where it has the value u; $\cdot$ denotes sequence concatenation, and ε the empty sequence.

$$\big(\mathbf{e}_1 \in \{(\_, r, \texttt{receive}(t_1 : \_)), (\_, r, \texttt{commit}(t_1))\} \wedge \mathbf{e}_2 = (\_, r, \texttt{commit}(t_2)) \wedge \mathbf{e}_1 \prec \mathbf{e}_2 \wedge r \neq r' \wedge \mathbf{f}_2 = (\_, r', \texttt{receive}(t_2 : \_))\big) \implies \big(\exists \mathbf{f}_1 \in \{(\_, r', \texttt{receive}(t_1 : \_)), (\_, r', \texttt{commit}(t_1))\}.\, \mathbf{f}_1 \prec \mathbf{f}_2\big) \quad \text{(CausalDeliv)}$$

$$\big(\mathbf{e}_1 = (\_, \_, \texttt{commit}(t_1)) \wedge \mathbf{e}_2 = (\_, \_, \texttt{commit}(t_2)) \wedge \mathbf{e}_1 \prec \mathbf{e}_2\big) \implies t_1 < t_2 \quad \text{(MonTS)}$$

$$\big(\mathbf{g} = (\_, r, \texttt{start}) \wedge \mathbf{e}_2 \in \{(\_, r, \texttt{commit}(t_2)), (\_, r, \texttt{receive}(t_2 : \_))\} \wedge \mathbf{f} = (\_, \_, \texttt{commit}(t_1)) \wedge t_1 < t_2 \wedge \mathbf{e}_2 \prec \mathbf{g}\big) \implies \big(\exists \mathbf{e}_1 \in \{(\_, r, \texttt{commit}(t_1)), (\_, r, \texttt{receive}(t_1 : \_))\}.\, \mathbf{e}_1 \prec \mathbf{g}\big) \quad \text{(TotalDeliv)}$$

$$\big(\mathbf{e}_1 = (\_, r, \texttt{write}(x, \_)) \wedge \mathbf{f}_1 = (\_, r, \texttt{commit}(t_1)) \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}_1) = t_1 \wedge \mathbf{e}_2 = (\_, r', \texttt{write}(x, \_)) \wedge \mathbf{f}_2 = (\_, r', \texttt{commit}(t_2)) \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}_2) = t_2 \wedge \mathbf{f}_2 \prec \mathbf{f}_1 \wedge r \neq r'\big) \implies \big(\exists \mathbf{g} \in \mathbf{E}.\, \mathbf{g} = (\_, r, \texttt{receive}(t_2 : \_)) \wedge \mathbf{g} \prec \mathbf{f}_1\big), \quad \text{(ConflictCheck)}$$

where for $\mathbf{e} \in \mathbf{E}$ we let

$$\mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = \begin{cases} t, & \text{if } \exists r.\, \mathbf{e} \in \{(\_, r, \texttt{read}(\_, \_)), (\_, r, \texttt{write}(\_, \_))\} \wedge \\ & \quad \exists \mathbf{g} \in \mathbf{E}.\, \mathbf{g} = (\_, r, \texttt{commit}(t)) \wedge \\ & \quad \neg(\exists \mathbf{f} \in \{(\_, r, \texttt{commit}(\_)), (\_, r, \texttt{abort})\}.\, (\mathbf{e} \prec \mathbf{f} \prec \mathbf{g})) \\ \text{undefined}, & \text{otherwise} \end{cases}$$

Φ	Constraints	Φ	Constraints	Φ	Constraints
RA	None	**PSI**	(CausalDeliv), (ConflictCheck)	**SI**	(MonTS), (TotalDeliv), (ConflictCheck)
CC	(CausalDeliv)	**PC**	(MonTS), (TotalDeliv)		

Figure 5 Constraints on concrete executions $\mathcal{C} = (\mathbf{E}, \prec)$ required by various consistency models. Free variables are universally quantified and range over the following domains: $\mathbf{e}_i, \mathbf{f}, \mathbf{f}_i, \mathbf{g} \in \mathbf{E}$ for $i = 1, 2$; $t_1, t_2 \in \mathbb{N}$; $r, r' \in \mathsf{RId}$.

of the other consistency models. Finally, a replica r that is not executing a transaction can receive a transaction log from the message pool (Receive), adding it to the database copy.

We define the semantics of Read Atomic by considering all sequences of transitions generated by $\rightarrowtriangle$ from an initial configuration where the log sets of all replicas and the message pool are empty. We thereby consider all possible operations that clients could issue to the database.

▶ **Definition 5.** Let $(R_0, M_0) = (\lambda r.\, (\emptyset, \mathsf{idle}), \emptyset)$. A *concrete execution* is a pair $\mathcal{C} = (\mathbf{E}, \prec)$, where: $\mathbf{E} \subseteq \mathsf{LEvent}$; $\prec$ is a prefix-finite, total order on $\mathbf{E}$; and if $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \ldots$ is the enumeration of the events in $\mathbf{E}$ defined by $\prec$, then for some configurations $(R_1, M_1), (R_2, M_2), \ldots \in \mathsf{Config}$ we have $(R_0, M_0) \xrightarrow{\mathbf{e}_1} (R_1, M_1) \xrightarrow{\mathbf{e}_2} (R_2, M_2) \xrightarrow{\mathbf{e}_3} \ldots$

4.2 Correspondence to Axiomatic Specifications and Other Models

We next show that the above operational specification indeed defines the semantics of Read Atomic, and that stronger models can be defined by assuming additional guarantees about communication between replicas. These guarantees are formalised by the constraints on concrete executions in Figure 5; in implementations they would be ensured by distributed protocols that our specifications abstract from.

We first map each concrete execution into a history, which includes only reads and writes in its committed transactions. The history of $\mathcal{C} = (\mathbf{E}, \prec)$ is defined as follows:

$$\begin{aligned}
&\mathsf{history}(\mathcal{C}) = \big\{T_t \mid \{\mathbf{e} \in \mathbf{E} \mid \mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t\} \neq \emptyset\big\}, \text{ where } T_t = (E_t, \mathsf{po}_t) \text{ for}\\
&E_t = \{(\iota, \mathbf{o}) \mid \exists \mathbf{e} \in \mathbf{E}.\, \mathbf{e} = (\iota, _, \mathbf{o}) \wedge \mathsf{TS}_{\mathcal{C}}(\mathbf{e}) = t\};\\
&\mathsf{po}_t = \{(\iota_1, \mathbf{o}_1), (\iota_2, \mathbf{o}_2) \mid (\iota_1, \mathbf{o}_1), (\iota_2, \mathbf{o}_2) \in E_t \wedge (\iota_1, _, \mathbf{o}_1) \prec (\iota_2, _, \mathbf{o}_2)\},
\end{aligned}$$

where $\mathsf{TS}_{\mathcal{C}}$ is defined in Figure 5. We lift the function $\mathsf{history}$ to sets of concrete executions as expected.

▶ **Theorem 6.** *For a consistency model Φ let $\mathsf{ConcExec}_\Phi$ be the set of concrete executions satisfying the model-specific constraints in Figure 5. Then $\mathsf{history}(\mathsf{ConcExec}_\Phi) = \mathsf{Hist}_\Phi$.*

Proof outline. We defer the full proof to [14, §B]. Here we sketch the argument for one set inclusion ($\subseteq$) and, on the way, explain the constraints in Figure 5. Fix a Φ and let $\mathcal{C} = (\mathbf{E}, \prec) \in \mathsf{ConcExec}_\Phi$. To show $\mathsf{history}(\mathcal{C}) \in \mathsf{Hist}_\Phi$ we let $\mathcal{A} = (\mathsf{history}(\mathcal{C}), \mathsf{VIS}, \mathsf{AR})$, where $\mathsf{AR} = \{(T_{t_1}, T_{t_2}) \mid t_1 < t_2\}$ and

$$\begin{aligned}
\mathsf{VIS} = \big\{(T_{t_1}, T_{t_2}) \mid \exists \mathbf{e}_1, \mathbf{e}_2 \in \mathbf{E}.\, \exists r.\, \mathbf{e}_1 \in \{(_, r, \mathtt{commit}(t_1)), (_, r, \mathtt{receive}(t_1 : _))\} \wedge\\
\mathbf{e}_2 = (_, r, \mathtt{commit}(t_2)) \wedge \mathbf{e}_1 \prec \mathbf{e}_2\big\}.
\end{aligned}$$

While AR merely lifts the order on timestamps to transactions, VIS reflects message delivery: $T_{t_1} \xrightarrow{\mathsf{VIS}} T_{t_2}$ if the effects of T_{t_1} have been incorporated into the state of the replica where T_{t_2} is executed. We can show that any abstraction execution $\mathcal{A}$ constructed from a concrete execution $\mathcal{C}$ as above satisfies Int and Ext, and hence, its history belongs to $\mathsf{Hist}_{\mathbf{RA}}$. The constraints on a concrete execution $\mathcal{C}$ in Figure 5 ensure that the abstract execution $\mathcal{A}$ constructed from it satisfies other axioms in Figure 2.

Constraint (CausalDeliv) implies the axiom TransVis, because it ensures that the message delivery is *causal* [9]: if a replica r sends the log of a transaction t_2 (event $\mathbf{e}_2$) after it sends or receives the log of t_1 (event $\mathbf{e}_1$), then every other replica r' will receive the log of t_2 (event $\mathbf{f}_2$) only after it receives or sends the log of t_1 (event $\mathbf{f}_1$).

The axiom PREFIX follows from constraints (MonTS) and (TotalDeliv). The constraint (MonTS) requires that timestamps agree with the order in which transactions commit. The constraint (TotalDeliv) requires that each transaction access a database snapshot that is closed under adding transactions with timestamps (t_1) smaller than the ones already present in the snapshot (t_2). In an implementation, the above constraints can be satisfied if replicas communicate via a central server, which assigns timestamps to transactions when they commit, and propagates their logs to replicas in the order of their timestamps [13].

The axiom NOCONFLICT follows from the constraint (ConflictCheck), similar to that in the original definitions of **SI** [8] and **PSI** [24]. The constraint allows a transaction t_1 to commit at a replica r (event $\mathbf{f}_1$) only if it passes a *conflict detection check*: if t_1 updates an object x (event $\mathbf{e}_1$) that is also updated by a transaction t_2 (event $\mathbf{e}_2$) committed at another replica r' (event $\mathbf{f}_2$), then the replica r must have received the log of t_2 (event $\mathbf{g}$). If this check fails, the only option left for the database is to abort t using the rule (Abort). Implementing the check in a realistic system would require the replica r to coordinate with others on commit [24]. ◀

The above operational specifications are closer to the intuition of practitioners [6, 19, 13, 24] and thus serve to validate our axiomatic specifications. However, they are more verbose and reasoning about database behaviour using them may get unwieldy. It requires us to keep track of low-level information about the system state, such as the logs at all replicas and the set of messages in transit. We then need to reason about how the system state is affected by a large number of possible interleavings of operations at different replicas. In contrast, our axiomatic specifications (§3) are more declarative and, in particular, do not refer to implementation-level details, such as message exchanges between replicas. These specifications thereby facilitate reasoning about the database behaviour.

5 Related Work

Our specification framework builds on the axiomatic approach to specifying consistency models, previously applied to weak shared-memory models [3] and eventual consistency [11, 12]. In particular, the visibility and arbitration relations were first introduced for specifying eventual consistency and causally consistent transactions [12]. In comparison to prior work, we handle more sophisticated transactional consistency models. Furthermore, our framework is specifically tailored to transactional models with atomic visibility, by defining visibility and arbitration relations on whole transactions as opposed to events. This avoids the need to enforce atomic visibility explicitly in all axioms [12], thus simplifying specifications.

Adya [2] has previously proposed specifications for weak consistency models of transactions in classical databases. His framework also broadly follows the axiomatic specification approach, but uses relations different from visibility and arbitration. Adya's work did not address the variety of consistency models for large-scale databases proposed recently, while our framework is particularly appropriate for these. On the other hand, Adya handled transactional consistency models that do not guarantee atomic visibility, such as Read Committed, which we do not address. Adya also specified snapshot isolation (**SI**), which is a weak consistency model older than the others we consider. However, his specification is low-level, since it introduces additional events to denote the times at which a transaction takes a snapshot of the database state. Saeida Ardekani et al. [22] have since proposed a higher-level specification for snapshot isolation; this specification still uses relations on individual events and thus does not exploit atomic visibility.

Partial orders have been used to define semantics of concurrent and distributed programs, e.g., by event structures [26]. Our results extend this research line by considering new kinds of relations among events, appropriate to describe computations of weakly consistent databases, and by relating the resulting abstract specifications to lower-level algorithms.

Prior work has investigated calculi with transactions communicating via message passing: cJoin [10], TCCS^m [17] and RCCS [15]. Even though replicated database implementations and our operational specifications are also based on message passing, the database interface that we consider allows client programs only to read and write objects. Thereby, it provides the programs with an (imperfect) illusion of *shared memory*, and our goal was to provide specifications for this interface that abstract from its message passing-based implementation.

6 Conclusion

We have proposed a framework for uniformly specifying transactional consistency models of modern replicated databases. The axiomatic nature of our framework makes specifications declarative and concise, with further simplicity brought by exploiting atomic visibility. We have illustrated the use of the framework by specifying several existing consistency models and thereby systematising the knowledge about them. We have also validated our axiomatic specifications by proving their equivalence to operational specifications that are closer to implementations.

We hope that our work will promote an exchange of ideas between the research communities of large-scale databases and concurrency theory. In particular, our framework provides a basis to develop techniques for reasoning about the correctness of application programs using modern databases; this is the subject of our ongoing work.

Finally, axiomatic specifications are well-suited for systematically exploring the design space of consistency models. In particular, insights provided by the specifications may suggest new models, as we illustrated by the Update Atomic model in §3. This is likely to help in the design of the sophisticated programming interfaces that replicated databases are starting to provide to compensate for the weakness of their consistency models. For example, so-called replicated data types [23] avoid lost updates by eventually merging concurrent updates without coordination between replicas, and sessions [25] provide additional consistency guarantees for transactions issued by the same client. Finally, there are also interfaces that allow the programmer to request different consistency models for different transactions [18], analogous to fences in weak memory models [3]. In the future we plan to generalise our techniques to handle the above features. We expect to handle replicated data types by integrating our framework with their specifications proposed in [11], and to handle sessions and mixed consistency models by studying additional constraints on the visibility and arbitration relations. We believe that the complexity of database consistency models and the above programming interfaces makes it indispensable to specify them formally and declaratively. Our work provides the necessary foundation for achieving this.

Acknowledgements. We thank Artem Khyzha, Vasileios Koutavas, Hongseok Yang and the anonymous reviewers for comments that helped improve the paper. This work was supported by the EU FET project ADVENT.

References

1 Daniel Abadi. Consistency tradeoffs in modern distributed database system design: CAP is only part of the story. *IEEE Computer*, 45(2), 2012.

2 Atul Adya. Weak consistency: A generalized theory and optimistic implementations for distributed transactions. PhD thesis, MIT, 1999.
3 Jade Alglave. A formal hierarchy of weak memory models. *FMSD*, 41(2), 2012.
4 Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Highly Available Transactions: virtues and limitations. In *VLDB*, 2014.
5 Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. The potential dangers of causal consistency and an explicit solution. In *SOCC*, 2012.
6 Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Scalable atomic visibility with RAMP transactions. In *SIGMOD*, 2014.
7 Mark Batty, Mike Dodds, and Alexey Gotsman. Library abstraction for C/C++ concurrency. In *POPL*, 2013.
8 Hal Berenson, Phil Bernstein, Jim Gray, Jim Melton, Elizabeth O'Neil, and Patrick O'Neil. A critique of ANSI SQL isolation levels. In *SIGMOD*, 1995.
9 Kenneth P. Birman and Thomas A. Joseph. Reliable communication in the presence of failures. *ACM Trans. Comput. Syst.*, 5(1), 1987.
10 Roberto Bruni, Hernán C. Melgratti, and Ugo Montanari. cJoin: Join with communicating transactions. *Mathematical Structures in Computer Science*, 25(3), 2015.
11 Sebastian Burckhardt, Alexey Gotsman, Hongseok Yang, and Marek Zawirski. Replicated data types: specification, verification, optimality. In *POPL*, 2014.
12 Sebastian Burckhardt, Daan Leijen, Manuel Fähndrich, and Mooly Sagiv. Eventually consistent transactions. In *ESOP*, 2012.
13 Sebastian Burckhardt, Daan Leijen, Jonathan Protzenko, and Manuel Fähndrich. Global sequence protocol: A robust abstraction for replicated shared state. In *ECOOP*, 2015.
14 Andrea Cerone, Giovanni Bernardi, and Alexey Gotsman. A framework for transactional consistency models with atomic visibility (extended version). Available from `http://software.imdea.org/~gotsman/`.
15 Vincent Danos and Jean Krivine. Transactions in RCCS. In *CONCUR*, 2005.
16 Rocco De Nicola and Matthew Hennessy. Testing equivalence for processes. In *ICALP*, 1983.
17 Vasileios Koutavas, Carlo Spaccasassi, and Matthew Hennessy. Bisimulations for communicating transactions (extended abstract). In *FOSSACS*, 2014.
18 Cheng Li, Daniel Porto, Allen Clement, Rodrigo Rodrigues, Nuno Preguiça, and Johannes Gehrke. Making geo-replicated systems fast if possible, consistent when necessary. In *OSDI*, 2012.
19 Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In *SOSP*, 2011.
20 Christos H. Papadimitriou. The serializability of concurrent database updates. *J. ACM*, 26(4), 1979.
21 M. Saeida Ardekani, P. Sutra, and M. Shapiro. Non-monotonic snapshot isolation: Scalable and strong consistency for geo-replicated transactional systems. In *SRDS*, 2013.
22 M. Saeida Ardekani, P. Sutra, M. Shapiro, and N. Preguiça. On the scalability of snapshot isolation. In *Euro-Par*, 2013.
23 Marc Shapiro, Nuno M. Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *SSS*, 2011.
24 Y. Sovran, R. Power, M. K. Aguilera, and J. Li. Transactional storage for geo-replicated systems. In *SOSP*, 2011.
25 Douglas B. Terry, Alan J. Demers, Karin Petersen, Mike Spreitzer, Marvin Theimer, and Brent W. Welch. Session guarantees for weakly consistent replicated data. In *PDIS*, 1994.
26 Glynn Winskel. Event structure semantics for CCS and related languages. In *ICALP*, 1982.

Safety of Parametrized Asynchronous Shared-Memory Systems is Almost Always Decidable*

Salvatore La Torre[1], Anca Muscholl[2], and Igor Walukiewicz[2]

1 **Dipartimento di Informatica, Università degli Studi di Salerno, Italy**
slatorre@unisa.it
2 **LaBRI, Bordeaux University, CNRS, France**
{anca,igw}@labri.fr

Abstract

Verification of concurrent systems is a difficult problem in general, and this is the case even more in a parametrized setting where unboundedly many concurrent components are considered. Recently, Hague proposed an architecture with a leader process and unboundedly many copies of a contributor process interacting over a shared memory for which safety properties can be effectively verified. All processes in Hague's setting are pushdown automata. Here, we extend it by considering other formal models and, as a main contribution, find very liberal conditions on the individual processes under which the safety problem is decidable: the only substantial condition we require is the effective computability of the downward closure for the class of the leader processes. Furthermore, our result allows for a hierarchical approach to constructing models of concurrent systems with decidable safety problem: networks with tree-like architecture, where each process shares a register with its children processes (and another register with its parent). Nodes in such networks can be for instance pushdown automata, Petri nets, or multi-pushdown systems with decidable reachability problem.

1998 ACM Subject Classification D.2.4 Software/Program Verification, C.2.4 Distributed Systems

Keywords and phrases Verification, parametrized systems, shared memory

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.72

1 Introduction

Parametrized concurrent systems, i.e., systems composed of an arbitrary number of concurrent components of finitely many kinds, are the natural models of many concrete systems such as distributed network protocols, operating system drivers, or multi-threaded applications for multi-core hardware, just to mention a few. In some cases, they are algorithmically easier to analyze than the corresponding non-parametrized models which makes them a suitable abstraction for concurrency (see [21]).

Verification of shared-memory systems is a notoriously difficult problem, even for simple properties as safety. As a classical example, two pushdown systems communicating via a shared variable can directly simulate a Turing machine. In order to gain decidability, it is

* The results were obtained during a visit of the first author at LaBRI, for which the financial support of Bordeaux INP is acknowledged. This work was also partially supported by the FARB grants 2012-2014, Università degli Studi di Salerno.

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 72–84

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

$Class_{\mathcal{D}}$: possible instances for the leader	$Class_{\mathcal{C}}$: possible instances for contributors
▪ pushdown automata, ▪ Petri nets, ▪ decidable subclasses of multi-stack pushdown automata, ▪ stacked counter automata, ▪ order-2 pushdown automata.	▪ anything in $Class_{\mathcal{D}}$, ▪ higher-order pushdown automata with collapse, ▪ lossy channel systems, ▪ hierarchical composition of $(\mathcal{C},\mathcal{D})$-systems with $Class_{\mathcal{D}}$ and $Class_{\mathcal{C}}$ from this table.

Figure 1 Examples of models that fit our general decidability result for $(\mathcal{C},\mathcal{D})$-systems.

crucial to limit the synchronization power of such systems, for example by placing restrictions on the policies of the synchronization primitives, or bounding the number of interactions. For parametrized systems, assuming that the components have no identities is helpful besides being appropriate for many concurrent systems of interest (see also [14] for a survey).

In this paper, we revisit the verification problem for safety properties of *parametrized asynchronous shared-memory systems*. These systems consist of a *leader* process $\mathcal{D}$ and an arbitrary number of identical *contributor* processes $\mathcal{C}$. The processes communicate via shared memory modelled by read/write registers. There are two important features of such $(\mathcal{C},\mathcal{D})$-systems: first, there are no locking mechanisms on the shared memory, and second, contributors do not have identities.

This setting has been proposed by Hague [19], who studied the case when leaders and contributors are pushdown automata and showed an EXPSPACE upper bound. Esparza et al. [16] settled the complexity of the problem for pushdown automata proving it PSPACE-complete. The interest for such systems is also related to the analysis of distributed protocols that use no synchronization primitives, which is the case on wireless sensor networks where a central co-ordinator (the base station) communicates with an arbitrary number of tiny agents that run concurrently and asynchronously (see [16]).

In this paper, we prove a general decidability result for verifying safety properties of $(\mathcal{C},\mathcal{D})$-systems. It gives conditions on leaders and contributors, expressed in terms of basic language theoretic closure and effectiveness properties, under which the problem is decidable. The main requirement is that the downward closure of the language of the leader should be effectively computable. This requirement is interesting in itself, and we remark that, in our setting, it is weaker than having effective semilinear Parikh images.

Our work shows that the setting of $(\mathcal{C},\mathcal{D})$-systems can be instantiated in many different ways, while preserving the decidability of safety properties. Figure 1 lists some examples of types of systems for leaders and for contributors that our theorem covers. For example, the leader and contributors can be themselves Petri nets, or restrictions of multi-pushdown processes with decidable reachability problem.

One interesting consequence of our main result is that it can be applied recursively, by instantiating a contributor $\mathcal{C}$ by another $(\mathcal{C},\mathcal{D})$-system (see Figure 2). This implies that safety properties can be verified for networks that have a tree-like architecture: each process shares a register with its children processes (and another register with its parent). Nodes in such networks can belong to one of the formal models listed above.

Finally, as a byproduct, our construction allows to reprove in a different way known complexity results for $(\mathcal{C},\mathcal{D})$-systems over pushdown automata [16]. We believe that our approach is simpler thanks to the use of downward closures.

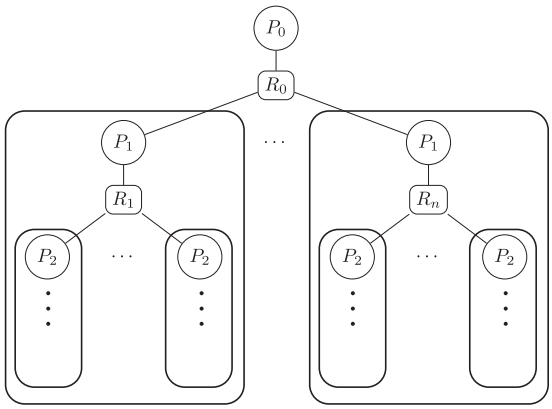


Figure 2 Example of a hierarchical composition of $(\mathcal{C}, \mathcal{D})$-systems, P_i are process types and R_i are R/W registers. P_0 is the leader $\mathcal{D}$ of a $(\mathcal{C}, \mathcal{D})$-system whose contributors are themselves $(\mathcal{C}, \mathcal{D})$-systems each one with leader from P_1 and contributors from P_2.

Related work. Parametrized verification of shared-memory multi-threaded programs has been studied for finite-state threads in [8, 22] and for threads modeled as pushdown systems in [7, 10, 24, 25, 9]. The main difference with our setting is that in those models the synchronization primitives are allowed, and thus for pushdown threads, reachability becomes undecidable even if we restrict to finite data domains. The decidability results in [7, 10, 24, 25, 9] concern the reachability analysis up to a bounded number of execution contexts, and in [7, 10], dynamic thread creation is allowed.

Parameterized reachability is also considered in [6] for shared-memory systems formed of one pushdown and several counters.

There is a rich literature concerning the verification of asynchronously-communicating parametrized programs that is mostly related to the verification of distributed protocols and varies for approaches and models (see e.g. [17, 11, 15, 23] for some early work, and [13, 5, 29] and references therein).

Parametrized tree systems, i.e., systems formed of an arbitrary number of processes operating on a tree-like architecture, have been studied by tree rewritings (see [4, 3] and references therein). Our hierarchical composition of $(\mathcal{C}, \mathcal{D})$-systems is quite different from the models studied there. Namely, each process shares a finite memory with its children processes and its parent process: all the interactions with the network neighbours are through asynchronous accesses to such memories. As processes, we allow several classes of systems, not just finite-state systems. On the other side, in our model there is no notion of global transitions.

Organization of the paper. In Section 2, we give some basic definitions and introduce the notion of $(\mathcal{C}, \mathcal{D})$-system. In Section 3, the accumulator semantics is introduced and shown equivalent to the standard semantics of $(\mathcal{C}, \mathcal{D})$-systems. In Section 4, we give two constructions that allow to decompose the semantics of $(\mathcal{C}, \mathcal{D})$-systems into the parts concerning respectively the leader and the contributors. In Section 5, these constructions are used to give a decision algorithm that shows the main result of the paper. In Section 6, we use our approach to study the computational complexity of the reachability for $(\mathcal{C}, \mathcal{D})$-systems for the classes of finite automata and pushdown automata. We conclude in Section 7 with a few remarks.

2 Preliminaries

We first define the parametrized systems that we consider, and their reachability problem. These systems consist of one instance of a leader process $\mathcal{D}$, and an arbitrary number of instances of a contributor process $\mathcal{C}$. Both $\mathcal{C}$ and $\mathcal{D}$ can be arbitrary, potentially infinite, transition systems. One can think of them as transition systems generated by, for example, pushdown automata, Petri nets, or lossy channel systems. Our decidability result will refer to the closure properties of classes of transition systems over which $\mathcal{C}$ and $\mathcal{D}$ range.

A *transition system* is a graph with states and labelled edges. The labels of edges are called *actions*. There may be infinitely many states in a transition system, but we will assume that the set of actions is finite. A transition system will come with an initial state. A *trace* is a sequence of actions labelling a path starting in the initial state. A word v is a *subword* of u if it can be obtained from u by erasing letters.

The *synchronized product* of two transition systems is a system whose state set is the product of the state sets of the two systems, and whose transitions are defined according to the rule: for actions common to the two systems the transition should be synchronized, whereas actions of only one of the two systems affect only the relevant component of the pair. In particular, the synchronized product of two transition systems over the same alphabet is just the standard product of the two.

For our decidability results we will assume implicitly that transition systems are given by some finite description. For example, when we will talk about the class of pushdown transition systems we will assume that they are given by pushdown automata.

We say that a class of transition systems is *effectively closed* under some operation if from a description of a transition system in the class we can effectively construct a description of the image of the transition system under that operation. Our decidability result will use a couple of abstract properties of classes of transition systems. For a class $\mathcal{C}$ of transitions systems we say that:

- $\mathcal{C}$ is *effectively closed under synchronized products with finite automata* if for every description of a system in $\mathcal{C}$, and every finite automaton, one can effectively find a description in $\mathcal{C}$ of the synchronized product of the transition system with the automaton.
- $\mathcal{C}$ has *decidable reachability problem* if there is an algorithm deciding for a given action and a description of a transition system in $\mathcal{C}$ if from the initial state of $\mathcal{C}$ there is a trace containing this action.
- $\mathcal{C}$ has an *effective downward closure* if there is an algorithm calculating for a given description of a transition system in $\mathcal{C}$ the finite automaton accepting all subwords of the traces of $\mathcal{C}$ from the initial state.

Observe that having effective downward closure implies having decidable reachability problem. We will give an example of the use of these notions in a simple result on page 77 (Corollary 2).

We proceed to the formal definition of $(\mathcal{C}, \mathcal{D})$-systems. These systems are composed of arbitrary many instances of a contributor process $\mathcal{C}$ and one instance of a leader process $\mathcal{D}$. The processes communicate through a shared register. We write G for the finite set of register values, and use g, h to range over elements of G. The initial value of the register is denoted g_{init}. The alphabets of both $\mathcal{C}$ and $\mathcal{D}$ contain actions representing reads and writes to the register:

$$\Sigma_{\mathcal{C}} = \{r(g), w(g) : g \in G\} \ , \qquad \Sigma_{\mathcal{D}} = \{\bar{r}(g), \bar{w}(g) : g \in G\} \ .$$

Both $\mathcal{C}$ and $\mathcal{D}$ are, possibly infinite, transition systems over these alphabets:

$$\mathcal{C} = \langle S, \delta \subseteq S \times \Sigma_{\mathcal{C}} \times S, s_{init} \rangle \qquad \mathcal{D} = \langle T, \Delta \subseteq T \times \Sigma_{\mathcal{D}} \times T, t_{init} \rangle \ .$$

The transition systems do not have internal actions. Adding them to the alphabets would not modify our results, so for simplicity we prefer not to deal with them here. Internal actions will be useful though when we consider hierarchical composition of $(\mathcal{C}, \mathcal{D})$-systems.

A $(\mathcal{C}, \mathcal{D})$-system consists of an unspecified number of copies of $\mathcal{C}$, one copy of $\mathcal{D}$, and a shared register. A configuration of such a system can be represented by a triple $(f : \mathbb{N} \to S, t \in T, g \in G)$, consisting of a function f counting the number of instances of $\mathcal{C}$ in a given state, the state t of $\mathcal{D}$ and the current register value g.

In principle we would expect that f is a partial function defined only on an initial interval of $\mathbb{N}$. This would represent that indeed there are only finitely many copies of $\mathcal{C}$. Given the questions we are interested in, this is irrelevant, so we prefer not to put this condition.

The transitions of the $(\mathcal{C}, \mathcal{D})$-system are presented below. We extend the transition relation δ of $\mathcal{C}$ from S to $\mathbb{N} \to S$ and write $f \xrightarrow{a} f'$ in δ, meaning that there is an index i such that $f(i) \xrightarrow{a} f'(i)$ in $\mathcal{C}$ and $f(j) = f'(j)$ for $j \neq i$.

$$
\begin{array}{ll}
(f,t,g) \xrightarrow{\bar{w}(h)} (f,t',h) & \text{if } t \xrightarrow{\bar{w}(h)} t' \text{ in } \Delta \\
(f,t,g) \xrightarrow{\bar{r}(h)} (f,t',h) & \text{if } t \xrightarrow{\bar{r}(h)} t' \text{ in } \Delta \text{ and } h = g \\
(f,t,g) \xrightarrow{w(h)} (f',t,h) & \text{if } f \xrightarrow{w(h)} f' \text{ in } \delta \\
(f,t,g) \xrightarrow{r(h)} (f',t,h) & \text{if } f \xrightarrow{r(h)} f' \text{ in } \delta \text{ and } h = g
\end{array}
$$

The reachability problem is to decide if in a given $(\mathcal{C}, \mathcal{D})$-system the register can contain some error value that we denote by $\#$. Observe that it may be assumed w.l.o.g. that this value is written by the leader $\mathcal{D}$. This means that we are asking if there is a trace of the $(\mathcal{C}, \mathcal{D})$-system from the initial configuration $(f_{init}, t_{init}, g_{init})$, with label from $(\Sigma_{\mathcal{C}} \cup \Sigma_{\mathcal{D}})^* \bar{w}(\#)$. Here, f_{init} is a constant function assigning the initial state of $\mathcal{C}$ to every $i \in \mathbb{N}$. In the following we will simply say that we want to decide if there is a *#-trace* in the $(\mathcal{C}, \mathcal{D})$-system.

3 Accumulator semantics

The semantics we have presented above, although natural, is not that easy to work with. Here we formulate a different semantics, called accumulator semantics, that is equivalent if the reachability problem is considered. As an example of the advantage offered by the accumulator semantics we give a very simple argument for the decidability in the case when $\mathcal{C}$ ranges over finite state systems.

In the accumulator semantics, instead of a function $f : \mathbb{N} \to S$ we use a set $A \subseteq S$ that we call accumulator to reflect the fact that it can only grow. The idea is that since we reason in parametrized setting, we do not need to count precisely how many copies of $\mathcal{C}$ have reached a given state. Once that a state is reached, it can be "duplicated" an arbitrary number of times. So in the accumulator semantics configurations are of the form $(A \subseteq S, t \in T, g \in G)$, and the transitions are:

$$
\begin{array}{ll}
(A,t,g) \xrightarrow{\bar{w}(h)} (A,t',h) & \text{if } t \xrightarrow{\bar{w}(h)} t' \text{ in } \Delta \\
(A,t,g) \xrightarrow{\bar{r}(h)} (A,t',h) & \text{if } t \xrightarrow{\bar{r}(h)} t' \text{ in } \Delta \text{ and } h = g \\
(A,t,g) \xrightarrow{w(h)} (A \cup \{s'\},t,h) & \text{if } s \xrightarrow{w(h)} s' \text{ in } \delta \text{ for some } s \in A \\
(A,t,g) \xrightarrow{r(h)} (A \cup \{s'\},t,h) & \text{if } h = g \text{ and } s \xrightarrow{r(h)} s' \text{ in } \delta \text{ for some } s \in A
\end{array}
$$

▶ **Proposition 1.** *There is a #-trace from $(f_{init}, t_{init}, g_{init})$ in the $(\mathcal{C}, \mathcal{D})$-system iff there is one from $(\{s_{init}\}, t_{init}, g_{init})$ in the accumulator semantics.*

Proof. For the left to right direction, we take a run $\{(f_i, t_i, g_i)\}_{i=1,\ldots,n}$ and show that $\{(A_i, t_i, g_i)\}_{i=1,\ldots,n}$ is a run, where A_i is the set of the states of $\mathcal{C}$ that have appeared as a value of one of $f_1, \ldots, f_i$.

For the right to left direction we prove a more general statement. Suppose that σ is a $\#$-trace in the accumulator semantics from a state (A, t, g) and f is such that $|\{i : f(i) = s\}| \geq 2^{|\sigma|}$ for all $s \in A$. Then we show that there is a $\#$-trace from (f, t, g) in $(\mathcal{C}, \mathcal{D})$-system. The proposition then follows since in f_{init} we have arbitrarily many copies of s_{init}. The proof of this statement is by induction on the length of σ. One step in the accumulator semantics is simulated by letting either $\mathcal{D}$ take the step, or half of the copies of $\mathcal{C}$ take the step. ◀

Note that the standard and the accumulator semantics do not generate the same traces. In order to simulate a step in the accumulator semantics, the $(\mathcal{C}, \mathcal{D})$-system may need to perform several steps.

If $\mathcal{C}$ is a finite state automaton then the A-part in the accumulator semantics is of bounded size. This gives a simple decidability result:

▶ **Corollary 2.** *Suppose that $Class_{\mathcal{D}}$ is closed under synchronized products with finite automata. The reachability problem for $(\mathcal{C}, \mathcal{D})$-systems where $\mathcal{C}$ is a finite-state automaton, and $\mathcal{D}$ is from $Class_{\mathcal{D}}$, effectively reduces to the reachability problem in $Class_{\mathcal{D}}$.*

Proposition 1 allows us to use the accumulator semantics as the semantics of $(\mathcal{C}, \mathcal{D})$-systems, and this will be our implicit assumption in the following sections.

4 Capacities and downward closures

Our objective is a decidability result for $(\mathcal{C}, \mathcal{D})$-systems. It will be obtained by combining two reductions that we describe in this section. First, we will decompose the semantics of a $(\mathcal{C}, \mathcal{D})$-system into the part concerning $\mathcal{C}$ and the one concerning $\mathcal{D}$. Lemma 4 reduces our problem to that of finding an input on which we can run separately two parts. The second step starts from the observation that instead of $\mathcal{D}$ we can work with the downward closure of $\mathcal{D}$ (Lemma 5). Then we can rely on the well-known fact that the downward closure of *any* language is regular. Using a decomposition technique similar to that of Lemma 4 we obtain our main technical result, Lemma 7. This will be turned into a decision procedure in the next section.

We start by defining the transition system $\mathcal{D}^\kappa$, that captures the part of the $(\mathcal{C}, \mathcal{D})$-system concerning $\mathcal{D}$. The system $\mathcal{D}^\kappa$ is obtained by abstracting the register contributions of $\mathcal{C}$ by a set $K \subseteq G$ of possible values. Let $\mathcal{D}^\kappa = \langle \mathcal{P}(G) \times T \times G, \delta, (\emptyset, t_{init}, g_{init}) \rangle$. So a configuration of $\mathcal{D}^\kappa$ has the form

$$(K \subseteq G, t \in T, g \in G).$$

Intuitively, K represents a *capacity*: the values that contributors have already written into the register up to the present point of the execution of the system. State $t \in T$ is the current state of $\mathcal{D}$, and $g \in G$ is the current content of the register.

To update the K-component of a configuration we introduce a new alphabet

$$\Sigma_\nu = \{\nu(g) : g \in G\},$$

and let the alphabet of $\mathcal{D}^\kappa$ be $\Sigma_{\mathcal{D}} \cup \Sigma_\nu$. The intuition behind the transitions of $\mathcal{D}^\kappa$ presented below is the following. Since an arbitrary number of copies of $\mathcal{C}$ can be started, whenever a value g is written in the register by a contributor we can construct a different run where this

instance of the contributor is duplicated some number of times. In this new run, any time in the future of the computation when the value g is needed in the register we can use one of these duplicates. We capture this phenomenon in the transitions of $\mathcal{D}^\kappa$ by enabling a read action $\bar{r}(h)$ whenever $h \in K$.

Precisely, the transitions of $\mathcal{D}^\kappa$ are:

$$(K,t,g) \xrightarrow{\bar{w}(h)} (K,t',h) \qquad \text{if } t \xrightarrow{\bar{w}(h)} t' \text{ in } \Delta,$$
$$(K,t,g) \xrightarrow{\bar{r}(h)} (K,t',h) \qquad \text{if } t \xrightarrow{\bar{r}(h)} t' \text{ in } \Delta \text{ and } h \in K \cup \{g\},$$
$$(K,t,g) \xrightarrow{\nu(h)} (K \cup \{h\},t,h) \qquad \text{if } h \notin K.$$

It is not difficult to see that if there is a #-trace in the $(\mathcal{C}, \mathcal{D})$-system then there is one in $\mathcal{D}^\kappa$. The opposite is clearly not true because $\mathcal{D}^\kappa$ ignores the form of $\mathcal{C}$: there is no check that $\nu(h)$ actions can indeed come from writes of $\mathcal{C}$. We will recover the equivalence by putting an additional condition on traces of $\mathcal{D}^\kappa$ (cf. Lemma 4 below).

In order to obtain a sufficient condition on traces of $\mathcal{D}^\kappa$ we construct a "capacity aware" version of $\mathcal{C}$. This is the transition system $\mathcal{C}^\kappa = \langle \mathcal{P}(G) \times S \times G, \delta^\kappa, (\emptyset, s_{init}, g_{init}) \rangle$ where δ^κ is:

$$(K,s,g) \xrightarrow{\bar{w}(h)} (K,s,h) \quad (K,s,g) \xrightarrow{\bar{r}(h)} (K,s,h) \quad (K,s,g) \xrightarrow{\nu(h)} (K \cup \{h\},s,h)$$
$$(K,s,g) \xrightarrow{w(h)} (K,s',h) \quad \text{if } s \xrightarrow{w(h)} s' \text{ in } \delta \text{ and } h \in K$$
$$(K,s,g) \xrightarrow{r(h)} (K,s',h) \quad \text{if } s \xrightarrow{r(h)} s' \text{ in } \delta \text{ and } h \in K \cup \{g\}.$$

This automaton follows the actions of $\mathcal{D}^\kappa$ (first line above) in order to be aware of the current contents of the register and the capacity. At the same time, $\mathcal{C}^\kappa$ can also do the $w(h)$ actions provided they are declared in the capacity K, and the $r(h)$ actions when h is either in the capacity K or in the register. So the capacity restricts the write actions of contributors and allows for more read actions.

We stress the following:

1. Both for $\mathcal{C}^\kappa$ and $\mathcal{D}^\kappa$, the content of the register after a transition is determined by the executed action.
2. Both systems have two kinds of reads: from the register g, and from the capacity K. We refer to actions $\bar{r}(h)$ and $r(h)$ as *capacity reads* whenever $h \in K$. The idea is that these reads simulate a read of a value written by a copy of $\mathcal{C}$.
3. The K-component of $\mathcal{C}^\kappa$ and $\mathcal{D}^\kappa$ is determined by the sequence of $\nu(h)$-moves. An execution of $\mathcal{D}^\kappa$ can have at most one $\nu(h)$ action for every $h \in G$.

Lemma 4 below formulates the condition on traces of the transition system $\mathcal{D}^\kappa$ that correspond to traces in the $(\mathcal{C}, \mathcal{D})$-system.

Notation. We will use the convention of writing $\Sigma_{\mathcal{D},\nu}$ for $\Sigma_{\mathcal{D}} \cup \Sigma_\nu$. Similarly for $\Sigma_{\mathcal{C},\nu}$ and $\Sigma_{\mathcal{C},\mathcal{D},\nu}$. By $v|_\Sigma$ we will denote the subword of v obtained by erasing the symbols not in Σ.

▶ **Definition 3.** A trace $v\,\nu(h) \in \Sigma^*_{\mathcal{D},\nu}$ is *$\mathcal{C}$-supported* if there exists a word $u \in \Sigma^*_{\mathcal{C},\mathcal{D},\nu}$ such that

$$u|_{\Sigma_{\mathcal{D},\nu}} = v \qquad \text{and} \qquad u\,\nu(h)w(h) \in L(\mathcal{C}^\kappa).$$

A trace $v \in \Sigma^*_{\mathcal{D},\nu}$ is *totally $\mathcal{C}$-supported* if every prefix $v'\nu(h)$ of v is $\mathcal{C}$-supported.

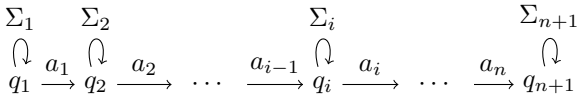


Figure 3 A pattern in $\mathcal{D}^\kappa\downarrow$.

The intuition behind this definition is that every $\nu(h)$ in a trace of $\mathcal{D}^\kappa$ should be supported by a run of contributors witnessing that the write action $w(h)$ is indeed possible.

▶ **Lemma 4.** *There is a #-trace in a $(\mathcal{C},\mathcal{D})$-system if and only if there is a totally $\mathcal{C}$-supported #-trace in $\mathcal{D}^\kappa$.*

Lemma 4 tells us that in order to find a #-trace in $(\mathcal{C},\mathcal{D})$-system we need to find a #-trace in $\mathcal{D}^\kappa$ and verify that it is supported. We will now see that we can actually work with the set of subwords of $\mathcal{D}^\kappa$. This is important, as for every language the set of its subwords is a regular language. Moreover, the minimal automaton for the downward closure has a very simple form. The main technical result of this section says that our initial problem reduces to finding a particular pattern in this minimal automaton.

▶ **Lemma 5.** *If $v_1\nu(h_1)\dots v_i\nu(h_i)$ is a $\mathcal{C}$-supported trace and v_j is a subword of $v'_j \in \Sigma^*_{\mathcal{D}}$ for every $j = 1,\dots,i$, then $v'_1\nu(h_1)\dots v'_i\nu(h_i)$ is $\mathcal{C}$-supported.*

The *downward closure* of language L, denoted $L\downarrow$, is the set of subwords of the words in L. In the following, we denote by $\mathcal{D}^\kappa\downarrow$ the minimal automaton accepting the downward closure of the set of traces from $\mathcal{D}^\kappa$. Every minimal automaton accepting a downward closed language is a graph where the only cycles are self-loops on some states. So every word accepted by $\mathcal{D}^\kappa\downarrow$ comes from a pattern of the form in Figure 3, where on each q_i there is a self-loop on letters from some, possibly empty, alphabet $\Sigma_i \subseteq \Sigma_{\mathcal{D}}$. Note that actions of the form $\nu(h)$ do not occur on self-loops, since by observation 3 on page 78 their number is bounded in any trace from $\mathcal{D}$.

As $\mathcal{D}^\kappa\downarrow$ is a finite automaton, there are finitely many such patterns. We can thus take patterns one by one, and check if there is one that determines a totally $\mathcal{C}$-supported trace. The problem is that to check this we need to fix a trace in advance, and it is not clear how to do this since we have no bound on the length of a fully supported trace. The definition of compatible patterns and the lemma that follows go around this problem.

▶ **Definition 6.** Consider a pattern as in Figure 3. The pattern is *$\mathcal{C}$-compatible* up to position i if for every $j = 1,\dots,i$ there are words $v_j \in \Sigma_j^*$ such that $v_0a_1\dots v_ia_i$ is $\mathcal{C}$-supported. The pattern is *fully $\mathcal{C}$-compatible* if for every $i = 1,\dots,n$ such that $a_i = \nu(h)$ for some h, the pattern is $\mathcal{C}$-compatible up to position i. A *#-pattern* is one ending with $a_n = \bar{w}(\#)$.

The difference between the above definition and Definition 3 is that in the latter we work with a single trace that is $\mathcal{C}$-supported. In Definition 6 we may have to consider different $\mathcal{C}$-supported traces for distinct positions of the pattern. This is necessary, because we cannot fix in advance a trace for all positions of the pattern.

▶ **Lemma 7.** *There is a totally $\mathcal{C}$-supported #-trace in $\mathcal{D}^\kappa$ iff there is a fully $\mathcal{C}$-compatible #-pattern in $\mathcal{D}^\kappa\downarrow$.*

Lemma 7 together with Lemma 4 reduces our reachability problem to the problem of finding a fully $\mathcal{C}$-compatible #-pattern in a finite automaton $\mathcal{D}^\kappa\downarrow$.

5 A general decidability result

In this section we present the main result of the paper giving conditions under which the reachability problem for $(\mathcal{C}, \mathcal{D})$-systems is decidable. The theorem refers to the properties of classes of transition systems defined on page 75. We also discuss the hypotheses of the theorem as well its applicability referring back to examples from Figure 1.

▶ **Theorem 8.** *Suppose $Class_{\mathcal{C}}$, $Class_{\mathcal{D}}$ are two classes of transition systems closed under synchronized products with finite automata. If $Class_{\mathcal{C}}$ has a decidable reachability problem, and $Class_{\mathcal{D}}$ has effective downward closure then the reachability problem for $(\mathcal{C}, \mathcal{D})$-systems, with $\mathcal{C}$ from $Class_{\mathcal{C}}$ and $\mathcal{D}$ from $Class_{\mathcal{D}}$, is decidable.*

Proof. We describe an algorithm deciding the reachability problem for $(\mathcal{C}, \mathcal{D})$-systems. Given $\mathcal{C}$ and $\mathcal{D}$, the algorithm first computes $\mathcal{C}^\kappa$ and $\mathcal{D}^\kappa$ as defined on page 78. The definition of $\mathcal{D}^\kappa$ tells us that it can be obtained by first extending $\mathcal{D}$ with actions in Σ_ν and then making a product with a finite automaton that takes care of the capacity set and the register content (similar for $\mathcal{C}^\kappa$). Then the algorithm computes the finite automaton $\mathcal{D}^\kappa{\downarrow}$ for the downward closure of $\mathcal{D}^\kappa$. These operations are effective since $\mathcal{D}$ is in $Class_{\mathcal{D}}$.

In the next step the algorithm examines all #-patterns in $\mathcal{D}^\kappa{\downarrow}$ of the form:

$$\begin{array}{ccccccccc} & \Sigma_1 & & \Sigma_2 & & & & \Sigma_{n+1} & \\ & \circlearrowleft & & \circlearrowleft & & & & \circlearrowleft & \\ q_1 & \xrightarrow{a_1} & q_2 & \xrightarrow{a_2} & \cdots & \xrightarrow{a_n} & q_{n+1} & \xrightarrow{\#} & q_{n+2} \end{array}$$

and checks if there is one that is fully $\mathcal{C}$-compatible. The algorithm answers yes if and only if it finds a #-pattern in $\mathcal{D}^\kappa{\downarrow}$ that is fully $\mathcal{C}$-compatible. Observe that by Lemma 7 this holds iff $\mathcal{D}^\kappa$ has a fully $\mathcal{C}$-supported #-trace, and thus by Lemma 4 iff there exists a #-trace in the $(\mathcal{C}, \mathcal{D})$-system.

To complete the proof, we need to show how to check that a pattern as above is fully $\mathcal{C}$-compatible. Let $k_1, \ldots, k_l$ be the indices such that a_{k_i} is of the form $\nu(h_i)$. The algorithm checks if the prefix of the pattern up to a_{k_i} is $\mathcal{C}$-compatible for $i = 1, \ldots, l$.

For each $i = 1, \ldots, l$, the check proceeds as follows. First, starting from the pattern up to a_{k_i} it constructs the finite automaton accepting $\Gamma_1^* a_1 \ldots \Gamma_{k_i}^* \nu(h_i) w(h_i)$ where $\Gamma_j = \Sigma_j \cup \Sigma_{\mathcal{C}}$ for $j = 1, \ldots, k_i$. Then, it takes the synchronized product of the resulting automaton with $\mathcal{C}^\kappa$. Denote it with $(\mathcal{C}^\kappa)_i$. The final step of the check is to test for reachability of $w(h_i)$ in $(\mathcal{C}^\kappa)_i$. Note that this can be done by hypothesis since from the properties of $Class_{\mathcal{C}}$, $(\mathcal{C}^\kappa)_i$ is still in $Class_{\mathcal{C}}$. In fact, the test succeeds iff the pattern up to a_{k_i} is $\mathcal{C}$-compatible. This concludes the proof. ◀

In Figure 1 we have listed some concrete instances of $(\mathcal{C}, \mathcal{D})$-systems for which the reachability problem is decidable thanks to Theorem 8. For all the listed classes the closure under synchronized products required by the theorem is immediate, since all of them have finite control.

The effective downward closure, and thus effective reachability problem, holds for pushdown automata [12], Petri net languages [18], stacked counter automata [31], and higher-order pushdown automata of order 2 [30].

Multi-stack pushdown automata (MPA) are Turing powerful already with two stacks. There are though subclasses of MPA with a decidable reachability problem such as path-tree MPA, and scope-bounded MPA [26, 27]. The former include bounded-phase MPA and ordered MPA. For all these classes it is known that visibly multi-pushdown languages have effective semilinear Parikh images. Note that since a run of an MPA is a word over a visible

alphabet, there is a simple reduction that allows to show semilinearity of Parikh images also for the non visibly-pushdown languages accepted by any of these classes. Then Corollary 9 below implies decidability for these classes of MPA.

Reachability is decidable for lossy channel systems [1] and higher-order pushdown automata with collapse [20]. Lossy channel systems do not have effective downward closure: this can be seen by a rather direct reduction from the problem of deciding boundedness of the set of reachable configurations [28]. For higher-order pushdown automata it is not known if the downward closure is effective.

Theorem 8 makes several assumptions about the classes $Class_{\mathcal{C}}$ and $Class_{\mathcal{D}}$. It is worth examining them closer.

The closure property of the theorem, closure under synchronized product, is implied by closure under rational transductions. Given two alphabets Σ and Γ, a *rational transduction* from Σ to Γ is the subset of $\Sigma^* \times \Gamma^*$ generated by a finite-state transducer.

A class of languages is *closed under rational transductions* if for every language L in the class, and every finite-state transducer T the image of L under T is in the class. Observe that the closure under synchronized products with finite automata does not imply closure under projections, and more generally under homomorphisms. So the closure requirements of our theorem are weaker than the closure under rational transductions.

Having an effective downward closure is an interesting condition in itself that probably deserves to be better understood. Zetzsche [30] has recently shown that a sufficient condition for a class to have an effective downward closure is to be closed under rational transductions and to have *effective semilinear Parikh images*. The latter means that there is an algorithm that given a description of a transition system calculates a semilinear representation of the Parikh image of the language of the transition system. A closer examination of his argument shows that our closure under synchronized product with finite automata, together with effective semilinear Parikh images, already implies effective downward closure. Thus in our theorem we can replace the requirement that $Class_{\mathcal{D}}$ has effective downward closure by effective semilinear Parikh images.

▶ **Corollary 9.** *Suppose that $Class_{\mathcal{C}}, Class_{\mathcal{D}}$ are two classes of transition systems closed under synchronized products with finite automata. If $Class_{\mathcal{C}}$ has a decidable reachability problem and $Class_{\mathcal{D}}$ has effective semilinear Parikh images then the reachability problem for $(\mathcal{C}, \mathcal{D})$-systems, with $\mathcal{C}$ from $Class_{\mathcal{C}}$ and $\mathcal{D}$ from $Class_{\mathcal{D}}$, is decidable.*

Some requirements of the theorem, as the closure under products with finite automata seem rather unavoidable. Observe that if, for example, we take $Class_{\mathcal{D}}$ to be the class of process algebra processes, then the register can act as a common state making the reachability problem undecidable even for the case when $Class_{\mathcal{C}}$ is a trivial class containing one process that does nothing. Clearly, the same holds for more general rewriting as process rewrite and term rewriting systems.

Another example of a class that is not closed under products with finite automata is the class of context-free FIFO rewriting systems (that has an effective downward closure though [2]). Our theorem cannot be applied with this class as $Class_{\mathcal{C}}$ or $Class_{\mathcal{D}}$, and we do not know if the reachability problem becomes undecidable in this case.

6 Complexity issues

We have not yet discussed complexity issues. One of the reasons why the algorithm from the proof of Theorem 8 may not be optimal is that it requires to generate the downward closure

explicitly. Here we consider two instances where the downward closure can be generated on-the-fly. The two results of this section are already known [16]. Our purpose is to indicate that our approach is algorithmically interesting, and gives arguably more transparent proofs.

The first result is quite immediate thanks to the accumulator semantics.

▶ **Proposition 10.** *The reachability problem for $(\mathcal{C}, \mathcal{D})$-systems is in* NP *when $\mathcal{C}$ ranges over finite state systems, and $\mathcal{D}$ over pushdown systems.*

Proof. The claim is obvious if $\mathcal{D}$ is also finite state, since the first component in the accumulator grows monotonically. Thus, a #-trace from $(\{s_{init}\}, t_{init}, g_{init})$ in the accumulator semantics can be guessed in polynomial time. If $\mathcal{D}$ is a pushdown system, then a trace in the accumulator semantics splits in $\leq |S|$ phases, where the accumulator component is constant in each phase. So this reduces to (1) guessing the sequence of A-values and (2) a reachability question for a pushdown system executing in $\leq |S|$ phases; each phase corresponding to a particular value of the accumulator. In a phase with value A, the value of the register can change via ϵ-transitions corresponding to contributor writes from states in A. ◀

The second result solves the case when both $\mathcal{D}$ and $\mathcal{C}$ are pushdown systems.

▶ **Theorem 11.** *The reachability problem for $(\mathcal{C}, \mathcal{D})$-systems is in* PSPACE *when both $\mathcal{C}$ and $\mathcal{D}$ range over pushdown systems.*

The proof of this result starts with a construction by Courcelle [12] that provides an exponential size NFA for $\mathcal{D}^{\kappa}\downarrow$; moreover the transitions of the automaton can be computed on-the-fly in PSPACE.

Let $\sigma = h_1, \ldots, h_i$ be a sequence of pairwise distinct values from G and $1 \leq j \leq i$. Let $\mathcal{F}_j$ be a pushdown automaton accepting $\mathcal{C}$-supported words over $\Sigma_{\mathcal{D},\nu}$ that contain exactly the prefix of length j of σ as the occurrences of Σ_ν-symbols. So $\mathcal{F}_j$ accepts words of the form

$$v_1\nu(h_1)\ldots v_{j-1}\nu(h_{j-1})v_j\nu(h_j)\,, \qquad v_k \in \Sigma^*_{\mathcal{D}} \tag{1}$$

and can be obtained essentially as the projection of $\mathcal{C}^{\kappa}$ on $\Sigma_{\mathcal{D},\nu}$ augmented with a check that the occurrences from Σ_ν correspond to $h_1, \ldots, h_j$ (we also need to check for a transition on $w(h_j)$ from the state entered after reading the last Σ_ν). Note that once we fix σ, the size of each pushdown $\mathcal{F}_j$ is polynomial in the size of $\mathcal{C}$.

▶ **Lemma 12.** *An NFA $\mathcal{B}_j$ can be effectively constructed such that:*
1. *$L(\mathcal{B}_j) \subseteq L(\mathcal{F}_j)$;*
2. *for every $u \in L(\mathcal{F}_j)$ there is a subword v of u with $v \in L(\mathcal{B}_j)$.*
The NFA $\mathcal{B}_j$ is of size exponential in the size of $\mathcal{C}$ and its transitions can be computed on-the-fly in PSPACE.

For the proof of the above lemma we refer e.g. to Theorem 7 in [16]. A alternative proof is to take the NFA that accepts words generated by a CFG equivalent to $\mathcal{F}_\sigma$ with derivation trees where no variable occurs more than once on any path. Note also that since $L(\mathcal{B}_j) \subseteq L(\mathcal{F}_j)$, the words accepted by this automaton are of the form (1) as well.

We use now Lemmas 5 and 12 in order to replace both $\mathcal{D}$ and $\mathcal{C}$ by NFAs. First we guess a sequence $\sigma = h_1, \ldots, h_i$ of distinct register values.

For every $1 \leq j \leq i$ let $\widehat{\mathcal{F}}_j$ be an NFA accepting extensions of the words accepted by $\mathcal{B}_j$, more precisely the words $u_1\nu(h_1)\ldots u_{j-1}\nu(h_{j-1})u_j\nu(h_j)$ with $u_k \in \Sigma^*_{\mathcal{D}}$ such that there are subwords v_k of u_k with $v_1\nu(h_1)\ldots v_{j-1}\nu(h_{j-1})v_j\nu(h_j) \in L(\mathcal{B}_j)$. Observe that $\widehat{\mathcal{F}}_j$ is of the same size as $\mathcal{B}_j$ and its transitions can also be generated in PSPACE. The next lemma shows that $L(\mathcal{F}_j) = L(\widehat{\mathcal{F}}_j)$ for every $1 \leq j \leq i$.

▶ **Lemma 13.** $L(\mathcal{F}_j) = L(\widehat{\mathcal{F}_j})$.

Proof. From item (2) of Lemma 12 and the definition of $\widehat{\mathcal{F}_j}$, we get $L(\mathcal{F}_j) \subseteq L(\widehat{\mathcal{F}_j})$.

For the other direction take a word $u \in \widehat{\mathcal{F}_j}$. By definition, it is necessarily of the form $u = u_1\nu(h_1)\dots u_{j-1}\nu(h_{j-1})u_i\nu(h_j)$. Moreover there is a word v accepted by $\mathcal{B}_j$ of the form $v = v_1\nu(h_1)\dots v_{j-1}\nu(h_{j-1})v_j w(h_j)$, with v_k subword of u_k for every k. Since by Lemma 12, $L(\mathcal{B}_j) \subseteq L(\mathcal{F}_j)$, we have that v is accepted by $\mathcal{F}_j$. Thus, by Lemma 5, u is $\mathcal{C}$-supported and contains the prefix of length j of σ as sequence of Σ_ν symbols, therefore it is accepted by $\mathcal{F}_j$. ◀

The algorithm required in Theorem 11 nondeterministically guesses on-the-fly a trace in $\mathcal{D}^\kappa{\downarrow}$, and runs simultaneously $\widehat{\mathcal{F}}_1, \dots, \widehat{\mathcal{F}}_i$ on this trace in order to check if it is fully supported according to Lemma 4. Since all the automata can be generated in Pspace the whole algorithm is in Pspace.

7 Conclusions

Parametrized models with decidable reachability problem are relatively rare. We have studied parametrized systems where processes have no identities, and there are no locking mechanisms on the shared memory [19]. The model has turned out to have interesting algorithmic properties: safety analysis is decidable when its components are chosen from a wide range of formal models. Technically, there are two novelties of in our approach: the accumulator semantics, and the use of downward closures. Our result allows for a compositional construction of a formal model of a distributed system, as schematically presented in Figure 2.

This work puts a spotlight on the effective downward closure property. It would be interesting to investigate this property for other models, as for example for higher-order pushdowns. Among other important issues raised by this work are the questions of the complexity of computing downward closures, and in particular of computing them on-the-fly.

It is not clear if there is an elegant characterization of classes $Class_{\mathcal{C}}$ and $Class_{\mathcal{D}}$ for which the reachability problem for $(\mathcal{C}, \mathcal{D})$-systems is decidable. The differences between Corollary 2 and Theorem 8 appear difficult to bridge.

References

1 P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, 1996.

2 Parosh Aziz Abdulla, Luc Boasson, and Ahmed Bouajjani. Effective lossy queue languages. In *ICALP'01*, LNCS, pages 639–651. Springer, 2001.

3 Parosh Aziz Abdulla, Frédéric Haziza, and Lukás Holík. All for the price of few. In *VMCAI'13*, LNCS, pages 476–495. Springer, 2013.

4 Parosh Aziz Abdulla, Noomene Ben Henda, Giorgio Delzanno, Frédéric Haziza, and Ahmed Rezine. Parameterized tree systems. In *FORTE'08*, LNCS, pages 69–83. Springer, 2008.

5 Benjamin Aminof, Tomer Kotek, Sasha Rubin, Francesco Spegni, and Helmut Veith2. Parameterized model checking of rendezvous systems. In *CONCUR'14*, LNCS, pages 109–124. Springer, 2014.

6 Mohamed Faouzi Atig, Ahmed Bouajjani, K. Narayan Kumar, and Prakash Saivasan. On bounded reachability analysis of shared memory systems. In *FSTTCS'14*, volume 29 of *LIPIcs*, pages 611–623. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2014.

7 Mohamed Faouzi Atig, Ahmed Bouajjani, and Shaz Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *Logical Methods in Computer Science*, 7(4):1–48, 2011.

8 Thomas Ball, Sagar Chaki, and Sriram K. Rajamani. Parameterized verification of multithreaded software libraries. In *TACAS'01*, LNCS, pages 158–173. Springer, 2001.

9 Benedikt Bollig, Paul Gastin, and Jana Schubert. Parameterized verification of communicating automata under context bounds. In *Reachability problems – International Workshop*, LNCS, pages 45–57. Springer, 2014.

10 Ahmed Bouajjani, Javier Esparza, Stefan Schwoon, and Jan Strejcek. Reachability analysis of multithreaded software with asynchronous communication. In *FSTTCS'05*, LNCS, pages 348–359. Springer, 2005.

11 Edmund M. Clarke, Orna Grumberg, and Somesh Jha. Verifying parameterized networks. *ACM Trans. Program. Lang. Syst.*, 19(5):726–750, 1997.

12 Bruno Courcelle. On constructing obstruction sets of words. *Bulletin of EATCS*, 1991.

13 Giorgio Delzanno. Parameterized verification and model checking for distributed broadcast protocols. In *ICGT'14*, LNCS, pages 1–16. Springer, 2014.

14 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification. In *STACS'14*, LIPIcs, pages 1–10. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2014.

15 Javier Esparza, Alain Finkel, and Richard Mayr. On the verification of broadcast protocols. In *LICS'99*, pages 352–359. IEEE, 1999.

16 Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. In *CAV'13*, LNCS, pages 124–140. Springer, 2013.

17 S. A. German and P. A. Sistla. Reasoning about systems with many processes. *J. ACM*, 39(3):675–735, 1992.

18 Peter Habermehl, Roland Meyer, and Harro Wimmel. The downward-closure of Petri net languages. In *ICALP'10*, LNCS, pages 466–477. Springer, 2010.

19 Matthew Hague. Parameterised pushdown systems with non-atomic writes. In *FSTTCS'11*, LIPIcs, pages 457–468. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2011.

20 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *LICS'08*, pages 452–461. IEEE, 2008.

21 Vineet Kahlon. Parameterization as abstraction: A tractable approach to the dataflow analysis of concurrent programs. In *LICS'08*, pages 181–192. IEEE, 2008.

22 Alexander Kaiser, Daniel Kroening, and Thomas Wahl. Dynamic cutoff detection in parameterized concurrent programs. In *CAV'10*, LNCS, pages 645–659. Springer, 2010.

23 Yonit Kesten, Amir Pnueli, Elad Shahar, and Lenore D. Zuck. Network invariants in action. In *CONCUR'02*, LNCS, pages 101–115. Springer, 2002.

24 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Model-checking parameterized concurrent programs using linear interfaces. In *CAV'10*, LNCS, pages 629–644. Springer, 2010.

25 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Sequentializing parameterized programs. In *FIT'12*, volume 87 of *EPTCS*, pages 34–47, 2012.

26 Salvatore La Torre, Margherita Napoli, and Gennaro Parlato. Scope-bounded pushdown languages. In *DLT'14*, LNCS, pages 116–128. Springer, 2014.

27 Salvatore La Torre, Margherita Napoli, and Gennaro Parlato. A unifying approach for multistack pushdown automata. In *MFCS'14*, LNCS, pages 377–389. Springer, 2014.

28 Richard Mayr. Undecidable problems in unreliable computations. *TCS*, 1-3(297):337–354, 2003.

29 Kedar S. Namjoshi and Richard J. Trefler. Analysis of dynamic process networks. In *TACAS'15*, LNCS, pages 164–178. Springer, 2015.

30 Georg Zetzsche. An approach to computing downward closures. In *ICALP'15*, LNCS, pages 440–451. Springer, 2015.

31 Georg Zetzsche. Computing downward closures for stacked counter automata. In *STACS'15*, LIPIcs, pages 743–756. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2015.

On the Succinctness of Idioms for Concurrent Programming

David Harel[1], Guy Katz[1], Robby Lampert[2], Assaf Marron[1], and Gera Weiss[3]

1 **Weizmann Institute of Science, Rehovot, Israel**
2 **Mobileye Vision Technologies Ltd., Jerusalem, Israel**
3 **Ben Gurion University, Beer-Sheva, Israel**

Abstract

The ability to create succinct programs is a central criterion for comparing programming and specification methods. Specifically, approaches to concurrent programming can often be thought of as idioms for the composition of automata, and as such they can then be compared using the standard and natural measure for the complexity of automata, descriptive succinctness. This measure captures the size of the automata that the evaluated approach needs for expressing the languages under discussion. The significance of this metric lies, among other things, in its impact on software reliability, maintainability, reusability and simplicity, and on software analysis and verification. Here, we focus on the succinctness afforded by three basic concurrent programming idioms: requesting events, blocking events and waiting for events. We show that a programming model containing all three idioms is exponentially more succinct than non-parallel automata, and that its succinctness is additive to that of classical nondeterministic and "and" automata. We also show that our model is strictly contained in the model of cooperating automata *à la* statecharts, but that it may provide similar exponential succinctness over non-parallel automata as the more general model – while affording increased encapsulation. We then investigate the contribution of each of the three idioms to the descriptive succinctness of the model as a whole, and show that they each have their unique succinctness advantages that are not subsumed by their counterparts. Our results contribute to a rigorous basis for assessing the complexity of specifying, developing and maintaining complex concurrent software.

1998 ACM Subject Classification D.1.3 Concurrent Programming

Keywords and phrases Descriptive Succinctness, Module Size, Automata, Bounded Concurrency

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.85

1 Introduction

As is well known, many measures of computational complexity are used to compare solutions to algorithmic and software development problems. However, when it comes to comparing the methods, languages and tools that are used to construct those solutions, one needs quite different criteria for comparison. One of the main approaches to this, which has been used ever since the Rabin-Scott work on nondeterministic automata [22], is the size of the description. Size comparisons are usually carried out on the finite automata level of detail, and the most common metric, often called *descriptive succinctness* or *state complexity*, is the total number of states needed by the automata to express certain languages.

A large amount of work has been dedicated to descriptive succinctness in recent decades. A few notable models whose succinctness has been studied in detail are nondeterministic and universal automata, alternating automata, reverse automata, unary automata, and also

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 85–99

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

various kinds of grammars and language formalisms (see, e.g., [15] for a survey). These studies have been motivated by the strong connection between succinctness and *software reliability* [20], indicating that succinct software is easier to develop, maintain and reuse. Further, the descriptive succinctness of a model is often connected to the complexity of various decision problems in it [15], and hence can be relevant also to verification problems.

In this paper, we set out to analyze the descriptive succinctness of various idioms used in concurrent programming, seeking, as in most previous studies, exponential gaps in descriptive power. In particular, we study whether the addition of certain idioms to a programming model exponentially improves that model's succinctness, and in what cases. In addition to the considerations mentioned above and to our desire to better understand the fundamental nature of these concurrency idioms, our motivation has another aspect: a careful selection of concurrency idioms may make resulting programs more amenable to formal analysis. Thus, a better characterization of concurrency idioms and of the types of problems which they are suitable for solving could allow programmers to more carefully tailor the programming model used to the problem at hand – on the one hand retaining "just enough" concurrency to efficiently solve the problem, while on the other hand keeping the model simple and amenable to analysis. However, these topics are beyond the scope of this paper; for a broader discussion of the usefulness of keeping concurrency idioms simple in tasks like program repair and compositional verification we refer the interested reader to [9, 11, 12].

Here, we focus on three fundamental concurrency idioms: requesting, blocking and waiting for events (defined formally in Section 2). The requesting and waiting-for idioms are fairly common in discrete-event programming languages, with versions thereof appearing as first-class citizens in, e.g., *publish-subscribe* architectures [6]; whereas the blocking idiom is somewhat less common, appearing, e.g., in the *live sequence charts* (LSCs) formalism [4]. All three idioms can, of course, be implemented in any high level language. Combined, they also form the *behavioral programming* (*BP*) model [13]. Research suggests that using these idioms may lead to simple code modules that are aligned with the specification [13].

Following the required definitions presented in Section 2, the paper's contributions appear in Sections 3 and 4. In Section 3 we study a model containing the requesting, waiting-for and blocking idioms (which we call the $\mathcal{RWB}$ model), and position it in comparison to other well known models. Specifically, we show that $\mathcal{RWB}$ is polynomially expressible as automata with cooperative concurrency *a la* statecharts [5], but that cooperative concurrency can be exponentially more succinct than $\mathcal{RWB}$. We then show that despite this gap, the $\mathcal{RWB}$ model, which affords greater encapsulation, shares some of the cooperative model's strength and offers considerable advantages when compared to non-parallel automata. Next, we show that the succinctness of $\mathcal{RWB}$ is additive to that of classical nondeterminism and universal ("and") nondeterminism, and that a combination of all three features yields a triple-exponential improvement in succinctness. This last result establishes a hierarchy of succinctness relations indicating, e.g., that the (more practical) nondeterministic or universal $\mathcal{RWB}$ models are double-exponentially more succinct than non-parallel automata.

Next, in Section 4, we study the separate contribution of each of $\mathcal{RWB}$'s idioms to the model's descriptive succinctness. We define variants of $\mathcal{RWB}$ in which each of these idioms is omitted, and show that the full $\mathcal{RWB}$ model has exponential succinctness advantages over each of the variants. We also show that each of these downgraded versions has succinctness advantages over one or both of the other downgraded versions and over non-parallel models. This establishes the fact that each of the idioms makes its own unique contribution to succinctness, and is not subsumed by its counterparts. Notable among these results is the fact that event blocking, which is less common as a first-class concurrency idiom, provides

exponential savings in succinctness. Further, we show that the succinctness afforded by each of these three idioms is not of equal power: for instance, the waiting-for idiom is weaker than the requesting one. Related work appears in Section 5, and we conclude with Section 6.

2 Definitions

2.1 The Request-Wait-Block Model

In this work we focus on the *Request-Wait-Block* ($\mathcal{RWB}$) model for concurrent programs. As we mentioned before, the requesting, waiting-for and blocking idioms are common and appear in various models such as *publish-subscribe* architectures [6], *live sequence charts* [4] and *behavioral programming* [13]. Further, research has shown that these idioms often enable programmers to specify and develop systems naturally and incrementally, with components that are aligned with how humans often describe behavior [7, 13]. Still, the $\mathcal{RWB}$ model is not intended to be programmed in directly – rather, it is intended as a formal representation of programs written in higher level languages, for the sake of rigorous analysis.

The formal definitions of the $\mathcal{RWB}$ model are as follows. An $\mathcal{RWB}$-*automaton* consists of orthogonal components called $\mathcal{RWB}$-threads:

▶ **Definition 1.** A *Request-Wait-Block-thread* ($\mathcal{RWB}$-*thread*) is a tuple $\langle Q, \Sigma, \delta, q_0, R, B\rangle$, where Q is a finite set of states, Σ is a finite set of events, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation and q_0 is an initial state. We require that δ be deterministic, i.e. $\langle q, e, q_1\rangle \in \delta \wedge \langle q, e, q_2\rangle \in \delta \implies q_1 = q_2$. For simplicity of notation, we use $\bar{\delta}$ to indicate the effect event e has in state q (or its absence):

$$\bar{\delta}(q,e) = \begin{cases} q' & ;\text{if exists } q' \in Q \text{ such that } \langle q, e, q'\rangle \in \delta \\ q & ;\text{otherwise}\,. \end{cases}$$

The mapping functions $R, B\colon Q \to 2^{\Sigma}$ associate a state with the set of events *requested* and *blocked*, respectively, by the $\mathcal{RWB}$-thread in that state.

Observe that there is no labeling function for waited-for events: the notion of waiting is expressed via the transitions between states. If state q has a transition labeled with event e that was not requested at q, the thread is considered to be waiting for event e in state q.

A composition of $\mathcal{RWB}$-threads yields an $\mathcal{RWB}$-automaton, defined as follows:

▶ **Definition 2.** An $\mathcal{RWB}$-*automaton* (*RWBA*) A over a finite event set Σ is a finite tuple of $\mathcal{RWB}$-threads $\langle T_1, \ldots, T_n\rangle$, denoted $T_i = \langle Q^i, \Sigma^i, \delta^i, q_0^i, R^i, B^i\rangle$, such that $\Sigma^i \subseteq \Sigma$ for all i, and the Q_i state sets are pairwise disjoint.

A *configuration* of an RWBA is the state of its threads, i.e. an element of $Q^1 \times \ldots \times Q^n$. A configuration $\hat{c} = \langle \hat{q}^1, \ldots, \hat{q}^n\rangle$ is a *successor* of configuration $c = \langle q^1, \ldots, q^n\rangle$ with respect to an event $e \in \Sigma$, denoted $c \xrightarrow{e} \hat{c}$, whenever

$$\underbrace{e \in \bigcup_{i=1}^{n} R^i(q^i)}_{e \text{ is requested}} \bigwedge \underbrace{e \notin \bigcup_{i=1}^{n} B^i(q^i)}_{e \text{ is not blocked}} \bigwedge_{i=1}^{n} \Big(\underbrace{(e \in \Sigma^i \implies \hat{q}^i = \bar{\delta}^i(q^i, e))}_{\substack{\text{affected threads read the event} \\ \text{and change state if needed}}} \wedge \underbrace{(e \notin \Sigma^i \implies \hat{q}^i = q^i)}_{\substack{\text{unaffected threads} \\ \text{stay in the same state}}} \Big).$$

Observe that, since the threads have deterministic transition functions, each configuration can have at most one successor with respect to a specific event. It may, however, have multiple successors, each with respect to a different event.

A *run* of A is a sequence of configurations $c_0c_1c_2\ldots$ such that, for all i, c_{i+1} is a successor (with respect to some event) of c_i and $c_0 = \langle q_0^1, \ldots, q_0^n\rangle$ is the initial configuration. A run may be an *infinite* sequence of successive configurations, or a *finite* sequence that ends in a *terminal configuration*, i.e., a configuration with no successors. Every run $r = c_0c_1c_2\ldots$ of an RWBA induces a set $words(r) = \{\sigma \in \Sigma^* \cup \Sigma^\omega : \forall_{0\leq i<|r|}, c_i \xrightarrow{\sigma[i]} c_{i+1}\}$. Note that $words(r) \subseteq \Sigma^*$ or $words(r) \subseteq \Sigma^\omega$, depending on r being finite or infinite, respectively. We say that a word $\sigma \in \Sigma^* \cup \Sigma^\omega$ is *accepted* by an RWBA A if there is a run r of A such that $\sigma \in words(r)$. The *language* of A, denoted $\mathcal{L}(A)$, is the set of all words accepted by A.

The acceptance condition in this definition is simple – all valid runs are accepted. Of course, the formalism can be modified to cater for more elaborate acceptance conditions, such as conventional accepting states or the various acceptance conditions for ω-automata. The motivation for the present choice is that we regard $\mathcal{RWB}$ as representing the underlying models of programming approaches. As such, languages are seen as generated by, rather than accepted by, a program; indeed, we use these two terms interchangeably.

Next, we define our notion of size, to be used in the analysis of the descriptive succinctness of various variants of RWBAs and other models.

▶ **Definition 3.** The size of an $\mathcal{RWB}$-automaton A with threads $\{\langle Q^i, \Sigma^i, \delta^i, q_0^i, R^i, B^i\rangle\}_{i=1}^n$ is $|A| = \sum_{i=1}^n |Q^i| + |\{\langle q, e, \hat{q}\rangle \in \delta^i\}|$, namely the total number of states and transitions in the threads. For simplicity, the requested and blocked events in every state are omitted from the calculation. They contribute no more than $|\Sigma| \cdot |Q^i|$ to the size of each thread, and have no effect on the size's order of magnitude as $|\Sigma|$ is considered constant.

2.2 Finite Parallel Automata

In order to measure the advantages of $\mathcal{RWB}$ and of other parallel models, we define the following non-parallel model to serve as a reference point:

▶ **Definition 4.** A *deterministic looping automaton* (*DLA*) A is a tuple $\langle Q, \Sigma, \delta, q_0\rangle$, where Q is a set of states, Σ is an alphabet, $\delta \subseteq Q \times \Sigma \times Q$ is a deterministic transition relation and $q_0 \in Q$ is an initial state. As it reads an input word, A traverses its states according to δ, in the usual manner. A accepts infinite words, as well as finite words that end in terminal states (states with no successors). A word is rejected if it contains a letter for which there is no matching transition, or if it ends in a non-terminal state. The *language* $\mathcal{L}(A)$ is the set of words accepted by A, and the *size* of A is $|A| = |Q| + |\{\langle q, e, \hat{q}\rangle \in \delta\}|$, namely the number of states plus the number of transitions in A.

We now discuss other parallel models, focusing on the three fundamental notions: *non-determinism* [22] ($\mathcal{E}$-automata) and its dual, *pure parallelism* ($\mathcal{A}$-automata), which when combined yield *alternating automata* [3], and *cooperative* concurrency ($\mathcal{C}$-automata) [5]. The first two notions take the form of $\exists$- and $\forall$-states in alternating automata, whereas cooperative automata play a role in formalisms and languages such as statecharts [8].

All three features – $\mathcal{E}$, $\mathcal{A}$ and $\mathcal{C}$ – may co-exist. Further, it is shown in [5] that each feature contributes exponentially to the succinctness of the model, independently and additively, so that, e.g., $(\mathcal{E}, \mathcal{A}, \mathcal{C})$-automata allow for triple-exponentially more succinct representations than is possible without these features. Below we give the definition of $(\mathcal{E}, \mathcal{A}, \mathcal{C})$-automata; the other models are regarded as restrictions thereof.

▶ **Definition 5.** An *alternating cooperative automaton* (an $(\mathcal{E}, \mathcal{A}, \mathcal{C})$-*automaton*) over a finite alphabet Σ is a tuple $M = \langle M^1, M^2, \ldots, M^n, \Phi\rangle$ where each M^i is a triple $\langle Q^i, \delta^i, q_0^i\rangle$. Q^i

are pairwise-disjoint state sets and q_0^i are the initial states. $\delta^i \subseteq Q^i \times \Sigma \times \Gamma \times Q^i$ are transition relations, where Γ is the set of propositional formulas over the states of all components, $\bigcup_{i=1}^{n} Q^i$. Elements from Γ serve as *guards*: a transition can be applied only if its guard evaluates to true. For example, for $q_1 \in Q^1$ and $q_2 \in Q^2$, the guard $q_1 \wedge \neg q_2$ evaluates to true precisely when component M^1 is in state q_1 and component M^2 is not in state q_2. Finally, $\Phi \in \Gamma$ is the $\mathcal{E}$-condition – a condition that, when true, implies that the configuration is existential (an *$\mathcal{E}$-configuration*); otherwise, the configuration is universal (an *$\mathcal{A}$-configuration*). In [5], these automata include a termination condition as well, but as we deal with the simple variant of looping automata, we may omit it.

A *configuration* of M is an element of $Q^1 \times Q^2 \times \ldots \times Q^n \times (\Sigma^* \cup \Sigma^\omega) \times \mathbb{N}$, indicating the state of each component, the (finite or infinite) input word, and the position of M in that word. A configuration c satisfies a guard condition $\gamma \in \Gamma$ if γ evaluates to true when assigned the states of c. Let $\sigma = \sigma_0\sigma_1 \ldots \in \Sigma^* \cup \Sigma^\omega$ and let $t = \langle q, a, \gamma, p\rangle$ be a transition in δ^i. We say that t is applicable to a configuration $c = \langle q^1, \ldots, q^n, \sigma, j\rangle$ if $\sigma_j = a, q^i = q$ and c satisfies γ. A configuration $\langle p^1, \ldots, p^n, \sigma, m\rangle$ is a successor of c if for each i there is a transition $\langle q^i, \sigma_j, \gamma^i, p^i\rangle \in \delta^i$ that is applicable to c, and $m = j + 1$.

A computation of M on input word σ can be described as a tree. It starts at the initial configuration $\langle q_0^1, q_0^2, \ldots, q_0^n, \sigma, 1\rangle$, and reads a letter. If the state has multiple successors, the computation "splits", and progresses in parallel for all possible successor states. The process then continues. Any infinite path in this tree is said to be *accepting*. A finite path is accepting iff it ends in a terminal configuration (a configuration with no successors). An $\mathcal{E}$-configuration is accepting iff there *exists* an accepting path starting at that state, whereas an $\mathcal{A}$-configuration is said to be accepting iff *every* path starting at that state is accepting. Word σ is accepted by M iff the root of its computation tree is accepting.

If each configuration of M has a single successor (i.e., all transitions are deterministic), we have a $\mathcal{C}$-automaton, which we might call a cooperative automaton. When $n = 1$ it is in fact an $(\mathcal{E}, \mathcal{A})$-automaton: an alternating looping automaton. When $n = 1$ and $\Phi = true$, M is a nondeterministic looping automaton; and when $n = 1$ and $\Phi = false$ it is a universal looping automaton. Finally, when both $n = 1$ and every configuration has a single successor, M is simply a deterministic looping automaton – a DLA.

▶ **Definition 6.** The *size* of an $(\mathcal{E}, \mathcal{A}, \mathcal{C})$-automaton M is defined to be the sum of the sizes of its condition and components; i.e. $|M| = |\Phi| + \sum_{i=1}^{n} |M^i|$, where $|M^i| = |Q^i| + \sum_{\langle q,a,\gamma,p\rangle \in \delta^i} |\gamma|$. A condition's size is defined as the length of the formula that represents it.

2.3 Succinctness Gaps

We next lay out the method of comparing the succinctness of two models. Informally, we say that a computational model $\mathcal{M}_1$ is more succinct than model $\mathcal{M}_2$ if there are programs that have descriptions in $\mathcal{M}_1$ that are significantly smaller than the smallest possible descriptions for those programs in $\mathcal{M}_2$. In this paper we consider a gap to be significant if it is at least exponential. Following [5], we define upper and lower bounds on gaps in succinctness:

▶ **Definition 7.** Let $\mathcal{M}_1, \mathcal{M}_2$ denote two computational models. We write $\mathcal{M}_1 \xrightarrow{p} \mathcal{M}_2$ (resp., $\mathcal{M}_1 \xrightarrow{\cdot} \mathcal{M}_2$) if there is a polynomial p (resp., a polynomial p and a constant $k > 1$) such that for any automaton $M_1 \in \mathcal{M}_1$ of size m there is an automaton $M_2 \in \mathcal{M}_2$ such that $\mathcal{L}(M_1) = \mathcal{L}(M_2)$, and M_2 is of size no more than $p(m)$ (resp., $k^{p(m)}$). In this case, we say that $\mathcal{M}_1$ is *at most polynomially* (resp., *exponentially*) *more succinct than* $\mathcal{M}_2$.

We write $\mathcal{M}_1 \not\xrightarrow{} \mathcal{M}_2$ if there is a family of ω-regular languages L_n, a polynomial p and a constant $k > 1$, such that L_n is accepted by an automaton $M_1 \in \mathcal{M}_1$ of size $p(f(n))$ for some monotonically-increasing function f, but the smallest $M_2 \in \mathcal{M}_2$ accepting it is at least of size $k^{f(n)}$. In this case, we say that $\mathcal{M}_1$ is *at least exponentially more succinct than* $\mathcal{M}_2$.

3 $\mathcal{RWB}$ and Parallel Automata

In this section, we investigate how $\mathcal{RWB}$-automata fare when considered in the context of $\mathcal{E}$-, $\mathcal{A}$- and $\mathcal{C}$-automata; that is, how the special $\mathcal{RWB}$ idioms relate to the conventional idioms of and- and or-nondeterminism and bounded concurrency. We observe that, of the three models, $\mathcal{RWB}$ seems most closely related to $\mathcal{C}$ – as the threads of an RWBA constitute cooperating components running in parallel – although this cooperation is more limited than in the $\mathcal{C}$ model. The first part of this section validates this observation, by proving that $\mathcal{RWB} \xrightarrow{p} \mathcal{C}$, but that $\mathcal{C} \not\xrightarrow{} \mathcal{RWB}$. This establishes a firm succinctness relationship between $\mathcal{C}$ and $\mathcal{RWB}$: the former is strictly stronger.

The proof that $\mathcal{C} \not\xrightarrow{} \mathcal{RWB}$ revolves around *counting* – a task for which the $\mathcal{C}$ model is particularly suited, as it allows one to count to n using automata of size only $O(\log^2 n)$ [5]. As we prove, in the general case of counting, $\mathcal{RWB}$-automata must be of size n, which is exponentially worse. This result gives rise to the question: does $\mathcal{RWB}$ retain any of $\mathcal{C}$'s power, i.e. is it succinctness-wise better than non-parallel automata?

We answer the question in the affirmative, in two parts. First, we show that $\mathcal{RWB}$ shares some of the power of $\mathcal{C}$ automata; e.g., in certain cases it is possible to count to n with $\mathcal{RWB}$-automata of size $O(\log^2 n \cdot \log\log n)$, and so $\mathcal{RWB} \not\xrightarrow{}$ DLA. Second, we study the relationship between $\mathcal{RWB}$ and the $\mathcal{E}$ and $\mathcal{A}$ models, and show that $\mathcal{RWB}$ can sometimes replace $\mathcal{C}$ in $(\mathcal{E}, \mathcal{A}, \mathcal{C})$-automata, while preserving that model's descriptive succinctness.

The relationship between $\mathcal{E}, \mathcal{A}$ and $\mathcal{C}$ has been extensively studied in [5], where it is shown that they are *orthogonal*, i.e. that their descriptive succinctness is independent and additive. In particular, [5] shows that the $(\mathcal{E}, \mathcal{A}, \mathcal{C})$ model offers a tight triple-exponential gap in succinctness compared to non-parallel automata. Our proof that the $(\mathcal{E}, \mathcal{A}, \mathcal{RWB})$ model affords the same triple-exponential gap thus strengthens the original result of [5], as it shows that a model in which components cannot freely observe other components' states, and is thus more encapsulated than $\mathcal{C}$, suffices for obtaining the triple exponential gap.

3.1 $\mathcal{RWB}$-Automata and $\mathcal{C}$-Automata

Of the three models $\mathcal{E}, \mathcal{A}$ and $\mathcal{C}$, it is natural to define $\mathcal{RWB}$ programs in terms of $\mathcal{C}$-automata, as the underlying parallel components of both make transitions that depend on other components. $\mathcal{C}$-automata take the most general form, allowing components to query the internal states of other components. This is established in the following proposition, proven in Appendix A of the supplementary material [10]:

▶ **Proposition 1.** $\mathcal{RWB} \xrightarrow{p} \mathcal{C}$

We next show that the converse does not hold; i.e., that there exists a family of languages that can be expressed succinctly using $\mathcal{C}$-automata, but that the smallest RWBAs that can express them are exponentially larger.

▶ **Proposition 2.** $\mathcal{C} \not\xrightarrow{} \mathcal{RWB}$

Proof. For $n \in \mathbb{N}$, consider the language $L_n = (0+1)^n 0^\omega$. For every n, there exists a $\mathcal{C}$-automaton of size $O(\log^2 n)$ that accepts L_n, as follows. The automaton consists of $\log n$ components, each representing a single bit of a $(\log n)$-bit counter that counts to n. Carries are performed using the guards: bit number $i+1$ moves from state 0 to 1 if and only if all previous bits $1 \dots i$ are in state 1. A final transition occurs when the counter reaches n, into a state that only allows 0s. As the $\log n$ components can have size $\log n$ because of the transition guards, the automaton is of size $O(\log^2 n)$. See [5] for details.

Now, let us consider the same language in the $\mathcal{RWB}$ model. Suppose that an $\mathcal{RWB}$-automaton A with threads $T_1, \dots, T_k$ accepts L_n. We show that at least one of these threads has to have $\Omega(n)$ states, thus proving the claim. Intuitively, the proof relies on the fact that while A reads the n-bit prefix of the word the threads cannot use events to communicate between themselves, and so a single thread has to handle the counting up to n.

Suppose, contrary-wise, that all threads have fewer than n states, and consider the word $\sigma = 0^{n-1} \cdot 1 \cdot 0^\omega \in L_n$. Examine an arbitrary thread T_i as it reads the $\rho = 0^{n-1}$ prefix of σ. By our assumption, thread T_i has fewer than n states. Consequently, by the pigeonhole principle, it has a state s_1 that it will visit at least twice as it reads ρ. The portion of the path of states that it traverses between these two visits, denoted $s_1 \xrightarrow{0} s_2 \xrightarrow{0} \dots \xrightarrow{0} s_{\alpha_i} \xrightarrow{0} s_1$, constitutes a *cycle* of length α_i in the thread's state graph. This holds for every thread T_i, and so all the threads must traverse cycles of lengths $\alpha_1, \dots, \alpha_n$ as they read ρ.

We now use a pumping argument to show that A accepts a word that is not in L_n. Let $\beta = \prod_{i=1}^{n} \alpha_i$. Consider the word $\sigma' = 0^{n-1} \cdot 0^\beta \cdot 1 \cdot 0^\omega$, and its prefix $\rho' = 0^{n-1} \cdot 0^\beta$. The word 0^ω is in L_n, and ρ' is a prefix of this word; hence, the automaton cannot reject the input word after reading ρ'. However, as the threads are traversing cycles of lengths that divide β, they will each be in the same state after reading ρ' as they would be after reading ρ. Thus, as they read the $1 \cdot 0^\omega$ suffix of σ', they would accept the word – just as they would accept σ. Since $\sigma' \notin L_n$, this is a contradiction. ◀

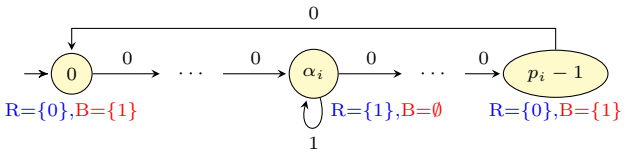
We note that the gap shown by Proposition 2 is tight, in the sense that $\mathcal{C}$-automata are at most (single) exponentially more succinct than $\mathcal{RWB}$-automata. See Appendix B of the supplementary material [10] for the proof.

3.2 Counting with Succinct $\mathcal{RWB}$-Automata

Proposition 2 implies that perhaps the $\mathcal{RWB}$ model is not much stronger than non-parallel automata; indeed, for the task of counting, an RWBA requires as many states as a DLA – exponentially many more than a $\mathcal{C}$-automaton requires. However, the main difference in power between $\mathcal{C}$ and $\mathcal{RWB}$ is in the ability of one component in a $\mathcal{C}$-automaton to observe the state of another without any restrictions, whereas in $\mathcal{RWB}$ a marker event (a *sentinel*) must be triggered for such an observation to be made. Thus, when a sentinel is present, the difference in succinctness between $\mathcal{C}$-automata and $\mathcal{RWB}$-automata diminishes greatly:

▶ **Proposition 3.** *For every $n \in \mathbb{N}$, there exists an $\mathcal{RWB}$-automaton A_n that accepts the language $L_n = 0^n 1^\omega$, such that A_n is of size $O(\log^2 n \cdot \log\log n)$.*

Proof. We use the first appearance of 1 to mark the end of the counting phase. Let $k \in \mathbb{N}$ be the smallest number such that the first k prime numbers $p_1, \dots, p_k$ satisfy $\prod_{i=1}^{k} p_i > n$, and let $\langle \alpha_1, \dots, \alpha_k \rangle$ be defined by $\alpha_i = n \bmod p_i$ for all $1 \le i \le k$. By the Chinese Remainder Theorem, n is the only integer in the range $[1.. \prod_{i=1}^{k} p_i]$ that has these remainders. Consider the $\mathcal{RWB}$-automaton A_n that has k threads $T_1, \dots, T_k$, where thread T_i is given by:



The sets of events requested (R) and blocked (B) in each state are listed by that state. In state α_i the thread requests 1 and blocks nothing, and in the other states it requests 0 and blocks 1. To see that this automaton accepts L_n, note that if even one of the threads is not in its respective α_i state, the next event in any accepted word has to be 0, because that thread requests 0 and blocks 1 and no thread ever blocks 0. On the other hand, once all threads are in their α_i states the only requested event is 1, resulting in a 1^ω suffix. Finally, The Chinese Remainder Theorem guarantees us that the first time the threads are all in their α_i states is precisely at the nth step, as required.

Since we chose the smallest k for which $\prod_{i=1}^{k} p_i > n$, it follows that $k = O(\log n)$. By the Prime Number Theorem we have $p_i = O(i \log i)$. Combining the two, we get that the total size of A_n is indeed $O(\log^2 n \cdot \log\log n)$. ◀

From Proposition 3 it follows that $\mathcal{RWB} \not\xrightarrow{\cdot} \text{DLA}$. Further, because $\mathcal{RWB} \xrightarrow{p} \mathcal{C}$ and $\mathcal{C} \not\xrightarrow{\cdot} \text{DLA}$ [5], we get that $\mathcal{RWB} \not\xrightarrow{\cdot} \text{DLA}$, i.e. that the bound is tight.

3.3 Combining $\mathcal{RWB}$ with $\mathcal{E}$- and $\mathcal{A}$-Automata

One of the main results of [5] establishes a tight triple-exponential gap in succinctness between $(\mathcal{E}, \mathcal{A}, \mathcal{C})$-automata and DLA. Specifically, there exists a family of languages L_n expressible by $(\mathcal{E}, \mathcal{A}, \mathcal{C})$-automata of size $O(\log^2 n)$, but that require at least 2^{2^n} states when expressed by a DLA. In this section we quantify the succinctness gap between the $(\mathcal{E}, \mathcal{A}, \mathcal{RWB})$ model – where $\mathcal{C}$ is replaced by $\mathcal{RWB}$ – and the DLA model.

The semantics of an $(\mathcal{E}, \mathcal{A}, \mathcal{RWB})$-automaton is as follows: as before, the threads run in parallel, and a transition may occur if the event is requested by at least one thread and is blocked by none. In this model, unlike in $\mathcal{RWB}$, we allow nondeterministic transitions in threads, and so a state may have multiple outgoing transitions labeled with the same event. We also adapt the $\mathcal{E}$-condition to operate in an $\mathcal{RWB}$-like fashion, by allowing threads to request/block that a configuration be universal. Thus, a configuration is existential by default, but becomes universal if this was requested by at least one thread and blocked by none (this $\mathcal{E}$-condition is somewhat arbitrary – other definitions could be used as well). Observe that this form of $\mathcal{E}$-condition is a restriction (i.e., a special case) of the $\mathcal{E}$-condition of [5]. The acceptance criteria is the same as for $(\mathcal{E}, \mathcal{A}, \mathcal{C})$ (see Definition 5).

Having shown in Proposition 1 that $\mathcal{RWB} \xrightarrow{p} \mathcal{C}$, it follows that the upper bound of [5] holds; that is, a program in the $(\mathcal{E}, \mathcal{A}, \mathcal{RWB})$ model will incur at most a triple-exponential blowup when transformed appropriately into a DLA. Next, we show that this bound is tight, by establishing a corresponding lower bound. The family of languages that we use is an adaptation of a similar family from [5]:

$$L_n = \{(0+1+\#)^*\#w\#(0+1+\#)^*\#\$w\bot 0^\omega \mid w \in \{0,1\}^n\} \cup \{(0+1+\#)^\omega\}$$

over the alphabet of $\{0, 1, \#, \$, \bot\}$. Intuitively, an automaton that accepts L_n encounters a sequence of words, separated by #s. Then, it encounters a \$, followed by a word w, terminated by $\bot$. The automaton must then decide if this w is of size n, and whether it was encountered before, in the initial sequence of words. If the answer is *yes*, the automaton

accepts the word if it ends in an infinite sequence of 0s; otherwise, it rejects the word. The automaton also accepts all words in which the \$ and $\perp$ signs never appear.

Pigeonhole and pumping arguments show that a non-parallel automaton that recognizes the language has to remember, by the time it reaches the \$ sign, all the words of length n that it has encountered previously. Thus, it must have at least 2^{2^n} states [3, 19]. However, an $(\mathcal{E}, \mathcal{A}, \mathcal{RWB})$-automaton for L_n may be triple-exponentially smaller, as we now show:

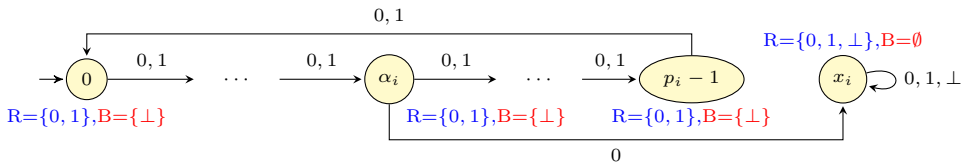
▶ **Proposition 4.** *L_n is recognizable by an $(\mathcal{E}, \mathcal{A}, \mathcal{RWB})$-automaton of size $O(\log^2 n \cdot \log\log n)$.*

Proof. Due to space considerations, we provide here only the core of the proof. Our strategy, inspired by [5], is as follows: in words where the \$ and $\perp$ signs appear, the automaton's nondeterminism is used to "guess" when the first instance of w is encountered. Then, universality is used to compare all n bits of the two occurrences of w simultaneously. Finally, ensuring that both copies of w are of length n, and performing the necessary counting to compare each pair of bits, is performed efficiently by the automaton's $\mathcal{RWB}$-threads.

More explicitly, the $\mathcal{RWB}$ idioms are used for 3 tasks: (1) verifying that the first occurrence of w is of size n; (2) verifying that the second occurrence of w is of size n; and (3) comparing a single pair of bits in the two occurrences of w. Because task (3) is performed universally for all n bits, it ensures that the two occurences of w are equal. For task (3) the automaton counts to n, but is suspended on # and resumed on \$. Thus, when the counting is finished, the next symbol should match the symbol on which the counting was started.

Tasks (1) and (2) can be performed succinctly by an RWBA, as both occurrences of w in L_n are terminated by a sentinel – # or $\perp$. Thus, the construction from Section 3.2 suffices. The automaton size these tasks require is $O(\log^2 n \cdot \log\log n)$. Task (3), however, requires counting *without* a sentinel, which – according to the proof of Proposition 2 – requires an $\mathcal{RWB}$-automaton of size $\Omega(n)$. However, we now show that in an $(\mathcal{E}, \mathcal{A}, \mathcal{RWB})$-automaton such counting can actually be performed succinctly, by leveraging the $\mathcal{E}$ and $\mathcal{A}$ idioms.

Let $k \in \mathbb{N}$ be the smallest number such that the first k prime numbers, $p_1, \ldots, p_k$, satisfy $\prod_{i=1}^{k} p_i > n$. Let $\langle \alpha_1, \ldots, \alpha_k \rangle$ be the tuple of remainders, i.e., $\alpha_i = n \bmod p_i$ for all $1 \leq i \leq k$. By the Chinese Remainder Theorem, these remainders uniquely determine n in the range $[1.. \prod_{i=1}^{k} p_i]$. Suppose, without loss of generality, that the symbol in the first occurrence of w was 0. Then, our goal is to count to n and verify that we reach another 0. Consider an $\mathcal{RWB}$-automaton with k threads $T_1, \ldots, T_k$, where T_i is given by:



All states $0, \ldots, p_i - 1$ request both 0 and 1 and block $\perp$; state x_i requests $0, 1$ and $\perp$. Finally, thread T_i requests that the global configuration be universal if and only if it is at state x_i. The details of suspending the count on # and resuming it on \$ are omitted from the figure, to reduce clutter; this can be performed by associating each state $s \in \{1, \ldots, p_i\}$ with an auxiliary state s', and having the appearance of # send the thread to s', where it loops, until a later appearance of \$ sends it back to s. If the \$ and $\perp$ signs do not appear, then the word is accepted, as it has the form $(0 + 1 + \#)^\omega$, which we included in L_n.

Intuitively, the automaton works as follows. All threads traverse their loops, counting to n. While in these loops, a $\perp$ symbol causes the word to be rejected. Hence, the only way a word that has a $\perp$ sign can be accepted is if all threads escape their loops before reaching $\perp$.

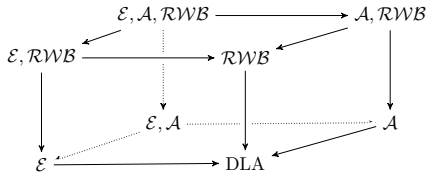


Figure 1 The succinctness hierarchy involving the $\mathcal{E}$, $\mathcal{A}$ and $\mathcal{RWB}$ models, and their combinations. Arrows indicate tight exponential gaps in succinctness. By Proposition 4, the $(\mathcal{E},\mathcal{A},\mathcal{RWB})$ model is at least triple exponentially more succinct than the DLA model; and, applying Proposition 1, it is also at most as succinct as the $(\mathcal{E},\mathcal{A},\mathcal{C})$ model. Combining this with the fact that $(\mathcal{E},\mathcal{A},\mathcal{C})$ is triple exponentially more succinct than DLA [5], we get that the same holds for $(\mathcal{E},\mathcal{A},\mathcal{RWB})$. Thus, any path along the edges of the depicted cube, starting at $(\mathcal{E},\mathcal{A},\mathcal{RWB})$ and ending at DLA, must include precisely 3 exponential gaps. The tight exponential gaps depicted in the figure then follow from known results regarding alternating automata and $\mathcal{C}$-automata [5], combined with Proposition 1.

The only way to escape the counting loops is through the α states. If thread T_i reaches state α_i and reads a 0 symbol, it may escape its loop, assuming the transition is existential; if it is universal, one branch of the thread will remain in the loop, and will reject the word.

The escape transition remains existential until some thread has used it to escape. Afterwards, that thread will remain in its x_i state, requesting that all successive configurations be universal. Hence, all threads must traverse the transition from α_i to x_i simultaneously in order for the word to be accepted. This can only happen if all threads are in their respective α states – which, by the Chinese Remainder Theorem, only occurs at index n – and if the next symbol is the required 0. Hence, since this testing is performed universally for all symbols in w, the word is rejected if even one pair of matching symbols differs.

We stress that this solution is in line with our previous observations that $\mathcal{RWB}$ is weaker than $\mathcal{C}$, in that $\mathcal{RWB}$ cannot succinctly count without a sentinel. In this construction, the behavior threads use the ability of the $\mathcal{E}$-condition semantics to peek into the states of other threads, thus achieving some of the power of the $\mathcal{C}$-automaton guards, and enabling it to count succinctly, even without a sentinel.

As in Proposition 3, analysis shows that the automaton is of size $O(\log^2 n \cdot \log\log n)$. ◀

We have thus established the triple-exponential succinctness gap between $(\mathcal{E},\mathcal{A},\mathcal{RWB})$ and DLA. While $(\mathcal{E},\mathcal{A},\mathcal{RWB})$ is not a practical programming model, we observe that, combined with the results of [5] and Proposition 1, this result immediately establishes a succinctness hierarchy concerning other, more practical models, such as $(\mathcal{E},\mathcal{RWB})$ and $(\mathcal{A},\mathcal{RWB})$. These corollaries are depicted and explained in Figure 1. In particular, these results indicate that the $\mathcal{RWB}$ idioms of requesting, blocking and waiting for events provide a succinctness advantage that is additive and independent of the succinctness provided by the $\mathcal{E}$ and $\mathcal{A}$ idioms – and that the $\mathcal{RWB}$ idioms are not just those of $\mathcal{E}$- or $\mathcal{A}$-automata in disguise. While similar results were previously shown for the $\mathcal{C}$ model [5], our results are stronger as they show that a limited version of $\mathcal{C}$ already suffices to uphold the hierarchy.

From a software-engineering point of view, $\mathcal{C}$-automata afford their succinctness by allowing each component to be aware of the internal state of each of the other components; this liberal awareness is not provided in the $\mathcal{RWB}$ model, resulting in increased module encapsulation, which is usually considered desirable (see, e.g., [21]).

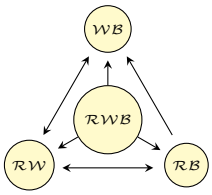


Figure 2 The descriptive succinctness of the $\mathcal{RWB}$, $\mathcal{WB}$, $\mathcal{RW}$ and $\mathcal{RB}$ models, compared to each other. A bi-directional arrow indicates a tight exponential gap in succinctness in both directions – either model may be more succinct than the other. A directed arrow indicates a tight exponential succinctness advantage of the source over the destination, but no such advantage in the reverse direction, i.e., a reverse translation is always possible with only a polynomial blowup.

4 Contributions of the Request, Wait, and Block Idioms

Whereas Section 3 was dedicated to comparing $\mathcal{RWB}$ to other parallel models succinctness-wise, in this section we focus on its internal structure. We study each of its main idioms of requesting, waiting-for, and blocking events, and quantify their contribution to the succinctness afforded by $\mathcal{RWB}$ as a whole. Towards this end, we define the following sub-models:

1. The $\mathcal{WB}$ model: Requesting is omitted. Any event that is not blocked can be triggered. Waiting-for and blocking are allowed. This model can be viewed as having all threads request all events in each state, which, in the notation of Definitions 1 and 2, corresponds to $R^i(q^i) = \Sigma^i$ for every state $q^i \in Q^i$ of thread T_i, for every i.
2. The $\mathcal{RB}$ model: Waiting is omitted; requesting and blocking are allowed. Threads are not informed of events they did not request, and cannot change states when such events are triggered. Formally, for every T_i, if $e \notin R^i(q)$ then $\delta^i(q, e) = q$.
3. The $\mathcal{RW}$ model: Blocking is omitted. Requesting and waiting-for are allowed, and any requested event may be triggered. Formally, $B^i(q) = \emptyset$ for every state q and T_i.

We begin by establishing a simple upper bound, proven in Appendix B of the supplementary material [10]:

▶ **Proposition 5.** *For any* $\mathcal{M}_1, \mathcal{M}_2 \in \{\mathcal{RWB}, \mathcal{WB}, \mathcal{RB}, \mathcal{RW}\}$, $\mathcal{M}_1 \dashrightarrow \mathcal{M}_2$

Next, we establish tight bounds on the difference in succinctness of every pair of these models, as depicted in Figure 2.

We begin by proving that the $\mathcal{RWB}$ model is exponentially more succinct than the $\mathcal{WB}$ variant, i.e., that the event requesting idiom exponentially improves the succinctness of the model.

▶ **Proposition 6.** $\mathcal{RWB} \not\dashrightarrow \mathcal{WB}$

Proof. Let $n \in \mathbb{N}$, and let $k \in \mathbb{N}$ be the smallest number such that the first k prime numbers, $p_1, \ldots, p_k$, satisfy $\prod_{i=1}^{k} p_i > n$. Define the family of languages $L_n = \{\ell_1 \ell_2 \ldots\}$ by:

$$\ell_j = \begin{cases} 0 \text{ or } 1 & ; \quad \exists i \text{ such that } p_i \mid \#_0(\ell_1 \ldots \ell_{j-1}) \\ 0 & ; \quad \text{otherwise} \end{cases}$$

Here $\#_0(\ell_1, \ldots, \ell_{j-1})$ is the number of 0s that have appeared in the word so far. The jth event can be either 0 or 1 if there is a p_i which divides this number. In the $\mathcal{RWB}$ model, this language can be accepted by an automaton of size $O(\log^2 n \cdot \log\log n)$, whereas the smallest

$\mathcal{WB}$-automaton that accepts it is of size at least n (for more details, see Appendix C of the supplementary material [10]). ◀

This proof affords some insight into the power of the requesting idiom. Particularly, requesting allows us to succinctly express *or* conditions – i.e., that an event only be triggered if a disjunction of conditions holds.

We now prove that the waiting idiom also affords exponential succinctness:

▶ **Proposition 7.** $\mathcal{RWB} \not\rightarrow \mathcal{RB}$

Proof. Let $n \in \mathbb{N}$, and consider the family of singleton languages $L_n = 0^n 1^\omega$. Section 3.2 shows an $\mathcal{RWB}$-automaton of size $O(\log^2 n \cdot \log\log n)$ that accepts L_n. However, the smallest $\mathcal{RB}$-automaton that accepts L_n must have a thread of size n; see Appendix D of the supplementary material [10] for details. ◀

The language used for this proof illustrates the power of the waiting idiom. In the basic construction of Section 3.2, each thread would count modulo some prime number, and would, upon the correct remainder, request 1. However, that thread would also wait for a 0 event, thus letting other threads supersede it; if one of them determined that it was not yet time to trigger a 1, they would block 1 and request 0. Without the wait-for idiom, however, a thread cannot observe events it did not request, preventing this sort of inter-thread cooperation.

We now show that $\mathcal{RWB}$ is exponentially more succinct than the $\mathcal{RW}$ variant, i.e. event blocking also yields exponential succinctness. We consider this result to be particularly interesting, as blocking is perhaps the least common, or most special, idiom of $\mathcal{RWB}$.

▶ **Proposition 8.** $\mathcal{RWB} \not\rightarrow \mathcal{RW}$

Proof. Let $n \in \mathbb{N}$, and let $k \in \mathbb{N}$ be the smallest number such that the first k prime numbers, $p_1, \ldots, p_k$, satisfy $\prod_{i=1}^{k} p_i > n$. Observe the languages $L_n = (0^{N-1}(0+1))^\omega$, for $N = \prod_{i=1}^{k} p_i$. In $\mathcal{RWB}$, this language is accepted by an automaton of size $O(\log^2 n \cdot \log\log n)$. In the $\mathcal{RW}$ model, however, an automaton accepting this language must be of size at least n (see Appendix E of the supplementary material [10] for precise details). ◀

The language used for the proof gives some intuition as to the power of the blocking idiom. Particularly, it shows that blocking can succinctly enforce *and* conditions – e.g., that an event is not blocked iff it gives the correct remainder for *all* the primes.

We conclude by examining how the $\mathcal{RWB}$ idioms fare with respect to each other. For example, can requesting be replaced by blocking without having to pay with an exponential decrease in succinctness? Our results, illustrated in Figure 2, show that the $\mathcal{WB}$ and $\mathcal{RW}$ models, and also the $\mathcal{RB}$ and $\mathcal{RW}$ models, are incomparable – i.e., there can be exponential gains in both directions. Also, we prove that the $\mathcal{WB}$ model is weaker than the $\mathcal{RB}$ model, and so, in a way, requesting outpowers waiting. We also show that each of the $\mathcal{WB}$, $\mathcal{RB}$ and $\mathcal{RW}$ models is exponentially more succinct than DLA. For more details, see Appendix F of the supplementary material [10].

5 Related Work

In this paper we focused on studying the $\mathcal{RWB}$ concurrency idioms from a succinctness point of view. For a software-engineering oriented comparison between these $\mathcal{RWB}$ idioms (in the context of Behavioral Programming) and other programming models, see [13] and references therein. Below we discuss some notable related work on descriptive succinctness.

Starting with [22], extensive comparative analysis of expressiveness and succinctness in various models of computations has been carried out. Examples include Büchi, Streett, and Emerson and Lei automata [23], two-way finite automata [24, 2], sweeping automata [16], and – most relevant to the present paper – cooperative automata [5, 14]. Expressiveness and succinctness in timed automata are studied in [1].

The issue of counting to n using unary automata, which played a central role in Section 3, was raised in [19] and has been studied extensively. It is well known that counting requires $\Theta(n)$ states in deterministic and nondeterministic finite automata. As any deterministic unary automaton with n states has an equivalent alternating automaton with $O(\log n)$ states [18], it follows that alternating automata can count with size $O(\log n)$. [17] shows a $\Theta(\sqrt{n})$ bound for counting with universal automata, whereas cooperating automata can count to n with size $O(\log^2 n)$ [5]. Counting in other automata types has also been studied: one-switch alternating automata, for instance, count to n with $O(\log^2 n \cdot \log\log n)$ states [2].

6 Conclusion and Future Work

In this work we set out to analyze the descriptive succinctness afforded by various concurrent programming idioms. Our motivation was the strong connections between the succinctness of the software's description and its simplicity, maintainability, reliability, analysis and verification. We focused on three basic and common idioms – requesting, blocking and waiting for events. We began by analyzing the succinctness of the three idioms taken together, showing that the $\mathcal{RWB}$ model can be translated into cooperating automata with only a polynomial increase in size, but that the converse translation might incur an exponential blowup. Hence, the $\mathcal{RWB}$ model, in which components cannot directly query the state of other components, is strictly less succinct than the $\mathcal{C}$ model. We continued by showing that $\mathcal{RWB}$ can nevertheless succinctly perform non-trivial tasks, that its succinctness is independent and additive to that of the $\mathcal{E}$- and $\mathcal{A}$-automata, and that $(\mathcal{E}, \mathcal{A}, \mathcal{RWB})$-automata are triple-exponentially more succinct than DLA – making them in some cases as strong as the more general $(\mathcal{E}, \mathcal{A}, \mathcal{C})$-automata of [5]. This result established a succinctness hierarchy, indicating the succinctness advantages of models like $(\mathcal{E}, \mathcal{RWB})$ and $(\mathcal{A}, \mathcal{RWB})$. These findings show that the $\mathcal{RWB}$ model, which offers stronger encapsulation and has additional software-engineering advantages over $\mathcal{C}$-automata [13], can sometimes retain the succinctness of the more general model.

We then quantified the contribution of the requesting, waiting-for and blocking idioms to the succinctness of $\mathcal{RWB}$ as a whole. We proved that they are each vital to the succinctness of $\mathcal{RWB}$, as the removal of either may cause an exponential blowup in size, and hence that they do not subsume one another.

The contribution of the present work is thus in substantiating formally the advantages for software engineering that the $\mathcal{RWB}$ idioms, in a variety of programming languages, appear to have; and also in gaining insights into the particular tasks for which each of the idioms is particularly useful. One natural future research direction is to study the succinctness afforded by additional idioms for concurrent programming, such as the lock-step progression idiom, by which all components process a triggered event simultaneously. Another direction is to further study the gap in succinctness between $\mathcal{RWB}$ and $\mathcal{C}$-automata; e.g., to characterize additional tasks, besides counting, in which $\mathcal{C}$'s superiority is manifested.

Acknowledgements. The research of Harel, Katz, Lampert and Marron was partly supported by an Advanced Research Grant from the ERC under the European Community's 7th

Framework Programme (FP7/2007-2013), by an ISF grant, and by the Philip M. Klutznick Fund for Research, the Joachimowicz Fund and the Benoziyo Fund for the Advancement of Science at the Weizmann Institute of Science. The research of Weiss was supported by the Lynn and William Frankel Center for CS at Ben-Gurion University, by a reintegration (IRG) grant under the European Community's FP7 Programme, and by an ISF grant.

References

1 R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

2 J. Birget. Two-Way Automata and Length-Preserving Homomorphisms. *Mathematical Systems Theory*, 29(3):191–226, 1996.

3 A. Chandra, D. Kozen, and L. Stockmeyer. Alternation. *J. Assoc. Comput. Mach.*, 28(1):114–133, 1981.

4 W. Damm and D. Harel. LSCs: Breathing Life into Message Sequence Charts. *J. on Formal Methods in System Design*, 19(1):45–80, 2001.

5 D. Drusinsky and D. Harel. On the Power of Bounded Concurrency I: Finite Automata. *J. Assoc. Comput. Mach.*, 41:517–539, 1994.

6 P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

7 M. Gordon, A. Marron, and O. Meerbaum-Salant. Spaghetti for the Main Course? Observations on the Naturalness of Scenario-Based Programming. In *Proc. 17th Conf. on Innovation and Technology in Computer Science Education (ITICSE)*, pages 198–203, 2012.

8 D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231–274, 1987.

9 D. Harel, A. Kantor, G. Katz, A. Marron, L. Mizrahi, and G. Weiss. On Composing and Proving the Correctness of Reactive Behavior. In *Proc. 13th Int. Conf. on Embedded Software (EMSOFT)*, pages 1–10, 2013.

10 D. Harel, G. Katz, R. Lampert, A. Marron, and G. Weiss. On the Succinctness of Idioms for Concurrent Programming: Supplementary Material. `http://www.wisdom.weizmann.ac.il/~bprogram/doc/Concur15Sup.pdf`.

11 D. Harel, G. Katz, A. Marron, and G. Weiss. Non-Intrusive Repair of Safety and Liveness Violations in Reactive Programs. *Trans. on Computational Collective Intelligence*, 16:1–33, 2014.

12 D. Harel, G. Katz, A. Marron, and G. Weiss. The Effect of Concurrent Programming Idioms on Verification. In *Proc. 3rd Int. Conf. on Model-Driven Engineering and Software Development (MODELSWARD)*, 2015.

13 D. Harel, A. Marron, and G. Weiss. Behavioral programming. *Comm. Assoc. Comput. Mach.*, 55(7):90–100, 2012.

14 T. Hirst and D. Harel. On the Power of Bounded Concurrency II: Pushdown Automata. *J. Assoc. Comput. Mach.*, 41:540–559, 1994.

15 M. Holzer and M. Kutrib. Descriptional and Computational Complexity of Finite Automata – a Survey. *Information and Computation*, 209(3):456–470, 2011.

16 J. Hromkovič and G. Schnitger. Lower Bounds on the Size of Sweeping Automata. *Journal of Automata, Languages and Combinatorics*, 14(1):23–13, 2009.

17 O. Kupferman, A. Ta-Shma, and M. Vardi. *Counting With Automata*, 1999. Tech. Report.

18 E. Leiss. Succinct Representation of Regular Languages by Boolean Automata. *Theoretical Computer Science*, 13:323–330, 1981.

19 A. Meyer and M. Fischer. Economy of Description by Automata, Grammars, and Formal Systems. In *Proc. 12th Sym. on Switching and Automata Theory (SWAT)*, pages 188–191, 1971.

20 J. Musa. *Software Reliability Engineered Testing*. McGraw-Hill, 1998.

21 D. Parnas. On the Criteria to be used in Decomposing Systems into Modules. *Comm. Assoc. Comput. Mach.*, 15(12):1053–1058, 1972.

22 M. Rabin and D. Scott. Finite Automata and Their Decision Problems. *IBM Journal of Research and Development*, 3(2):114–125, 1959.

23 S. Safra and M. Vardi. On ω-Automata and Temporal Logic. In *Proc. 21st Sym. on Theory of Computing (STOC)*, pages 127–137, 1989.

24 W. Sakoda and M. Sipser. Nondeterminism and the Size of Two Way Finite Automata. In *Proc. 10th Sym. on Theory of Computing (STOC)*, pages 275–286, 1978.

Assume-Admissible Synthesis*

Romain Brenguier, Jean-François Raskin, and Ocan Sankur

Université Libre de Bruxelles, Brussels, Belgium

Abstract

In this paper, we introduce a novel rule for synthesis of reactive systems, applicable to systems made of n components which have each their own objectives. It is based on the notion of *admissible* strategies. We compare our novel rule with previous rules defined in the literature, and we show that contrary to the previous proposals, our rule define sets of solutions which are *rectangular*. This property leads to solutions which are robust and resilient. We provide algorithms with optimal complexity and also an abstraction framework.

1998 ACM Subject Classification D.2.4 Software/Program verification, F.3.1 Specifying and Verifying and Reasoning about Programs, I.2.2 Program synthesis

Keywords and phrases Multi-player games, controller synthesis, admissibility

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.100

1 Introduction

The automatic synthesis of reactive systems has recently attracted a considerable attention. The theoretical foundations of most of the contributions in this area rely on two-player zero sum games played on graphs: one player (player 1) models the system to synthesize, and the other player (player 2) models its environment. The game is zero-sum: the objective of player 1 is to enforce the specification of the system while the objective of player 2 is the *negation* of this specification. This is a *worst-case* assumption: because the cooperation of the environment cannot be assumed, we postulate that it is *antagonistic*.

A fully adversarial environment is usually a bold abstraction of reality. Nevertheless, it is popular because it is *simple* and *sound*: a *winning strategy* against an antagonistic player is winning against *any* environment which pursues its own objective. But this approach may fail to find a winning strategy even if there exist solutions when the objective of the environment is taken into account. Also, this model is for two players only: system *vs* environment. In practice, both the system and the environment may be composed of several parts to be constructed individually or whose objectives should be considered one at a time. It is thus crucial to take into account different players' objectives when synthesizing strategies; accordingly, alternative notions have been proposed in the literature.

A first classical alternative is to *weaken* the winning condition of player 1 using the objective of the environment, requiring the system to win only when the environment meets its objective. This approach together with its weaknesses have been discussed in [3], we will add to that later in the paper. A second alternative is to use concepts from n-players *non*-zero sum games. This is the approach taken both by *assume-guarantee synthesis* [6] (AG), and by *rational synthesis* [16] (RS). AG relies on *secure equilibria* [8] (SE), a refinement of Nash equilibria [23] (NE). In SE, objectives are lexicographic: players first try to maximize their own specifications, and then try to falsify the specifications of others. It is shown in [8]

* Supported by the ERC starting grant inVEST (FP7-279499).

that SE are those NE which represent enforceable contracts between the two players. In RS, the system is assumed to be monolithic and the environment is made of components that are *partially controllable*. In RS, we search for a profile of strategies where the system ensures its objective and the players that model the environment are given an "*acceptable*" strategy profiles, from which it is assumed that they will not deviate. "Acceptable" is formalized either by NE, dominating strategies (Dom), or subgame perfect equilibria (SPE).

Contributions. As a first and central contribution, we propose a novel notion of synthesis where we take into account different players' objectives using the concept of *admissible* strategies [1, 2, 4]. For a player with objective ϕ, a strategy σ is *dominated* by σ' if σ' does as well as σ w.r.t. ϕ against all strategies of the other players, and better for some of those strategies. A strategy σ is *admissible* if it is *not* dominated by another strategy. In [2], the admissibility notion was lifted to games played on graphs, and algorithmic questions left open were solved in [4], with the goal of *model checking* the set of runs that survive the iterative elimination of dominated strategies. Here, we use this notion to derive a meaningful notion to *synthesize* systems with several players, with the following idea. *Rational* players should only play admissible strategies since dominated strategies are clearly *suboptimal*. In *assume-admissible synthesis* (AA), we make the assumption that players play admissible strategies. Then for each player, we search for an admissible strategy that is *winning* against all admissible strategies of other players. AA is *sound*: any strategy profile that is winning against admissible strategies of other players, satisfies the objectives of all the players (Theorem 1).

As a second contribution, we compare the different synthesis rules. First we apply all the rules on a simple but representative example, and show the main advantages of AA w.r.t. the other rules. Then we compare systematically the different approaches. We show when a solution for one rule implies a solution for another rule and we prove that, contrary to other rules, AA yields rectangular sets of solutions (Theorem 4). We argue that the rectangularity property is essential for practical applications. As a third contribution, we provide algorithms to decide the existence of assume-admissible winning strategy profiles and prove the optimal complexity of our algorithm (Theorem 8): PSPACE-complete for Müller, and PTIME for Büchi objectives. As a last important contribution, we provide an abstraction framework which allows us to define sufficient conditions to compute sets of winning assume-admissible strategies for each player in the game compositionally (Theorem 12).

Additional pointers to related works. We have already mentioned assume-guarantee synthesis [6] and rational synthesis [16, 20]. Those are the closest related works to ours as they pursue the same scientific objective: to synthesis strategy profiles for non-zero sum multi-player games by taking into account the specification of each player. As those works are defined for similar formal setting, we are able to provide formal statements in the core of the paper that add elements of comparison with our work.

In [15], Faella studies several alternatives to the notion of winning strategy including the notion of admissible strategy. His work is for two-players only, and only the objective of one player is taken into account, the objective of the other player is left unspecified. Faella uses the notion of admissibility to define a notion of *best-effort* in synthesis while we use the notion of admissibility to take into account the objectives of the other players in an n player setting where each player has his own objective.

The notion of admissible strategy is definable in strategy logics [9, 22] and decision problems related to the AA rule can be reduced to satisfiability queries in such logics.

Nevertheless this would not lead to worst-case optimal algorithms. Based on our previous work [4], we develop in this paper worst-case optimal algorithms.

In [12], Damm and Finkbeiner use the notion of *dominant strategy* to provide a compositional semi-algorithm for the (undecidable) distributed synthesis problem. So while we use the notion of admissible strategy, they use a notion of dominant strategy. The notion of dominant strategy is *strictly stronger*: every dominant strategy is admissible but an admissible strategy is not necessary dominant. Also, in multiplayer games with omega-regular objectives with complete information (as considered here), admissible strategies are always guaranteed to exist [2] while it is not the case for dominant strategies. We will show in an example that the notion of dominant strategy is too strong for our purpose. Also, note that the objective of Damm and Finkbeiner is different from ours: they use dominance as a mean to formalize a notion of *best-effort* for components of a distributed system w.r.t. their common objective, while we use admissibility to take into account the objectives of the other components when looking for a winning strategy for one component to enforce its own objective. Additionally, our formal setting is different from their setting in several respects. First, they consider zero-sum games between a distributed team of players (processes) against a unique environment, each player in the team has the same specification (the specification of the distributed system to synthesize) while the environment is considered as adversarial and so its specification is the negation of the specification of the system. In our case, each player has his *own* objective and we do not distinguish between protagonist and antagonist players. Second, they consider distributed synthesis: each individual process has its own view of the system while we consider games with perfect information in which all players have a complete view of the system state. Finally, let us point out that Damm and Finkbeiner use the term *admissible* for specifications and *not* for strategies (as already said, they indeed consider dominant strategies and not admissible strategies). In our case, we use the notion of *admissible* strategy which is classical in game theory, see e.g. [17, 1]. This vocabulary mismatch is unfortunate but we decided to stick to the term of "admissible strategy" which is well accepted in the literature, and already used in several previous works on (multi-player) games played on graphs [2, 15, 4].

Structure of the paper. Sect. 2 contains definitions. In Sect. 3, we review synthesis rules introduced in the literature and define assume-admissible synthesis. In Sect. 4, we consider an example; this allows us to underline some weaknesses of the previous rules. Sect. 5 presents a formal comparison of the different rules. Sect. 6 contains algorithms for Büchi and Müller objectives, and Sect. 7 abstraction techniques applied to our rule.

2 Definitions

A *turn-based multiplayer arena* is a tuple $\mathsf{A} = \langle \mathcal{P}, (\mathsf{S}_i)_{i\in\mathcal{P}}, s_{\mathsf{init}}, (\mathsf{Act}_i)_{i\in\mathcal{P}}, \delta\rangle$ where $\mathcal{P}$ is a finite set of players; for $i \in \mathcal{P}$, S_i is a finite set of player-i states; we let $\mathsf{S} = \biguplus_{i\in\mathcal{P}} \mathsf{S}_i$; $s_{\mathsf{init}} \in \mathsf{S}$ is the initial state; for every $i \in \mathcal{P}$, Act_i is the set of player-i actions; we let $\mathsf{Act} = \bigcup_{i\in\mathcal{P}} \mathsf{Act}_i$; and $\delta\colon \mathsf{S} \times \mathsf{Act} \mapsto \mathsf{S}$ is the transition function. A *run* ρ is a sequence of alternating states and actions $\rho = s_1a_1s_2a_2\ldots \in (\mathsf{S}\cdot\mathsf{Act})^\omega$ such that for all $i \geq 1$, $\delta(s_i, a_i) = s_{i+1}$. We write $\rho_i = s_i$, and $\mathsf{act}_i(\rho) = a_i$. A *history* is a finite prefix of a run ending in a state. We denote by $\rho_{\leq k}$ the history $s_1a_1\ldots s_k$; and write $\mathsf{last}(\rho_{\leq k}) = s_k$, the last state of the history. The set of states *occurring infinitely often* in a run ρ is $\mathrm{Inf}(\rho) = \{s \in \mathsf{S} \mid \forall j \in \mathbb{N}.\ \exists i > j, \rho_i = s\}$.

A *strategy* of player i is a function $\sigma_i : (\mathsf{S}^*\cdot\mathsf{S}_i) \to \mathsf{Act}_i$. A *strategy profile* for the set of players $P \subseteq \mathcal{P}$ is a tuple of strategies, one for each player of P. We write $-i$ for the set

$\mathcal{P} \setminus \{i\}$. Let $\Sigma_i(\mathsf{A})$ be the set of the strategies of player i in A, written Σ_i if A is clear from context, and Σ_P the strategy profiles of $P \subseteq \mathcal{P}$.

A run ρ is *compatible* with strategy σ for player i if for all $j \geq 1$, $\rho_j \in \mathsf{S}_i$ and $\mathsf{act}_j(\rho) = \sigma(\rho_{\leq j})$. It is compatible with strategy profile $\sigma_{\mathcal{P}}$ if it is compatible with each σ_i for $i \in \mathcal{P}$. The *outcome* of a strategy profile $\sigma_{\mathcal{P}}$ is the unique run compatible with $\sigma_{\mathcal{P}}$ starting at s_{init}, denoted $\mathsf{Out}_{\mathsf{A}}(\sigma_{\mathcal{P}})$. We write $\mathsf{Out}_{\mathsf{A},s}(\sigma_{\mathcal{P}})$ for the outcome starting at state s. Given $\sigma_P \in \Sigma_P$ with $P \subseteq \mathcal{P}$, let $\mathsf{Out}_{\mathsf{A}}(\sigma_P)$ denote the set of runs compatible with σ_P, and extend it to $\mathsf{Out}_{\mathsf{A}}(\Sigma')$ where Σ' is a set of strategy profiles. For $E \subseteq \mathsf{S}_i \times \mathsf{Act}_i$, let $\mathsf{Strat}_i(E)$ denote the set of player-i strategies σ that only use actions in E in all outcomes compatible with σ.

An *objective* ϕ is a subset of runs. A strategy σ_i of player i is *winning* for objective ϕ_i if for all $\sigma_{-i} \in \Sigma_{-i}$, $\mathsf{Out}_{\mathsf{A}}(\sigma_i, \sigma_{-i}) \in \phi_i$. A *game* is an arena equipped with an objective for each player, written $\mathsf{G} = \langle \mathsf{A}, (\phi_i)_{i \in \mathcal{P}} \rangle$ where for each player i, ϕ_i is an objective. Given a strategy profile σ_P for the set of players P, we write $\mathsf{G}, \sigma_P \models \phi$ if $\mathsf{Out}_{\mathsf{A}}(\sigma_P) \subseteq \phi$. We write $\mathsf{Out}_{\mathsf{G}}(\sigma_P) = \mathsf{Out}_{\mathsf{A}}(\sigma_P)$, and $\mathsf{Out}_{\mathsf{G}} = \mathsf{Out}_{\mathsf{G}}(\Sigma)$. For any coalition $C \subseteq \mathcal{P}$, and objective ϕ, we denote by $\mathsf{Win}_C(\mathsf{A}, \phi)$ the set of states s such that there exists $\sigma_C \in \Sigma_C$ with $\mathsf{Out}_{\mathsf{G},s}(\sigma_C) \subseteq \phi$.

Although we prove some of our results for general objectives, we give algorithms for ω-regular objectives represented by Muller conditions. A Muller condition is given by a family $\mathcal{F}$ of sets of states: $\phi_i = \{\rho \mid \mathrm{Inf}(\rho) \in \mathcal{F}\}$. Following [19], we assume that $\mathcal{F}$ is given by a Boolean circuit whose inputs are S, which evaluates to true exactly on valuations encoding subsets $S \in \mathcal{F}$. We also use linear temporal logic (LTL) [24] to describe objectives. LTL formulas are defined by $\phi := \mathtt{G}\phi \mid \mathtt{F}\phi \mid \mathtt{X}\phi \mid \phi \mathtt{U} \phi \mid \phi \mathtt{W} \phi \mid S$ where $S \subseteq \mathsf{S}$ (We refer to [14] for the semantics.) We consider the special case of Büchi objectives, given by $\mathtt{GF}(B) = \{\rho \mid B \cap \mathrm{Inf}(\rho) \neq \varnothing\}$. Boolean combinations of formulas $\mathtt{GF}(S)$ define Muller conditions representable by polynomial-size circuits.

In any game G, a player i strategy σ_i is *dominated* by σ_i' if for all $\sigma_{-i} \in \Sigma_{-i}$, $\mathsf{G}, \sigma_i, \sigma_{-i} \models \phi_i$ implies $\mathsf{G}, \sigma_i', \sigma_{-i} \models \phi_i$ and there exists $\sigma_{-i} \in \Sigma_{-i}$, such that $\mathsf{G}, \sigma_i', \sigma_{-i} \models \phi_i$ and $\mathsf{G}, \sigma_i, \sigma_{-i} \not\models \phi_i$, (this is classically called *weak* dominance, but we call it dominance for simplicity). A strategy which is not dominated is *admissible*. Thus, admissible strategies are maximal, and incomparable, with respect to the dominance relation. We write $\mathsf{Adm}_i(\mathsf{G})$ for the set of *admissible* strategies in Σ_i, and $\mathsf{Adm}_P(\mathsf{G}) = \prod_{i \in P} \mathsf{Adm}_i(G)$ the product of the sets of admissible strategies for $P \subseteq \mathcal{P}$.

Strategy σ_i is *dominant* if for all σ_i', and σ_{-i}, $\mathsf{G}, \sigma_i', \sigma_{-i} \phi_i$ implies $\mathsf{G}, \sigma_i, \sigma_{-i} \models \phi_i$. The set of dominant strategies for player i is written $\mathsf{Dom}_i(\mathsf{G})$. A *Nash equilibrium* for G is a strategy profile $\sigma_{\mathcal{P}}$ such that for all $i \in \mathcal{P}$, and $\sigma_i' \in \Sigma_i$, $\mathsf{G}, \sigma_{-i}, \sigma_i' \models \phi_i$ implies $\mathsf{G}, \sigma_{\mathcal{P}} \models \phi_i$; thus no player can improve its outcome by deviating from the prescribed strategy. A Nash equilibrium for G from s, is a Nash equilibrium for G where the initial state is replaced by s. A *subgame-perfect equilibrium* for G is a strategy profile $\sigma_{\mathcal{P}}$ such that for all histories h, $(\sigma_i \circ h)_{i \in \mathcal{P}}$ is a Nash equilibrium in G from state $\mathsf{last}(h)$, where given a strategy σ, $\sigma \circ h$ denotes the strategy $\mathsf{last}(h) \cdot h' \mapsto \sigma(h \cdot h')$.

3 Synthesis Rules

In this section, we review synthesis rules proposed in the literature, and introduce a novel one: the *assume-admissible* synthesis rule (AA). Unless stated otherwise, we fix for this section a game G, with players $\mathcal{P} = \{1, \ldots, n\}$ and their objectives $\phi_1, \ldots, \phi_n$.

Rule Coop. The objectives are *achieved cooperatively* if there is a strategy profile $\sigma_{\mathcal{P}} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ such that $\mathsf{G}, \sigma_{\mathcal{P}} \models \bigwedge_{i \in \mathcal{P}} \phi_i$.

This rule [21, 10] asks for a strategy profile that *jointly* satisfies the objectives of all the players. This rule makes *very strong assumptions*: players fully cooperate and strictly follow their respective strategies. This concept is *not robust* against deviations and postulates that the behavior of every component in the system is *controllable*. This weakness is well-known: see e.g. [6] where the rule is called *weak co-synthesis*.

Rule Win**.** The objectives are *achieved adversarially* if there is a strategy profile $\sigma_{\mathcal{P}} = (\sigma_1, \ldots, \sigma_n)$ such that for all $i \in \mathcal{P}$, $\mathsf{G}, \sigma_i \models \phi_i$.

This rule does *not* require any cooperation among players: the rule asks to synthesize for each player i a strategy which enforces his/her objective ϕ_i against all possible strategies of the other players. Strategy profiles obtained by Win are extremely *robust*: each player is able to ensure his/her objective no matter how the other players behave. Unfortunately, this rule is often not applicable in practice: often, none of the players has a winning strategy against *all* possible strategies of the other players. The next rules soften this requirement by taking into account the objectives of other players.

Rule Win-under-Hyp**.** Given a two-player game G with $\mathcal{P} = \{1, 2\}$ in which player 1 has objective ϕ_1, player 2 has objective ϕ_2, player 1 can *achieve adversarially* ϕ_1 under *hypothesis* ϕ_2, if there is a strategy σ_1 for player 1 such that $\mathsf{G}, \sigma_1 \models \phi_2 \rightarrow \phi_1$.

The rule *winning under hypothesis* applies for two-player games only. Here, we consider the synthesis of a strategy for player 1 against player 2 under the hypothesis that player 2 behaves according to his/her specification. This rule is a relaxation of the rule Win as player 1 is *only* expected to win when player 2 plays so that the outcome of the game satisfies ϕ_2. While this rule is often reasonable, it is *fundamentally* plagued by the following problem: instead of trying to satisfy ϕ_1, player 1 could try to falsify ϕ_2, see e.g. [3]. This problem disappears if player 2 has a winning strategy to enforce ϕ_2, and the rule is then safe. We come back to that later in the paper (see Lemma 1).

Chatterjee et al. in [6] proposed synthesis rules inspired by Win-under-Hyp but avoid the aforementioned problem. The rule was originally proposed in a model with two components and a scheduler. We study here two natural extensions for n players.

Rules $\mathsf{AG}^{\wedge}$ **and** $\mathsf{AG}^{\vee}$**.** The objectives are achieved by

($\mathsf{AG}^{\wedge}$**)** *assume-guarantee-*$\wedge$ if there exists a strategy profile $\sigma_{\mathcal{P}}$ such that

1. $\mathsf{G}, \sigma_{\mathcal{P}} \models \bigwedge_{i \in \mathcal{P}} \phi_i$,
2. for all players i, $\mathsf{G}, \sigma_i \models (\bigwedge_{j \in \mathcal{P} \setminus \{i\}} \phi_j) \Rightarrow \phi_i$.

($\mathsf{AG}^{\vee}$**)** *assume-guarantee-*$\vee$[1] if there exists a strategy profile $\sigma_{\mathcal{P}}$ such that

1. $\mathsf{G}, \sigma_{\mathcal{P}} \models \bigwedge_{i \in \mathcal{P}} \phi_i$,
2. for all players i, $\mathsf{G}, \sigma_i \models (\bigvee_{j \in \mathcal{P} \setminus \{i\}} \phi_j) \Rightarrow \phi_i$.

The two rules differ in the second requirement: $\mathsf{AG}^{\wedge}$ requires that player i wins whenever *all* the other players win, while $\mathsf{AG}^{\vee}$ requires player i to win whenever *one* of the other player wins. Clearly $\mathsf{AG}^{\vee}$ is stronger, and the two rules are equivalent for two-player games. As shown in [8], for two-player games, a profile of strategy for $\mathsf{AG}^{\wedge}$ (or $\mathsf{AG}^{\vee}$) is a Nash equilibrium in a derived game where players want, in lexicographic order, first to satisfy

[1] This rule was introduced in [5], under the name Doomsday equilibria, as a generalization of the AG rule of [6] to the case of n-players.

their own objectives, and then as a secondary objective, want to falsify the objectives of the other players. As NE, $\mathsf{AG}^\wedge$ and $\mathsf{AG}^\vee$ require players to *synchronize* on a particular strategy profiles. As we will see, this is not the case for the new rule that we propose.

[16] and [20] introduce two versions of *rational synthesis* (RS). In the two cases, one of the player, say player 1, models the system while the other players model the environment. The existential version ($\mathsf{RS}^\exists$) searches for a strategy for the system, and a profile of strategies for the environment, such that the objective of the system is satisfied, and the profile for the environment is *stable* according to a solution concept which is either NE, SPE, or Dom. The universal version ($\mathsf{RS}^\forall$) searches for a strategy for the system, such that for all environment strategy profiles that are *stable* according to the solution concept, the objective of the system holds. We write $\Sigma^{\mathsf{NE}}_{G,\sigma_1}$, resp. $\Sigma^{\mathsf{SPE}}_{\mathsf{G},\sigma_1}$, for the set of strategy profiles $\sigma_{-1} = (\sigma_2, \sigma_3, \dots, \sigma_n)$ that are NE (resp. SPE) equilibria in the game G when player 1 plays σ_1, and $\Sigma^{\mathsf{Dom}}_{G,\sigma_1}$ for the set of strategy profiles σ_{-1} where each strategy σ_j, $2 \leq j \leq n$, is dominant in the game G when player 1 plays σ_1.

Rules $\mathsf{RS}^{\exists,\forall}(\mathsf{NE},\mathsf{SPE},\mathsf{Dom})$ **.** Let $\gamma \in \{\mathsf{NE},\mathsf{SPE},\mathsf{Dom}\}$, the objective is achieved by:
($\mathsf{RS}^\exists(\gamma)$) existential rational synthesis under γ if there is a strategy σ_1 of player 1, and a profile $\sigma_{-1} \in \Sigma^\gamma_{\mathsf{G},\sigma_1}$, such that $\mathsf{G}, \sigma_1, \sigma_{-1} \models \phi_1$.
($\mathsf{RS}^\forall(\gamma)$) universal rational synthesis under γ if there is a strategy σ_1 of player 1, such that $\Sigma^\gamma_{\mathsf{G},\sigma_1} \neq \emptyset$, and for all $\sigma_{-1} \in \Sigma^\gamma_{\mathsf{G},\sigma_1}$, $\mathsf{G}, \sigma_1, \sigma_{-1} \models \phi_1$.

Clearly, ($\mathsf{RS}^\forall(\gamma)$) is stronger than ($\mathsf{RS}^\exists(\gamma)$) and more robust. As $\mathsf{RS}^{\exists,\forall}(\mathsf{NE},\mathsf{SPE})$ are derived from NE and SPE, they require players to synchronize on particular strategy profiles.

Novel rule. We now present our novel rule based on the notion of *admissible strategies*.

Rule AA**.** The objectives are achieved by *assume-admissible (*AA*) strategies* if there is a strategy profile $\sigma_\mathcal{P}$ such that:
1. for all $i \in \mathcal{P}$, $\sigma_i \in \mathsf{Adm}_i(\mathsf{G})$;
2. for all $i \in \mathcal{P}$, $\forall \sigma'_{-i} \in \mathsf{Adm}_{-i}(\mathsf{G})$. $\mathsf{G}, \sigma'_{-i}, \sigma_i \models \phi_i$.

A player-i strategy satisfying conditions 1 and 2 above is called *assume-admissible-winning* (AA-*winning*). A profile of AA-winning strategies is an AA-*winning strategy profile*. The rule AA requires that each player has a strategy *winning* against *admissible* strategies of other players. So we assume that players do not play strategies which are *dominated*, which is reasonable as dominated strategies are clearly *suboptimal options*.

Contrary to Coop, $\mathsf{AG}^\wedge$, and $\mathsf{AG}^\vee$, AA does not require that the strategy profile is winning for each player. As for Win, this is a consequence of the definition:

▶ **Theorem 1.** *For all* AA*-winning strategy profile* $\sigma_\mathcal{P}$, $\mathsf{G}, \sigma_\mathcal{P} \models \bigwedge_{i \in \mathcal{P}} \phi_i$.

The condition that AA strategies are admissible is necessary for Thm. 1; it does not suffice to have strategies that are winning against admissible strategies.

4 Synthesis Rules at the Light of an Example

We illustrate the synthesis rules on an example of a real-time scheduler with two tasks. The system is composed of Sched (player 1) and Env (player 2). Env chooses the truth value for r_1, r_2 (r_i is a request for task i), and Sched controls q_1, q_2 (q_i means that task i has been scheduled). Our model is a turn-based game: first, Env chooses a value for r_1, r_2, then in

the next round Sched chooses a value for q_1, q_2, and we repeat forever. The requirements for Sched and Env are as follows:

1. Sched is not allowed to schedule the two tasks at the same time. When r_1 is true, then task 1 must be scheduled (q_1) *within* three rounds. When r_2 is true, task 2 must be scheduled (q_2) in *exactly* three rounds.
2. Whenever Env issues r_i then it does not issue this request again before the occurrence of the grant q_i. Env issues infinitely many requests r_1 and r_2.

We say that a request r_i is *pending* whenever the corresponding grant has not yet been issued. Those requirements can be expressed in LTL as follows:

- $\phi_{\mathsf{Sched}} = \mathtt{G}(r_1 \rightarrow \mathtt{X}q_1 \vee \mathtt{XXX}q_1) \wedge \mathtt{G}(r_2 \rightarrow \mathtt{XXX}q_2) \wedge \mathtt{G}\neg(q_1 \wedge q_2)$.
- $\phi_{\mathsf{Env}} = \mathtt{G}(r_1 \rightarrow \mathtt{X}(\neg r_1 \mathtt{W} q_1)) \wedge \mathtt{G}(r_2 \rightarrow \mathtt{X}(\neg r_2 \mathtt{W} q_2)) \wedge (\mathtt{GF}r_1) \wedge (\mathtt{GF}r_2)$.

A solution compatible with the previous rules in the literature. First, we note that there is no winning strategy neither for Sched, nor for Env. In fact, first let $\hat{\sigma_1}$ be the strategy of Sched that never schedules any of the two tasks, i.e. leaves q_1 and q_2 constantly false. This is clearly forcing $\neg\phi_{\mathsf{Env}}$ against all strategies of Env. Second, let $\hat{\sigma_2}$ be s.t. Env always requests the scheduling of both task 1 and task 2, i.e. r_1 and r_2 are constantly true. It is easy to see that this enforces $\neg\phi_{\mathsf{Sched}}$ against any strategy of Sched. So, there is no solution with rule Win[2]. But clearly those strategies are also not compatible with the objectives of the respective players, so this leaves the possibility to apply successfully the other rules. We now consider a strategy profile which is a solution for all the rules except for AA.

Let (σ_1, σ_2) be strategies for player 1 and 2 respectively, such that the outcome of (σ_1, σ_2) is "Env emits r_1, then Sched emits q_1, Env emits r_2, then Sched waits one round and emits q_2, and repeat." If a deviation from this *exact* execution is observed, then the two players switch to strategies $\hat{\sigma_1}$ and $\hat{\sigma_2}$ respectively, i.e. to the strategies that falsify the specification of the other players. The reader can now convince himself/herself that (σ_1, σ_2) is a solution for Coop, AG and $\mathsf{RS}^{\exists}(\mathsf{NE}, \mathsf{SPE}, \mathsf{Dom})$. Furthermore, we claim that σ_1 is a solution for Win-under-Hyp and $\mathsf{RS}^{\forall}(\mathsf{NE}, \mathsf{SPE}, \mathsf{Dom})$. But, assume now that Env is a device driver which requests the scheduling of tasks by the scheduler of the kernel of an OS when the device that it supervised requires it. Clearly (σ_1, σ_2), which is compatible with all the previous rules (but Win), makes little sense in this context. On the other hand, ϕ_{Sched} and ϕ_{Env} are natural specifications for such a system. So, there is clearly room for other synthesis rules!

Solutions provided by AA, our novel rule. For Env, we claim that the set of *admissible strategies*, noted $\mathsf{Adm}(\phi_{\mathsf{Env}})$, are exactly those that (*i*) do not emit a new request before the previous one has been acknowledged, and (*ii*) do always eventually emit a (new) request when the previous one has been granted. Indeed as we have seen above, Env and Sched can cooperate to satisfy $\phi_{\mathsf{Sched}} \wedge \phi_{\mathsf{Env}}$, so any strategy of Env which would imply the falsification of ϕ_{Env} is dominated and so it is not admissible. Also, we have seen that Env does not have a winning strategy for ϕ_{Env}, so Env cannot do better.

Now, let us consider the following strategy for Sched. (*i*) if pending requests r_1 and r_2 were made one round ago, then grant q_1; if pending requests r_1 and r_2 were made three rounds ago, then behave arbitrarily (it is no more possible to satisfy the specification); (*ii*) if pending request r_2 was made three rounds ago, but not r_1, then grant q_2; (*iii*) if pending r_1 was

[2] Also, it is easy to see that Env does not have a dominant strategy for his specification. So, considering dominant strategies as *best-effort strategies* would not lead to a solution for this example. To find a solution, we need to take into account the objectives of the other players.

made three rounds ago, but not r_2, then grant q_1. We claim that this strategy is *admissible* and while it is *not* winning against *all* possible strategies of Env, it is *winning* against *all admissible* strategies of Env. So, this strategy enforces ϕ_{Sched} against all reasonable strategies of Env w.r.t. to his/her own objective ϕ_{Env}. In fact, there is a whole set of such strategies for Sched, noted $\mathsf{WinAdm}_{\mathsf{Sched}}$. Similarly, there is a whole set of strategies for Env which are both *admissible* and winning against the *admissible* strategies of Sched, noted $\mathsf{WinAdm}_{\mathsf{Env}}$. We prove in the next section that the solutions to AA are *rectangular sets*: they are exactly the solutions in $\mathsf{WinAdm}_{\mathsf{Sched}} \times \mathsf{WinAdm}_{\mathsf{Env}}$. This ensures that AA leads to *resilient* solutions: players do not need to synchronize with the other players on a particular strategy profile but they can arbitrarily choose inside their sets of strategies that are admissible and winning against the admissible strategies of the other players.

5 Comparison of Synthesis Rules

In this section, we compare the synthesis rules to understand which ones yield solutions more often, and to assess their robustness. Some relations are easy to establish; for instance, rules $\mathsf{Win}, \mathsf{AG}^{\vee}, \mathsf{AG}^{\wedge}, \mathsf{AA}$ imply Coop by definition (and Thm. 1). We summarize the implication relations between the rules in Fig. 1. We present the rules $\mathsf{AG}^{\vee}, \mathsf{AG}^{\wedge}$, and the variants of $\mathsf{RS}^{\cdot}(\cdot)$ in one group, respectively. A dashed arrow from A to B means that rule A implies *some* rule in B; while a plain arrow means that A implies *all* rules in B (*e.g.* AA implies $\mathsf{AG}^{\wedge}$ but not $\mathsf{AG}^{\vee}$; while Win implies both rules.) An absence of path means that A does not imply any variant of B. Thus the figure explains which approaches yield solutions more often, by abstracting away the precise variants. The following theorem states the correctness of our diagram.

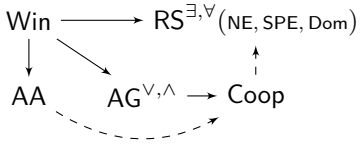


Figure 1 Comparison of synthesis rules.

▶ **Theorem 2.** *The implication relations of Fig. 1 hold.*

In the controller synthesis framework using two-player games between a controller and its environment, some works advocate the use of environment objectives which the environment can guarantee against any controller [7]. Under this assumption, Win-under-Hyp implies AA:

▶ **Lemma 3.** *Let* $\mathsf{G} = \langle \mathsf{A}, \phi_1, \phi_2 \rangle$ *be a two-player game. If player* 2 *has a winning strategy for* ϕ_2 *and Win-under-Hyp has a solution, then* AA *has a solution.*

We now consider the *robustness* of the profiles synthesized using the above rules. An AA-winning strategy profile $\sigma_{\mathcal{P}}$ is robust in the following sense: The set of AA-winning profiles is *rectangular*, *i.e. any* combination of AA-winning strategies independently chosen for each player, is an AA-winning profile. Second, if one replaces *any* subset of strategies in AA-winning profile $\sigma_{\mathcal{P}}$ by arbitrary admissible strategies, the objectives of all the other players still hold. Formally, a *rectangular set* of strategy profiles is a set that is a Cartesian product of sets of strategies, given for each player. A synthesis rule is *rectangular* if the set of strategy profiles satisfying the rule is rectangular. The RS rules require a specific definition since player 1 has a particular role: we say that $\mathsf{RS}^{\forall,\exists}(\gamma)$ is rectangular if for any strategy σ_1 witnessing the rule, the set of strategy profiles $(\sigma_2, \dots, \sigma_n) \in \Sigma^{\gamma}_{\mathsf{G},\sigma_1}$ s.t. $\mathsf{G}, \sigma_1, \dots, \sigma_n \models \phi_1$ is rectangular. We show that apart from AA, only Win and $\mathsf{RS}^{\forall}(\mathsf{Dom})$ are rectangular.

▶ **Theorem 4.** *We have:*

1. *Rule* AA *is rectangular; and for all games* G*, for all* AA*-winning strategy profile* σ_P*, coalition* $C \subseteq \mathcal{P}$*, if* $\sigma'_C \in \mathsf{Adm}_C(\mathsf{G})$*, then* $\mathsf{G}, \sigma_{-C}, \sigma'_C \models \bigwedge_{i \in -C} \phi_i$*.*
2. *The rules* Win *and* $\mathsf{RS}^{\forall}(\mathsf{Dom})$ *are rectangular; the rules* Coop*,* $\mathsf{AG}^{\vee}$*,* $\mathsf{AG}^{\wedge}$*,* $\mathsf{RS}^{\exists}(\mathsf{NE}, \mathsf{SPE}, \mathsf{Dom})$*, and* $\mathsf{RS}^{\forall}(\mathsf{NE}, \mathsf{SPE})$ *are not rectangular.*

6 Algorithm for Assume-Admissible Synthesis

We now recall the characterization of the outcomes of admissible strategy profiles given in [4], and derive algorithms for the AA rule. We use the game of Fig. 2 as a running example for this section. Clearly, none of the players of this game has a winning strategy for his own objective when not taking into account the objective of the other player, but, as we will see, both players have an admissible and winning strategy against the *admissible* strategies of the other player, and so the AA rule applies.

The notion of *value* associated to the states of a game plays an important role in the characterization of admissible strategies and their outcomes [2, 4]. Fix a game G. A state s has value 1 for player i, written $\mathsf{Val}_i(s) = 1$, if player i has a winning strategy from s; $\mathsf{Val}_i(s) = -1$ if for all strategy profiles $\sigma_{\mathcal{P}} \in \Sigma_{\mathcal{P}}$, $\mathsf{Out}_{\mathsf{G},s}(\sigma_{\mathcal{P}})$ does not satisfy ϕ_i; and otherwise $\mathsf{Val}_i(s) = 0$. A *player* j *decreases its own value in history* h if there is a position k such that $\mathsf{Val}_j(h_k) > \mathsf{Val}_j(h_{k+1})$ and $h_k \in \mathsf{S}_j$. We proved in [4], that admissible strategies do not decrease their own values. Let us call such strategies *value-preserving*. In fact, if the current state has value 1, there is a winning strategy which stays within the winning region; if the value is 0, then although other players may force the play into states of value -1, a good strategy for player i will not do this by itself.

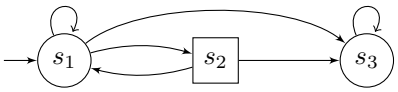


Figure 2 Game G with two players $\mathcal{P} = \{1, 2\}$. Player 1 controls the round states, and has objective $\mathtt{GF}s_2$, and player 2 controls the square state and has objective $\mathtt{GF}s_1$.

▶ **Lemma 5** ([4, Lem. 1])**.** *For all games* G*, players* i*, and histories* ρ*, if* $\mathsf{last}(\rho) \in \mathsf{S}_i$ *and* $\sigma_i \in \mathsf{Adm}_i$ *then* $\mathsf{Val}_i(\delta(\mathsf{last}(\rho), \sigma_i(\rho))) = \mathsf{Val}_i(\mathsf{last}(\rho))$*.*

For player i, let us define the sets $V_{i,x} = \{s \mid \mathsf{Val}_i(s) = x\}$ for $x \in \{-1, 0, 1\}$, which partition S. We define the set of *value-preserving edges* for player i as $E_i = \{(s, a) \in \mathsf{S} \times \mathsf{Act} \mid s \in \mathsf{S}_i \Rightarrow \mathsf{Val}_i(\delta(s, a)) = \mathsf{Val}_i(s)\}$. Observe that value-preserving strategies for player i are exactly those respecting E_i.

In our running example of Fig. 2, it should be clear that any strategy that chooses a transition that goes to s_3 is *not* admissible nor for Player 1 neither for Player 2, as by making this choice both players are condemned to lose their own objective while their other choices leave a chance to win; so the choice of going to s_3 would decrease their own value. So, we can already conclude that Player 2 always chooses $s_2 \mapsto s_1$, his only admissible strategy.

Not all value-preserving strategies are admissible: for Müller objectives, staying inside the winning region does not imply the objective. Moreover, in states of value 0, admissible strategies must visit states where other players can "help" satisfy the objective. Formally, *help states* for player i are other players' states with value 0 and at least two different successors of value 0 or 1. Let $H_i = \{s \in \mathsf{S} \setminus \mathsf{S}_i \mid \mathsf{Val}_i(s) = 0 \wedge \exists s' \neq s''.\ s' \in \delta(s, \mathsf{Act}) \wedge s'' \in \delta(s, \mathsf{Act}) \wedge \mathsf{Val}_i(s') \geq 0 \wedge \mathsf{Val}_i(s'') \geq 0\}$. Given this, the following lemma, adapted from [4], characterizes the outcomes of admissible strategies. We denote by $\mathtt{G}(E_i)$ the set of runs that respect E_i, *i.e.* $\mathtt{G}(\bigvee_{(s,a) \in E_i} s \wedge \mathtt{X}(\delta(s, a)))$.

▶ **Lemma 6.** *For all games* G*, and players* i*,* $\mathsf{Out}_{\mathsf{G}} \cap \Phi_i = \mathsf{Out}_{\mathsf{G}}(\mathsf{Adm}_i, \Sigma_{-i})$*, where* $\Phi_i = \mathtt{G}(E_i) \wedge (\mathtt{GF}(V_{i,1}) \Rightarrow \phi_i) \wedge (\mathtt{GF}(V_{i,0}) \Rightarrow \phi_i \vee \mathtt{GF}(H_i))$.

In our running example of Fig. 2, a strategy of Player 1 which, after some point, always chooses $s_1 \mapsto s_1$ is dominated by strategies that chose infinitely often $s_1 \mapsto s_2$. This is a corollary of the lemma above. Indeed, while all those strategies only visit states with value 0 (and so do not decrease the value for Player 1), the strategy that always chooses $s_1 \mapsto s_1$ has an outcome which is loosing for Player 1 while the other strategies are compatible with outcomes that are winning for Player 1. So, outcome of admissible strategies for Player 1 that always visit states with values 0, also visits s_2 infinitely often. Using the fact that strategies are value-preserving and the last observation, we can now conclude that both players have (admissible) winning strategies against the admissible strategies of the other players. For instance when Player 1 always chooses to play $s_1 \mapsto s_2$, he wins against the admissible strategies of Player 2.

Note that Φ_i can be decomposed into a safety condition $S_i = \mathtt{G}(E_i)$ and a *prefix independent* condition $M_i = (\mathtt{GF}(V_{i,1}) \Rightarrow \phi_i) \wedge (\mathtt{GF}(V_{i,0}) \Rightarrow (\phi_i \vee \mathtt{GF}(H_i))$ which can be expressed by a Müller condition described by a circuit of polynomial size.

For player i, we let $\Omega_i = \mathsf{Out}_{\mathsf{G}}(\mathsf{Adm}_i) \wedge (\mathsf{Out}_{\mathsf{G}}(\mathsf{Adm}_{-i}) \Rightarrow \phi_i)$, which describes the outcomes of admissible strategies of player i, which satisfy objective ϕ_i under the hypothesis that they are compatible with other players' admissible strategies. In fact, it follows from [4] that Ω_i captures the *outcomes* of AA-winning strategies for player i.

▶ **Lemma 7.** *A player* i *strategy is* AA*-winning iff it is winning for objective* Ω_i*.*

Objective Ω_i is not directly expressible as a Müller condition, since Φ_i and $\bigwedge_j \Phi_j$ contain safety parts. Nevertheless, the information whether $\mathtt{G}(E_i)$, or $\mathtt{G}(\cup_{j \neq i} E_j)$ has been violated can be encoded in the state space. Formally, for each player i, we define game G'_i by taking the product of G with $\{\top, 0, \bot\}$; that is, the states are $\mathsf{S} \times \{\top, 0, \bot\}$, and the initial state $(s_{\mathsf{init}}, 0)$. The transitions are defined as for G for the first component; while from state $(s, 0)$, any action a outside E_i leads to $(\delta(s, a), \bot)$, and any action a outside E_j, $j \neq i$, leads to $(\delta(s, a), \top)$. The second component is absorbing at $\bot, \top$. We now rewrite the condition Ω_i for G'_i as $\Omega'_i = \big(\mathtt{GF}(\mathsf{S} \times \{0\}) \wedge M'_i \wedge (\wedge_{j \neq i} M'_j \Rightarrow \phi'_i)\big) \vee (\mathtt{GF}(\mathsf{S} \times \{\top\}) \wedge M'_i)$, where M'_i is the set of runs of G'_i whose projections to G are in M_i, and similarly for ϕ'_i.

Now, checking AA-synthesis is reduced to solving games with Müller conditions. Moreover, we also obtain a polynomial-time algorithm when all objectives are Büchi conditions, by showing that Ω'_i is expressible by a parity condition with four colors.

▶ **Theorem 8.** AA*-synthesis in multiplayer games is* PSPACE*-complete, and* P*-complete for Büchi objectives. Player* i *wins for objective* Ω_i *in* G *iff he wins for objective* Ω'_i *in* G'_i*.*

7 Abstraction

We present abstraction techniques to compute assume-admissible strategy profiles following the *abstract interpretation* framework [11]; see [18] for games. Abstraction is a crucial feature for scalability in practice, and we show here that the AA rule is amenable to abstraction techniques. The problem is not directly reducible to computing AA-winning strategies in abstract games obtained as *e.g.* in [13]; in fact, it can be easily seen that the set of admissible strategies of an abstract game is incomparable with those of the concrete game in general.

Overview. Informally, to compute an AA-winning strategy for player i, we construct an abstract game $\mathcal{A}'_i$ with objective $\underline{\Omega'_i}$ s.t. winning strategies of player i in $\mathcal{A}'_i$ map to AA-winning strategies in G. To define $\mathcal{A}'$, we re-visit the steps of the algorithm of Section 6 by defining approximations computed on the abstract state space. More precisely, we show how to compute under- and over-approximations of the sets $V_{x,k}$, namely $\underline{V}_{x,k}$ and $\overline{V}_{x,k}$, using fixpoint computations on the abstract state space only. We then use these sets to define approximations of the value preserving edges ($\underline{E}_k$ and $\overline{E}_k$) and those of the help states ($\underline{H}_k$ and $\overline{H}_k$). These are then combined to define objective $\underline{\Omega'}_k$ s.t. if player k wins the abstract game for $\underline{\Omega'}_k$, then he wins the original game for Ω'_k, and thus has an AA-winning strategy.

Abstract Games. Consider $\mathsf{G} = \langle \mathsf{A}, (\phi_i)_{i\in\mathcal{P}}\rangle$ with $\mathsf{A} = \langle \mathcal{P}, (\mathsf{S}_i)_{i\in\mathcal{P}}, s_{\mathsf{init}}, (\mathsf{Act}_i)_{i\in\mathcal{P}}, \delta\rangle$ where each ϕ_i is a Müller objective given by a family of sets of states $(\mathcal{F}_i)_{i\in\mathcal{P}}$. Let $\mathsf{S}^{\mathsf{a}} = \biguplus_{i\in\mathcal{P}} \mathsf{S}^{\mathsf{a}}_i$ denote a finite set, namely the *abstract state space*. A *concretization function* $\gamma\colon \mathsf{S}^{\mathsf{a}} \mapsto 2^{\mathsf{S}}$ is a function such that:

1. the abstract states partitions the state space: $\biguplus_{s^{\mathsf{a}}\in\mathsf{S}^{\mathsf{a}}} \gamma(s^{\mathsf{a}}) = \mathsf{S}$,
2. it is compatible with players' states: for all players i and $s^{\mathsf{a}} \in \mathsf{S}^{\mathsf{a}}_i$, $\gamma(s^{\mathsf{a}}) \subseteq \mathsf{S}_i$.

We define the corresponding *abstraction* function $\alpha : \mathsf{S} \to \mathsf{S}^{\mathsf{a}}$ where $\alpha(s)$ is the unique state s^{a} s.t. $s \in \gamma(s^{\mathsf{a}})$. We also extend α, γ naturally to sets of states; and to histories, by replacing each element of the sequence by its image.

We further assume that γ is *compatible* with all objectives $\mathcal{F}_i$ in the sense that the abstraction of a set S is sufficient to determine whether $S \in \mathcal{F}_i$: for all $i \in \mathcal{P}$, for all $S, S' \subseteq \mathsf{S}$ with $\alpha(S) = \alpha(S')$, we have $S \in \mathcal{F}_i \Leftrightarrow S' \in \mathcal{F}_i$. If the objective ϕ_i is given by a circuit, then the circuit for the corresponding abstract objective ϕ^{a}_i is obtained by replacing each input on state s by $\alpha(s)$. We thus have $\rho \in \phi_i$ if, and only if, $\alpha(\rho) \in \phi^{\mathsf{a}}_i$.

The *abstract transition relation* Δ^{a} induced by γ is defined by: $(s^{\mathsf{a}}, a, t^{\mathsf{a}}) \in \Delta^{\mathsf{a}} \Leftrightarrow \exists s \in \gamma(s^{\mathsf{a}}), \exists t \in \gamma(t^{\mathsf{a}}), t = \delta(s, a)$. We write $\mathsf{post}_\Delta(s^{\mathsf{a}}, a) = \{t^{\mathsf{a}} \in \mathsf{S}^{\mathsf{a}} \mid \Delta(s^{\mathsf{a}}, a, t^{\mathsf{a}})\}$, and $\mathsf{post}_\Delta(s^{\mathsf{a}}, \mathsf{Act}) = \cup_{a\in\mathsf{Act}} \mathsf{post}_\Delta(s^{\mathsf{a}}, a)$. For each coalition $C \subseteq \mathcal{P}$, we define a game in which players C play together against coalition $-C$; and the former resolves non-determinism in Δ^{a}. Intuitively, the winning region for C in this abstract game will be an over-approximation of the original winning region. Given C, the *abstract arena* $\mathcal{A}^C$ is $\langle \{C, -C\}, (\mathsf{S}_C, \mathsf{S}_{-C}), \alpha(s_{\mathsf{init}}), (\mathsf{Act}_C, \mathsf{Act}_{-C}), \delta^{\mathsf{a},C}\rangle$, where $\mathsf{S}_C = \big(\bigcup_{i\in C} \mathsf{S}^{\mathsf{a}}_i\big) \cup \big(\bigcup_{i\in\mathcal{P}} \mathsf{S}^{\mathsf{a}}_i \times \mathsf{Act}_i\big)$, $\mathsf{S}_{-C} = \bigcup_{i\notin C} \mathsf{S}^{\mathsf{a}}_i$; and $\mathsf{Act}_C = \big(\bigcup_{i\in C} \mathsf{Act}_i\big) \cup \mathsf{S}^{\mathsf{a}}$ and $\mathsf{Act}_{-C} = \bigcup_{i\in -C} \mathsf{Act}_i$. The relation $\delta^{\mathsf{a},C}$ is given by: if $s^{\mathsf{a}} \in \mathsf{S}^{\mathsf{a}}$, then $\delta^{\mathsf{a},C}(s^{\mathsf{a}}, a) = (s^{\mathsf{a}}, a)$. If $(s^{\mathsf{a}}, a) \in \mathsf{S}^{\mathsf{a}} \times \mathsf{Act}$ and $t^{\mathsf{a}} \in \mathsf{S}^{\mathsf{a}}$ satisfies $(s^{\mathsf{a}}, a, t^{\mathsf{a}}) \in \Delta^{\mathsf{a}}$ then $\delta^{\mathsf{a},C}((s^{\mathsf{a}}, a), t^{\mathsf{a}}) = t^{\mathsf{a}}$; while for $(s^{\mathsf{a}}, a, t^{\mathsf{a}}) \notin \Delta^{\mathsf{a}}$, the play leads to an arbitrarily chosen state u^{a} with $\Delta(s^{\mathsf{a}}, a, u^{\mathsf{a}})$. Thus, from states (s^{a}, a), coalition C chooses a successor t^{a}.

We extend γ to histories of $\mathcal{A}^C$ by first removing states of $(\mathsf{S}^{\mathsf{a}}_i \times \mathsf{Act}_i)$; and extend α by inserting these intermediate states. Given a strategy σ of player k in $\mathcal{A}^C$, we define its *concretization* as the strategy $\gamma(\sigma)$ of G that, at any history h of G, plays $\gamma(\sigma)(h) = \sigma(\alpha(h))$. We write $\mathsf{Win}_D(\mathcal{A}^C, \phi^{\mathsf{a}}_k)$ for the states of S^{a} from which the coalition D has a winning strategy in $\mathcal{A}^C$ for objective ϕ^{a}_k, with $D \in \{C, -C\}$. Informally, it is easier for coalition C to achieve an objective in $\mathcal{A}^C$ than in G, that is, $\mathsf{Win}_C(\mathcal{A}^C, \phi^{\mathsf{a}}_k)$ over-approximates $\mathsf{Win}_C(\mathsf{A}, \phi_k)$:

▶ **Lemma 9.** *If the coalition C has a winning strategy for objective ϕ_k in G from s then it has a winning strategy for ϕ^{a}_k in $\mathcal{A}^C$ from $\alpha(s)$.*

Value-Preserving Strategies. We now provide under- and over-approximations for value-preserving strategies for a given player. We start by computing approximations $\underline{V}_{k,x}$ and $\overline{V}_{k,x}$ of the sets $V_{k,x}$, and then use these to obtain approximations of the value-preserving edges E_k.

Fix a game G, and a player k. Let us define the *controllable predecessors* for player k as $\mathsf{CPRE}_{\mathcal{A}^{\mathcal{P}\setminus\{k\}},k}(X) = \{s^{\mathsf{a}} \in \mathsf{S}^{\mathsf{a}}_k \mid \exists a \in \mathsf{Act}_k, \mathsf{post}_\Delta(s^{\mathsf{a}}, a) \subseteq X\} \cup \{s^{\mathsf{a}} \in \mathsf{S}^{\mathsf{a}}_{\mathcal{P}\setminus\{k\}} \mid \forall a \in \mathsf{Act}_{-k}, \mathsf{post}_\Delta(s^{\mathsf{a}}, a) \subseteq X\}$. We let

$$\begin{aligned}
&\overline{V}_{k,1} = \mathsf{Win}_{\{k\}}(\mathcal{A}^{\{k\}}, \phi^{\mathsf{a}}_k), && \overline{V}_{k,-1} = \mathsf{Win}_{\emptyset}(\mathcal{A}^{\emptyset}, \neg\phi^{\mathsf{a}}_k),\\
&\overline{V}_{k,0} = \mathsf{Win}_{\mathcal{P}\setminus\{k\}}(\mathcal{A}^{\mathcal{P}\setminus\{k\}}, \neg\phi^{\mathsf{a},k}) \cap \mathsf{Win}_{\mathcal{P}}(\mathcal{A}^{\mathcal{P}}, \phi^{\mathsf{a}}_k),\\
&\underline{V}_{k,1} = \mathsf{Win}_{\{k\}}(\mathcal{A}^{\mathcal{P}\setminus\{k\}}, \phi^{\mathsf{a}}_k), && \underline{V}_{k,-1} = \mathsf{Win}_{\emptyset}(\mathcal{A}^{\mathcal{P}}, \neg\phi^{\mathsf{a}}_k)\\
&\underline{V}_{k,0} = \nu X.\big(\mathsf{CPRE}_{\mathcal{A}^{\mathcal{P}\setminus\{k\}},k}(X \cup \underline{V}_{k,1} \cup \underline{V}_{k,-1}) \cap F\big),\\
&\qquad \text{where } F = \mathsf{Win}_{\mathcal{P}\setminus\{k\}}(\mathcal{A}^{\{k\}}, \neg\phi^{\mathsf{a}}_k) \cap \mathsf{Win}_{\mathcal{P}}(\mathcal{A}^{\emptyset}, \phi^{\mathsf{a}}_k).
\end{aligned}$$

The last definition uses the $\nu X.f(X)$ operator which is the greatest fixpoint of f. These sets define approximations of the sets $V_{k,x}$. Informally, this follows from the fact that to define *e.g.* $\overline{V}_{k,1}$, we use the game $\mathcal{A}^{\{k\}}$, where player k resolves itself the non-determinism, and thus has more power than in G. In contrast, for $\underline{V}_{k,1}$, we solve $\mathcal{A}^{\mathcal{P}\setminus\{k\}}$ where the adversary resolves non-determinism. We state these properties formally:

▶ **Lemma 10.** *For all players k and $x \in \{-1, 0, 1\}$, $\gamma(\underline{V}_{k,x}) \subseteq V_{k,x} \subseteq \gamma(\overline{V}_{k,x})$.*

We thus have $\cup_x \gamma(\overline{V}_{k,x}) = \mathsf{S}$ (as $\cup_x V_{k,x} = \mathsf{S}$) but this is not the case for $\underline{V}_{k,x}$; so let us define $\underline{V} = \cup_{j\in\{-1,0,1\}} \underline{V}_{k,j}$. We now define approximations of E_k based on the above sets.

$$\begin{aligned}
\overline{E}_k &= \{(s^{\mathsf{a}}, a) \in \mathsf{S}^{\mathsf{a}} \times \mathsf{Act} \mid s^{\mathsf{a}} \in \mathsf{S}^{\mathsf{a}}_k \Rightarrow \exists x, s^{\mathsf{a}} \in \overline{V}_{k,x}, \mathsf{post}_\Delta(s^{\mathsf{a}}, a) \cap \cup_{l\geq x} \overline{V}_{k,l} \neq \emptyset\},\\
\underline{E}_k &= \{(s^{\mathsf{a}}, a) \in \mathsf{S}^{\mathsf{a}} \times \mathsf{Act} \mid s^{\mathsf{a}} \in \mathsf{S}^{\mathsf{a}}_k \Rightarrow \exists x, s^{\mathsf{a}} \in \underline{V}_{k,x}, \mathsf{post}_\Delta(s^{\mathsf{a}}, a) \subseteq \cup_{l\geq x} \underline{V}_{k,l}\}\\
&\qquad\qquad \cup \{(s^{\mathsf{a}}, a) \mid s^{\mathsf{a}} \notin \underline{V}\}.
\end{aligned}$$

Intuitively, $\overline{E}_k$ is an over-approximation of E_k, and $\underline{E}_k$ under-approximates E_k when restricted to states in $\underline{V}$ (notice that $\underline{E}_k$ contains all actions from states outside $\underline{V}$). In fact, our under-approximation will be valid only inside $\underline{V}$; but we will require the initial state to be in this set, and make sure the play stays within $\underline{V}$. We show that sets $\underline{E}_k$ and $\overline{E}_k$ provide approximations of value-preserving strategies.

▶ **Lemma 11.** *For all games G, and players k, $\mathsf{Strat}_k(E_k) \subseteq \gamma(\mathsf{Strat}_k(\overline{E}_k))$, and if $s_{\mathsf{init}} \in \gamma(\underline{V})$, then $\emptyset \neq \gamma(\mathsf{Strat}_k(\underline{E}_k)) \subseteq \mathsf{Strat}_k(E_k)$.*

Abstract Synthesis of AA-winning strategies. We now describe the computation of AA-winning strategies in abstract games. Consider game G and assume sets $\underline{E}_i, \overline{E}_i$ are computed for all players i. Roughly, to compute a strategy for player k, we will constrain him to play only edges from $\underline{E}_k$, while other players j will play in $\overline{E}_j$. By Lemma 11, any strategy of player k maps to value-preserving strategies in the original game, and all value-preserving strategies for other players are still present. We now formalize this idea, incorporating the help states in the abstraction.

We fix a player k. We construct an abstract game in which winning for player k implies that player k has an effective AA-winning strategy in G. We also define $\mathcal{A}'_k = \langle\{\{k\}, -k\}, (\mathsf{S'}^{\mathsf{a}}_k, \mathsf{S'}^{\mathsf{a}}_{-k} \cup \mathsf{S'}^{\mathsf{a}} \times \mathsf{Act}), \alpha(s_{\mathsf{init}}), (\mathsf{Act}_k, \mathsf{Act}_{-k}), \delta_{\mathcal{A}^k}\rangle$, where $\mathsf{S'}^{\mathsf{a}} = \mathsf{S}^{\mathsf{a}} \times \{\bot, 0, \top\}$; thus we modify $\mathcal{A}^{\mathcal{P}\setminus\{k\}}$ by taking the product of the state space with $\{\top, 0, \bot\}$. Intuitively, as in Section 6, initially the second component is 0, meaning that no player has violated the value-preserving edges. The component becomes $\bot$ whenever player k plays an action outside of $\underline{E}_k$; and $\top$ if another player j plays outside $\overline{E}_j$. We extend γ to $\mathcal{A}'_k$ by $\gamma((s^{\mathsf{a}}, x)) = \gamma(s^{\mathsf{a}}) \times \{x\}$, and extend it to histories of $\mathcal{A}'_k$ by first removing the intermediate states $\mathsf{S'}^{\mathsf{a}} \times \mathsf{Act}$. We thus see $\mathcal{A}'_k$ as an abstraction of $\mathsf{A'}$ of Section 6.

In order to define the objective of $\mathcal{A}'_k$, let us first define approximations of the help states H_k, where we write $\Delta(s^{\mathsf{a}}, \mathsf{Act}, t^{\mathsf{a}})$ to mean $\exists a \in \mathsf{Act}, \Delta(s^{\mathsf{a}}, a, t^{\mathsf{a}})$.

$$\begin{aligned}\overline{H}_k &= \{s^{\mathsf{a}} \in \overline{V}_{k,0} \setminus \mathsf{S}^{\mathsf{a}}_k \mid \exists t^{\mathsf{a}}, u^{\mathsf{a}} \in \overline{V}_{k,0} \cup \overline{V}_{k,1}.\ \Delta(s^{\mathsf{a}}, \mathsf{Act}, t^{\mathsf{a}}) \wedge \Delta(s^{\mathsf{a}}, \mathsf{Act}, u^{\mathsf{a}})\}\\ \underline{H}_k &= \{s^{\mathsf{a}} \in \underline{V}_{k,0} \setminus \mathsf{S}^{\mathsf{a}}_k \mid \exists a \neq b \in \mathsf{Act}, \mathsf{post}_\Delta(s^{\mathsf{a}}, a) \cap \mathsf{post}_\Delta(s^{\mathsf{a}}, b) = \emptyset,\\ &\qquad\qquad \mathsf{post}_\Delta(s^{\mathsf{a}}, a) \cup \mathsf{post}_\Delta(s^{\mathsf{a}}, b) \subseteq \underline{V}_{k,0} \cup \underline{V}_{k,1}\}.\end{aligned}$$

We define the following approximations of the objectives M'_k and Ω'_k in $\mathcal{A}'_k$.

$$\begin{aligned}\underline{M'}_k &= (\mathtt{GF}(\overline{V}_{k,1}) \Rightarrow \phi^{\mathsf{a}}_k) \wedge \left(\mathtt{GF}(\overline{V}_{k,0}) \Rightarrow (\phi^{\mathsf{a}}_k \vee \mathtt{GF}(\underline{H}_k))\right),\\ \overline{M'}_k &= (\mathtt{GF}(\underline{V}_{k,1}) \Rightarrow \phi^{\mathsf{a}}_k) \wedge \left(\mathtt{GF}(\underline{V}_{k,0}) \Rightarrow (\phi^{\mathsf{a}}_k \vee \mathtt{GF}(\overline{H}_k))\right),\\ \underline{\Omega'}_k &= \left(\mathtt{GF}(\mathsf{S}^{\mathsf{a}} \times \{0\}) \wedge \underline{M'}_k \wedge \left(\textstyle\bigwedge_{j \neq k} \overline{M'}_j \Rightarrow \phi^{\mathsf{a}}_k\right)\right) \vee (\mathtt{GF}(\mathsf{S}^{\mathsf{a}} \times \{\top\}) \wedge \underline{M'}_k).\end{aligned}$$

▶ **Theorem 12.** *For all games* G*, and players* k*, if* $s_{\mathsf{init}} \in \underline{V}$*, and player* k *has a winning strategy in* $\mathcal{A}'_k$ *for objective* $\underline{\Omega}'_k$*, then he has a winning strategy in* G'_k *for* Ω_k*; and thus a* AA*-winning strategy in* G*.*

Now, if Theorem 12 succeeds to find an AA-winning strategy for each player k, then the resulting strategy profile is AA-winning.

8 Conclusion

In this paper, we have introduced a novel synthesis rule, called the *assume admissible synthesis*, for the synthesis of strategies in non-zero sum n players games played on graphs with omega-regular objectives. We use the notion of admissible strategy, a classical concept from game theory, to take into account the objectives of the other players when looking for winning strategy of one player. We have compared our approach with other approaches such as assume guarantee synthesis and rational synthesis that target the similar scientific objectives. We have developed worst-case optimal algorithms to handle our synthesis rule as well as dedicated abstraction techniques. As future works, we plan to develop a tool prototype to support our assume admissible synthesis rule.

References

1 Brandenburger Adam, Friedenberg Amanda, H Jerome, et al. Admissibility in games. *Econometrica*, 2008.

2 Dietmar Berwanger. Admissibility in infinite games. In *Proc. of STACS'07*, volume 4393 of *LNCS*, pages 188–199. Springer, February 2007.

3 Roderick Bloem, Rüdiger Ehlers, Swen Jacobs, and Robert Könighofer. How to handle assumptions in synthesis. In *SYNT'14*, volume 157 of *EPTCS*, pages 34–50, 2014.

4 Romain Brenguier, Jean-François Raskin, and Mathieu Sassolas. The complexity of admissibility in omega-regular games. In *CSL-LICS'14, 2014*. ACM, 2014.

5 Krishnendu Chatterjee, Laurent Doyen, Emmanuel Filiot, and Jean-François Raskin. Doomsday equilibria for omega-regular games. In *VMCAI'14*, volume 8318, pages 78–97. Springer, 2014.

6 Krishnendu Chatterjee and Thomas A Henzinger. Assume-guarantee synthesis. In *TACAS'07*, volume 4424 of *LNCS*. Springer, 2007.

7 Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. Environment assumptions for synthesis. In *CONCUR 2008*, volume 5201 of *LNCS*, pages 147–161. Springer, 2008.

8 Krishnendu Chatterjee, Thomas A Henzinger, and Marcin Jurdziński. Games with secure equilibria. *Theoretical Computer Science*, 365(1):67–82, 2006.

9 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy logic. *Inf. Comput.*, 208(6):677–693, 2010.

10 Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer, 1981.

11 Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL'77*. ACM, 1977.

12 Werner Damm and Bernd Finkbeiner. Automatic compositional synthesis of distributed systems. In *FM 2014*, volume 8442 of *LNCS*, pages 179–193. Springer, 2014.

13 Luca de Alfaro, Patrice Godefroid, and Radha Jagadeesan. Three-valued abstractions of games: Uncertainty, but with precision. In *LICS'04*. IEEE, 2004.

14 E Allen Emerson. Temporal and modal logic. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, 995:1072, 1990.

15 Marco Faella. Admissible strategies in infinite games over graphs. In *MFCS 2009*, volume 5734 of *Lecture Notes in Computer Science*, pages 307–318. Springer, 2009.

16 Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In *TACAS'10*, volume 6015 of *LNCS*, pages 190–204. Springer, 2010.

17 Drew Fudenberg and Jean Tirole. *Game Theory*. MIT Press, Cambridge, MA, 1991. Translated into Chinesse by Renin University Press, Bejing: China.

18 Thomas A. Henzinger, Rupak Majumdar, Freddy Y. C. Mang, and Jean-François Raskin. Abstract interpretation of game properties. In *SAS*, pages 220–239, 2000.

19 Paul Hunter. *Complexity and Infinite Games on Finite Graphs*. PhD thesis, Computer Laboratory, University of Cambridge, 2007.

20 O. Kupferman, G. Perelli, and M.Y. Vardi. Synthesis with rational environments. In *Proc. 12th European Conference on Multi-Agent Systems*, LNCS. Springer, 2014.

21 Zohar Manna and Pierre Wolper. Synthesis of communicating processes from temporal logic specifications. In *Logics of Programs*, volume 131 of *LNCS*, pages 253–281. Springer, 1981.

22 Fabio Mogavero, Aniello Murano, and Moshe Y. Vardi. Reasoning about strategies. In *FSTTCS 2010*, volume 8 of *LIPIcs*. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2010.

23 John Nash. Equilibrium points in n-person games. *Proc. NAS*, 1950.

24 Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

Reactive Synthesis Without Regret

Paul Hunter*, **Guillermo A. Pérez**†, **and Jean-François Raskin***

Département d'Informatique, Université Libre de Bruxelles (U.L.B.), Belgium
{phunter,gperezme,jraskin}@ulb.ac.be

Abstract

Two-player zero-sum games of infinite duration and their quantitative versions are used in verification to model the interaction between a controller (Eve) and its environment (Adam). The question usually addressed is that of the existence (and computability) of a strategy for Eve that can maximize her payoff against any strategy of Adam. In this work, we are interested in strategies of Eve that minimize her regret, i.e. strategies that minimize the difference between her actual payoff and the payoff she could have achieved if she had known the strategy of Adam in advance. We give algorithms to compute the strategies of Eve that ensure minimal regret against an adversary whose choice of strategy is (i) unrestricted, (ii) limited to positional strategies, or (iii) limited to word strategies, and show that the two last cases have natural modelling applications. We also show that our notion of regret minimization in which Adam is limited to word strategies generalizes the notion of good for games introduced by Henzinger and Piterman, and is related to the notion of determinization by pruning due to Aminof, Kupferman and Lampert.

1998 ACM Subject Classification F.1.1 Automata, D.2.4 Formal methods

Keywords and phrases Quantitative games, Regret, Verification, Synthesis, Game theory

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.114

1 Introduction

The model of two player games played on graphs is an adequate mathematical tool to solve important problems in computer science, and in particular the reactive system synthesis problem [20]. In that context, the game models the non-terminating interaction between the system to synthesize and its environment. Games with quantitative objectives are useful to formalize important quantitative aspects such as mean-response time or energy consumption. They have attracted large attention recently, see e.g. [7, 3]. Most of the contributions in this context are for zero-sum games: the objective of Eve (that models the system) is to maximize the value of the game while the objective of Adam (that models the environment) is to minimize this value. This is a worst-case assumption: because the cooperation of the environment cannot be assumed, we postulate that it is *antagonistic*.

In this antagonistic approach, the main solution concept is that of a *winning strategy*. Given a threshold value, a winning strategy for Eve ensures a minimal value greater than the threshold against any strategy of Adam. However, sometimes there are no winning strategies. What should the behaviour of the system be in such cases? There are several possible answers to this question. One is to consider *non-zero sum* extensions of those games: the environment (Adam) is not completely antagonistic, rather it has its own specification. In such games, a strategy for Eve must be winning only when the outcome satisfies the objectives of Adam, see e.g. [5]. Another option for Eve is to play a strategy which minimizes

* Authors supported by the ERC inVEST (279499) project.
† Author supported by F.R.S.-FNRS fellowship.

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 114–127

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

her *regret*. The regret is informally defined as the difference between what a player actually wins and what she could have won if she had known the strategy chosen by the other player. Minimization of regret is a central concept in decision theory [2]. This notion is important because it usually leads to solutions that agree with common sense.

Let us illustrate the notion of regret minimization on the example of Fig. 1. In this example, Eve owns the squares and Adam owns the circles (we do not use the letters labelling edges for the moment). The game is played for infinitely many rounds and the value of a play for Eve is the long run average of the values of edges traversed during the play (the so-called *mean-payoff*). In this game, Eve is only able to secure a mean-payoff of $\frac{1}{2}$ when Adam is fully antagonistic. Indeed, if Eve (from v_1) plays to v_2 then Adam can force a mean-payoff value of 0, and if she plays to v_3 then the mean-payoff value is at least $\frac{1}{2}$. Note also that if Adam is not fully antagonistic, then the mean-payoff could be as high as 2. Now, assume that Eve does not try to force the highest value in the worst-case but tries to minimize her regret. If she plays $v_1 \mapsto v_2$ then the regret is equal to 1. This is because Adam can play the following strategy: if Eve plays to v_2 (from v_1) then he plays $v_2 \mapsto v_1$ (giving a mean-payoff of 0), and if Eve plays to v_3 then he plays to v_5 (giving a mean-payoff of 1). If she plays $v_1 \mapsto v_3$ then her regret is $1\frac{1}{2}$ since Adam can play the symmetric strategy. It should thus be clear that the strategy of Eve which always chooses $v_1 \mapsto v_2$ is indeed minimizing her regret.

In this paper, we will study three variants of *regret minimization*, each corresponding to a different set of strategies we allow Adam to choose from. The first variant is when Adam can play any possible strategy (as in the example above), the second variant is when Adam is restricted to playing *memoryless strategies*, and the third variant is when Adam is restricted to playing *word strategies*. To illustrate the last two variants, let us consider again the example of Fig. 1. Assume now that Adam is playing memoryless strategies only. Then in this case, we claim that there is a strategy of Eve that ensures regret 0. The strategy is as follows: first play to v_2, if Adam chooses to go back to v_1, then Eve should henceforth play $v_1 \mapsto v_3$. We claim that this strategy has regret 0. Indeed, when v_2 is visited, either Adam chooses $v_2 \mapsto v_4$, and then Eve secures a mean-payoff of 2 (which is the maximal possible value), or Adam chooses $v_2 \mapsto v_1$ and then we know that $v_1 \mapsto v_2$ is not a good option for Eve as cycling between v_1 and v_2 yields a payoff of only 0. In this case, the mean-payoff is either 1, if Adam plays $v_3 \mapsto v_5$, or a payoff of $\frac{1}{2}$, if he plays $v_3 \mapsto v_1$. In all the cases, the regret is 0. Let us now turn to the restriction to word strategies for Adam. When considering this restriction, we use the letters that label the edges of the graph. A word strategy for Adam is a function $w : \mathbb{N} \to \{a, b\}$. In this setting Adam plays a sequence of letters and this sequence is independent of the current state of the game. When Adam plays word strategies, the strategy that minimizes regret for Eve is to always play $v_1 \mapsto v_2$. Indeed, for any word in which the letter a appears, the mean-payoff is equal to 2, and the regret is 0, and for any word in which the letter a does not appear, the mean-payoff is 0 while it would have been equal to $\frac{1}{2}$ when playing $v_1 \mapsto v_3$. So the regret of this strategy is $\frac{1}{2}$ and it is the minimal regret that Eve can secure. Note that the three different strategies give three different values in our example. This is in contrast with the worst-case analysis of the same problem (memoryless strategies suffice for both players).

We claim that at least the two last variants are useful for modelling purposes. For example, the memoryless restriction is useful when designing a system that needs to perform well in an environment which is only partially known. In practical situations, a controller may discover the environment with which it is interacting at run time. Such a situation can be modelled by an arena in which choices in nodes of the environment model an entire family of environments and each memoryless strategy models a specific environment of the family.

Table 1 Complexity of deciding the regret threshold problem.

Payoff type	Any strategy	Memoryless strategies	Word strategies
Sup, Inf, and LimSup	PTIME-c (Thm. 1)	coNP-h (Lem. 8) and in PSPACE (Lem. 6)	EXPTIME-c (Thm. 10)
LimInf	PTIME-c (Thm. 1)	PSPACE-c (Thm. 5)	EXPTIME-c (Thm. 10)
$\underline{\text{MP}}$, $\overline{\text{MP}}$	MP equivalent (Thm. 1)	PSPACE-c (Thm. 5)	Undecidable (Lem. 13)

In such cases, if we want to design a controller that performs reasonably well against all the possible environments, we can consider a controller that minimizes regret: the strategy of the controller will be as close as possible to an optimal strategy if we had known the environment beforehand. This is, for example, the modelling choice done in the famous Canadian traveller's problem [19]: a driver is attempting to reach a specific location while ensuring the traversed distance is *not too far* from the shortest feasible path. The partial knowledge is due to some roads being closed because of snow. The Canadian traveller, when planning his itinerary, is in fact searching for a strategy to minimize his regret for the shortest path measure against a memoryless adversary who determines the roads that are closed. Similar situations naturally arise when synthesizing controllers for *robot motion planning* [21]. We now illustrate the usefulness of the variant in which Adam is restricted to play word strategies. Assume that we need to design a system embedded into an environment that produces disturbances: if the sequence of disturbances produced by the environment is independent of the behavior of the system, then it is natural to model this sequence not as a function of the state of the system but as a temporal sequence of events, i.e. a *word* on the alphabet of the disturbances. Clearly, if the sequences are not the result of an antagonistic process, then minimizing the regret against all disturbance sequences is an adequate solution concept to obtain a reasonable system and may be preferable to a system obtained from a strategy that is optimal under the antagonistic hypothesis.

Contributions. In this paper, we provide algorithms to solve the *regret threshold problem* (strict and non-strict) in the three variants explained above, i.e. given a game and a threshold, does there exist a strategy for Eve with a regret that is (strictly) less than the threshold against all (resp. all memoryless, resp. all word) strategies for Adam. Almost all of our algorithms are reductions to well-known games, therefore synthesizing the corresponding controller amounts to computing the strategy of Eve in the resulting game. We study this problem for six common quantitative measures: Inf, Sup, LimInf, LimSup, $\underline{\text{MP}}$, $\overline{\text{MP}}$. For all measures, but MP, the strict and non-strict threshold problems are equivalent. We state our results for both cases for consistency. In almost all the cases, we provide matching lower bounds showing the worst-case optimality of our algorithms. Our results are summarized in Table 1.

For the variant in which Adam plays word strategies only, we show that we can recover decidability of mean-payoff objectives when the memory of Eve is fixed in advance: in this case, the problem is NP-complete (Theorems 14 and 15).

Related works. The notion of regret minimization is a central one in game theory, see e.g. [22] and references therein. Also, *iterated* regret minimization has been recently proposed by Halpern et al. as a concept for *non-zero* sum games [13]. There, it is applied to matrix games and not to game graphs. In a previous contribution, we have applied the iterated regret

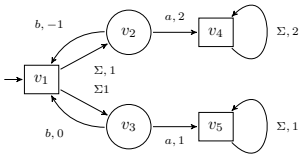


Figure 1 Example weighted arena G_0.

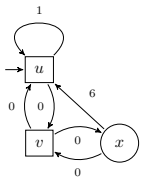


Figure 2 Example weighted arena G_1.

minimization concept to non-zero sum games played on weighted graphs for the shortest path problem [12]. Restrictions on how Adam is allowed to play were not considered there. As we do not consider an explicit objective for Adam, we do not consider iteration of the regret minimization here.

The disturbance-handling embedded system example was first given in [8]. In that work, the authors introduce *remorsefree strategies*, which correspond to strategies which minimize regret in games with ω-regular objectives. They do not establish lower bounds on the complexity of realizability or synthesis of remorsefree strategies and they focus on word strategies of Adam only.

In [14], Henzinger and Piterman introduce the notion of *good for games automata*. A non-deterministic automaton is good for solving games if it fairly simulates the equivalent deterministic automaton. We show that our notion of regret minimization for word strategies extends this notion to the quantitative setting (Proposition 17). Our definitions give rise to a natural notion of approximate determinisation for weighted automata on infinite words.

In [1], Aminof et al. introduce the notion of *approximate determinisation by pruning* for weighted sum automata over finite words. For $\alpha \in (0, 1]$, a weighted sum automaton is *α-determinisable by pruning* if there exists a finite state strategy to resolve non-determinism and that constructs a run whose value is at least α times the value of the maximal run of the given word. So, they consider a notion of approximation which is a *ratio*. We will show that our concept of regret, when Adam plays word strategies only, defines instead a notion of approximation with respect to the *difference* metric for weighted automata (Proposition 16). There are other differences with their work. First, we consider infinite words while they consider finite words. Second, we study a general notion of regret minimization problem in which Eve can use any strategy while they restrict their study to fixed memory strategies only and leave the problem open when the memory is not fixed a priori.

Finally, the main difference between these related works and this paper is that we study the Inf, Sup, LimInf, LimSup, $\underline{\mathsf{MP}}$, $\overline{\mathsf{MP}}$ measures while they consider the total sum measure or qualitative objectives.

2 Preliminaries

A *weighted arena* is a tuple $G = (V, V_\exists, E, w, v_I)$ where (V, E, w) is a finite edge-weighted graph[1] with integer weights, $V_\exists \subseteq V$, and $v_I \in V$ is the initial vertex. In the sequel we depict vertices owned by Eve (i.e. $V_\exists$) with squares and vertices owned by Adam (i.e. $V \setminus V_\exists$) with circles. We denote the maximum absolute value of a weight in a weighted arena by W.

A *play* in a weighted arena is an infinite sequence of vertices $\pi = v_0 v_1 \dots$ where $v_0 = v_I$ and $(v_i, v_{i+1}) \in E$ for all i. We extend the weight function to partial plays by setting $w(\langle v_i \rangle_{i=k}^{l}) = \sum_{i=k}^{l-1} w(v_i, v_{i+1})$.

[1] W.l.o.g. G is assumed to be total: for each $v \in V$, there exists $v' \in V$ such that $(v, v') \in E$.

A *strategy for* Eve (Adam) is a function σ that maps partial plays ending with a vertex v in $V_\exists$ ($V \setminus V_\exists$) to a successor of v. A strategy has memory m if it can be realized as the output of a finite state machine with m states (see e.g. [15] for a formal definition). A *memoryless (or positional) strategy* is a strategy with memory 1, that is, a function that only depends on the last element of the given partial play. A play $\pi = v_0 v_1 \dots$ is *consistent with a strategy* σ for Eve (Adam) if whenever $v_i \in V_\exists$ ($v_i \in V \setminus V_\exists$), $\sigma(\langle v_j \rangle_{j \leq i}) = v_{i+1}$. We denote by $\mathfrak{S}_\exists(G)$ ($\mathfrak{S}_\forall(G)$) the set of all strategies for Eve (Adam) and by $\Sigma^m_\exists(G)$ ($\Sigma^m_\forall(G)$) the set of all strategies for Eve (Adam) in G that require memory of size at most m, in particular $\Sigma^1_\exists(G)$ ($\Sigma^1_\forall(G)$) is the set of all memoryless strategies of Eve (Adam) in G. We omit G if the context is clear.

Payoff functions. A play in a weighted arena defines an infinite sequence of weights. We define below several classical *payoff functions* that map such sequences to real numbers.[2] Formally, for a play $\pi = v_0 v_1 \dots$ we define:

- the Inf (Sup) payoff, is the minimum (maximum) weight seen along a play: $\mathsf{Inf}(\pi) = \inf\{w(v_i, v_{i+1}) : i \geq 0\}$ and $\mathsf{Sup}(\pi) = \sup\{w(v_i, v_{i+1}) : i \geq 0\}$;
- the LimInf (LimSup) payoff, is the minimum (maximum) weight seen infinitely often: $\mathsf{LimInf}(\pi) = \liminf_{i \to \infty} w(v_i, v_{i+1})$ and $\mathsf{LimSup}(\pi) = \limsup_{i \to \infty} w(v_i, v_{i+1})$;
- the *mean-payoff* value of a play, i.e. the limiting average weight, defined using lim inf or lim sup since the running averages might not converge: $\underline{\mathsf{MP}}(\pi) = \liminf_{k \to \infty} \frac{1}{k} w(\langle v_i \rangle_{i<k})$ and $\overline{\mathsf{MP}}(\pi) = \limsup_{k \to \infty} \frac{1}{k} w(\langle v_i \rangle_{i<k})$.

A payoff function $\mathbf{Val}$ is *prefix-independent* if for all plays $\pi = v_0 v_1 \dots$, for all $j \geq 0$, $\mathbf{Val}(\pi) = \mathbf{Val}(\langle v_j \rangle_{j \geq i})$. It is well-known that LimInf, LimSup, $\underline{\mathsf{MP}}$, and $\overline{\mathsf{MP}}$ are prefix-independent. Often, the arguments that we develop work uniformly for these four measures because of their prefix-independent property. Inf and Sup are not prefix-independent but often in the sequel we apply a simple transformation to the game and encode Inf into a LimInf objective, and Sup into a LimSup objective. The transformation consists of encoding in the vertices of the arena the minimal (maximal) weight that has been witnessed by a play, and label the edges of the new graph with this same recorded weight. When this simple transformation does not suffice, we mention it explicitly.

Regret. Consider a fixed weighted arena G, and payoff function $\mathbf{Val}$. Given strategies σ, τ, for Eve and Adam respectively, and $v \in V$, we denote by $\pi^v_{\sigma\tau}$ the unique play starting from v that is consistent with σ and τ and denote its value by: $\mathbf{Val}^v_G(\sigma, \tau) := \mathbf{Val}(\pi^v_{\sigma\tau})$. We omit G if it is clear from the context. If v is omitted, it is assumed to be v_I.

Let $\Sigma_\exists \subseteq \mathfrak{S}_\exists$ and $\Sigma_\forall \subseteq \mathfrak{S}_\forall$ be sets of strategies for Eve and Adam respectively. Given $\sigma \in \Sigma_\exists$ we define the *regret of* σ *in* G *w.r.t.* $\Sigma_\exists$ *and* $\Sigma_\forall$ as:

$$\mathbf{reg}^\sigma_{\Sigma_\exists, \Sigma_\forall}(G) := \sup_{\tau \in \Sigma_\forall} (\sup_{\sigma' \in \Sigma_\exists} \mathbf{Val}(\sigma', \tau) - \mathbf{Val}(\sigma, \tau)).$$

We define the *regret of* G *w.r.t.* $\Sigma_\exists$ *and* $\Sigma_\forall$ as:

$$\mathbf{Reg}_{\Sigma_\exists, \Sigma_\forall}(G) := \inf_{\sigma \in \Sigma_\exists} \mathbf{reg}^\sigma_{\Sigma_\exists, \Sigma_\forall}(G).$$

When $\Sigma_\exists$ or $\Sigma_\forall$ are omitted from $\mathbf{reg}(\cdot)$ and $\mathbf{Reg}(\cdot)$ they are assumed to be the set of all strategies for Eve and Adam.

[2] The values of all functions are not infinite, and therefore in $\mathbb{R}$ since we deal with finite graphs only.

We will make use of two other values associated with the vertices of an arena: the *antagonistic* and *cooperative* values, defined for plays from a vertex $v \in V$ as

$$\mathbf{aVal}^v(G) := \sup_{\sigma \in \mathfrak{S}_\exists} \inf_{\tau \in \mathfrak{S}_\forall} \mathbf{Val}^v(\sigma, \tau) \qquad \mathbf{cVal}^v(G) := \sup_{\sigma \in \mathfrak{S}_\exists} \sup_{\tau \in \mathfrak{S}_\forall} \mathbf{Val}^v(\sigma, \tau).$$

When clear from context G will be omitted, and if v is omitted it is assumed to be v_I.

▶ Remark. It is well-known that **cVal** and **aVal** can be computed in polynomial time, w.r.t. the underlying graph of the given arena, for all payoff functions but MP [4, 6]. For MP, **cVal** is known to be computable in polynomial time for **aVal** it can be done in UP ∩ coUP [17] and in *pseudo-polynomial time* [23, 3].

3 Variant I: Adam plays any strategy

For this variant, we establish that for all the payoff functions that we consider, the problem of computing the antagonistic value and the problem of computing the regret value are *inter-reducible* in polynomial time. As a direct consequence, we obtain the following theorem:

▶ **Theorem 1.** *Deciding if the regret value is less than a given threshold (strictly or non-strictly) is* PTIME*-complete (under log-space reductions) for* Inf*,* Sup*,* LimInf*, and* LimSup*, and equivalent to mean-payoff games (under polynomial-time reductions) for* $\underline{\mathsf{MP}}$ *and* $\overline{\mathsf{MP}}$*.*

Upper bounds. We now describe an algorithm to compute regret for all payoff functions.

▶ **Lemma 2.** *For payoff functions* Inf*,* Sup*,* LimInf*,* LimSup*,* $\underline{\mathsf{MP}}$*, and* $\overline{\mathsf{MP}}$ *computing the regret of a game is at most as hard as computing the antagonistic value of a (polynomial-size) game with the same payoff function.*

Sketch. We describe how the algorithm works for the $\underline{\mathsf{MP}}$ function, the algorithm is similar for all other payoff functions and details are given in the technical report [16]. Let us fix a weighted arena G. We define a new weight function w' as follows. For any edge $e = (u, v)$ let $w'(e) = -\infty$ if $u \in V \setminus V_\exists$, and if $u \in V_\exists$ then $w'(e) = \max\{\mathbf{cVal}^{v'} : (u, v') \in E \setminus \{e\}\}$. Intuitively, w' represents the best value obtainable for a strategy of Eve that differs at the given edge. It is not difficult to see that in order to minimize regret, Eve is trying to simultaneously maximize the value given by the original weight function w, and minimize the maximum w'-weighted edge seen. For $b \in \mathrm{Range}(w')$ we define G^b to be the graph obtained by restricting G to edges e with $w'(e) \leq b$.

Next, we will construct a new weighted arena $\hat{G}$ such that the regret of G is a function of the *antagonistic* value of $\hat{G}$. Figure 3 depicts the general form of the arena we construct. We have three vertices $v_0 \in \hat{V} \setminus \hat{V}_\exists$ and $v_1, v_\perp \in \hat{V}_\exists$ and a "copy" of G as G^b for each $b \in \mathrm{Range}(w') \setminus \{-\infty\}$. We have a self-loop of weight 0 on v_0 which is the initial vertex of $\hat{G}$, a self-loop of weight $-2W - 1$ on $v_\perp$, and weight 0 edges from v_0 to v_1 and from v_1 to the initial vertices of G^b for all b. Recall that G^b might not be total. To fix this we add, for all vertices without a successor, a weight 0 edge to $v_\perp$. The remainder of the weight function $\hat{w}$, is defined for each edge e^b in G^b as $\hat{w}(e^b) = w(e) - b$.

Intuitively, in $\hat{G}$ Adam first decides whether he can ensure a non-zero regret. If this is the case, then he moves to v_1. Next, Eve chooses a maximal value she will allow for strategies which differ from the one she will play (this is the choice of b). The play then moves to the corresponding copy of G, i.e. G^b. She can now play to maximize her mean-payoff value. However, if her choice of b was not correct then the play will end in $v_\perp$. We claim this construction ensures that $\mathbf{Reg}(G) = -\mathbf{aVal}(\hat{G})$. ◀

Lower bounds. For all the payoff functions, from G we can construct in logarithmic space G' such that the antagonistic value of G is equal to the regret value of G', and so we have:

▶ **Lemma 3.** *For payoff functions* Inf, Sup, LimInf, LimSup, $\underline{\mathsf{MP}}$, *and* $\overline{\mathsf{MP}}$ *computing the regret of a game is at least as hard as computing the antagonistic value of a (polynomial-size) game with the same payoff function.*

Sketch. Suppose G is a weighted arena with initial vertex v_I. Consider the weighted arena G' obtained by adding to G the gadget of Figure 5. The initial vertex of G' is set to be v'_I. We claim that the right choice of values for the parameters L, M_1, M_2, N_1, N_2 makes it so that the antagonistic value of G is a function of the regret of the game G'.

For concreteness, let us consider the payoff function $\underline{\mathsf{MP}}$, and let $L = M_1 = M_2 = 0$, $N_1 = W + 1$, and $N_2 = -3W - 2$. At v'_I, Eve has a choice: she can choose to remain in the gadget or she can move to the original game G. If she chooses to remain in the gadget, her payoff will be $-3W - 2$, meanwhile Adam could choose a strategy that would have achieved a payoff of $\mathbf{cVal}(G)$ if she had chosen to play to G. Hence her regret in this case is $\mathbf{cVal}(G) + 3W + 2 \geq 2W + 2$. Otherwise, if she chooses to play to G, she can achieve a payoff of at most $\mathbf{aVal}(G)$ if Adam is adversarial. As $\mathbf{cVal}(G) \leq W$ and W is the maximum possible payoff achievable in G, the strategy of Adam which now maximizes Eve's regret is the one which remains in the gadget – giving a payoff of $W + 1$. Her regret in this case is $K + 1 - \mathbf{aVal}(G) \leq 2W + 1$. Therefore, to minimize her regret she will play this strategy. It follows that $\mathbf{Reg}(G') = W + 1 - \mathbf{aVal}(G)$, and thus the adversarial value of G can be deduced from the regret value of G'. ◀

Memory requirements for Eve and Adam. It follows from the reductions underlying the proof of Lemma 2 that Eve only requires positional strategies to minimize regret when there is no restriction on Adam's strategies. On the other hand, Adam's strategy for maximizing regret consists of a combination of three positional strategies: first he moves to the optimal vertex for deviating, then he plays his optimal (positional) strategy in the antagonistic game. His strategy for the alternative scenario, assuming Eve had deviated, is his optimal strategy in the co-operative game which is also positional. This combined strategy is clearly realizable as a strategy with three memory states, giving us:

▶ **Corollary 4.** *For payoff functions* LimInf, LimSup, $\underline{\mathsf{MP}}$ *and* $\overline{\mathsf{MP}}$*:* $\mathbf{Reg}(G) = \mathbf{Reg}_{\Sigma^1_\exists, \Sigma^3_\forall}(G)$.

The algorithm we give relies on the prefix-independence of the payoff function. As the transformation from Inf and Sup to equivalent prefix-independent ones is polynomial it follows that polynomial memory (w.r.t. the size of the underlying graph of the arena) suffices for both players.

4 Variant II: Adam plays memoryless strategies

For this variant, we provide a polynomial space algorithm to solve the problem for all the payoff functions, we then provide lower bounds.

▶ **Theorem 5.** *Deciding if the regret value is less than a given threshold (strictly or non-strictly) playing against memoryless strategies of Adam is* PSPACE*-complete for* LimInf, $\underline{\mathsf{MP}}$ *and* $\overline{\mathsf{MP}}$*; in* PSPACE *and* coNP*-hard for* Inf, Sup *and* LimSup.

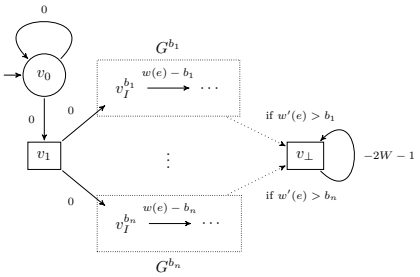


Figure 3 Weighted arena $\hat{G}$, constructed from G. Dotted lines represent several edges added when the condition labelling it is met.

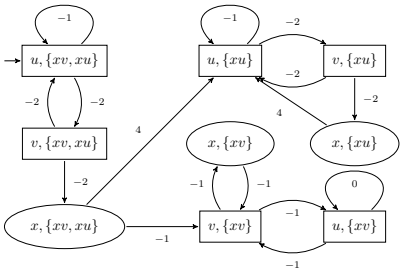


Figure 4 Weighted arena $\hat{G}_1$, constructed from G_1. In the edge set component only edges leaving Adam nodes are depicted.

Upper bounds. Let us now show how to compute regret against positional adversaries.

▶ **Lemma 6.** *For payoff functions* Inf, Sup, LimInf, LimSup, $\underline{\text{MP}}$ *and* $\overline{\text{MP}}$*, the regret of a game played against a positional adversary can be computed in polynomial space.*

Sketch. Once again, we describe how the algorithm works for the $\underline{\text{MP}}$. Given a weighted arena G, we construct a new weighted arena $\hat{G}$ such that we have that $-\mathbf{aVal}(\hat{G})$ is equivalent to the regret of G. The argument works for all prefix independent payoff functions and the details are given in the technical report [16] for Inf, Sup.

The vertices of $\hat{G}$ encode the choices made by Adam. For a subset of edges $D \subseteq E$, let $G{\restriction}D$ denote the weighted arena $(V, V_\exists, E \cap D, w, v_I)$. The new weighted arena $\hat{G}$ is the tuple $(\hat{V}, \hat{V_\exists}, \hat{E}, \hat{w}, \hat{v_I})$ where

(i) $\hat{V} = V \times \mathcal{P}(E)$;
(ii) $\hat{V_\exists} = \{(v, e) \in \hat{V} : v \in V_\exists\}$;
(iii) $\hat{v_I} = (v_I, E)$;
(iv) $\hat{E}$ contains the edge $\big((u, C), (v, D)\big)$ if and only if $(u, v) \in C$ and, either $u \in V_\exists$ and $D = C$, or $u \in V \setminus V_\exists$ and $D = C \setminus \{(u, x) \in E : x \neq v\}$;
(v) $\hat{w}\big((u, C), (v, D)\big) = w(u, v) - \mathbf{cVal}(G{\restriction}D)$.

The application of this transformation for the graph of Fig. 2 is given in Fig. 4.

Finally, we recall that the value of a mean-payoff game is equivalent to the value of its *cycle forming game* [10]. This finite cycle forming game is identical to the mean-payoff game except that it is stopped as soon as a cycle is formed and the value of the game is given by the mean-payoff value of the cycle. It follows that one can use an Alternating Turing Machine to compute the value of $\hat{G}$ in time bounded by the length of the longest simple path in $\hat{G}$: $|V|(|E| + 1)$. Since APTIME = PSPACE, the result follows. ◀

Lower bounds. We give a reduction from the QSAT Problem to the problem of determining whether, given $r \in \mathbb{Q}$, $\mathbf{Reg}_{\mathfrak{S}_\exists, \Sigma^1_\forall}(G) \lhd r$ for the payoff functions LimInf, $\underline{\text{MP}}$, and $\overline{\text{MP}}$ (for $\lhd \in \{<, \leq\}$). Then we provide a reduction from the complement of the 2-disjoint-paths Problem for LimSup, Sup, and Inf.

▶ **Lemma 7.** *For $r \in \mathbb{Q}$, weighted arena G and payoff function* LimInf, $\underline{\text{MP}}$, *or* $\overline{\text{MP}}$*, determining whether $\mathbf{Reg}_{\mathfrak{S}_\exists, \Sigma^1_\forall}(G) \lhd r$, for $\lhd \in \{<, \leq\}$, is* PSPACE*-hard.*

Sketch. The crux of the reduction from QSAT is a gadget for each clause of the QSAT formula. Visiting this gadget allows Eve to gain information about the highest payoff obtainable in the gadget, each entry point corresponds to a literal from the clause, and the

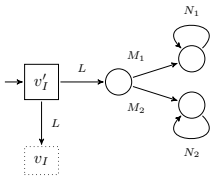


Figure 5 Gadget to reduce a game to its regret game.

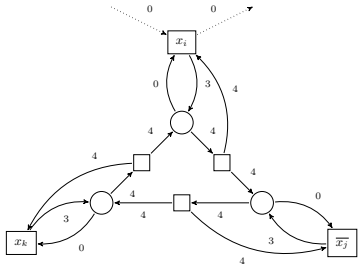


Figure 6 Clause gadget for the QBF reduction for clause $x_i \vee \neg x_j \vee x_k$.

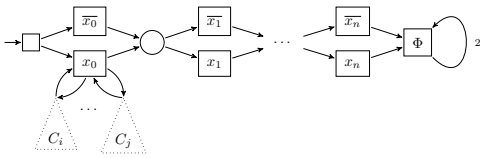


Figure 7 Depiction of the reduction from QBF.

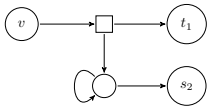


Figure 8 Regret gadget for 2-disjoint-paths reduction.

literal is visited when it is made true by the valuation of variables chosen by Eve and Adam in the reduction described below. Figure 6 depicts an instance of the gadget for a particular clause. Let us focus on the mean-payoff function. Note that staying in the inner 6-vertex triangle would yield a mean-payoff value of 4. However, in order to do so, Adam needs to cooperate with Eve at all three corner vertices. Also note that if he does cooperate in at least one of these vertices then Eve can secure a payoff value of at least $\frac{11}{3}$.

The complete reduction for MP consists in describing how to construct, from an input QBF Φ, a weighted arena G of linear size w.r.t. Φ. In G, Eve can ensure $\mathbf{Reg}_{\mathfrak{S}_\exists, \Sigma^1_\forall}(G) < 2$ if and only if the QBF true. We assume Φ is in 3-CNF and w.l.o.g. we assume that each clause contains at least one existentially quantified variable.

It is common to consider a QBF as a game between an existential and a universal player. The game we construct mimics the choices of the existential and universal player and makes sure that the regret of the game is smaller than 2 if and only if Φ is true. Figure 7 depicts the general structure of the game. Eve and Adam choose valuations for the variables. Depending on these choices, clause gadgets and literals that are made true by the constructed valuation can be visited by Eve.

Assume the QBF is true, then Eve has a value-choosing strategy s.t. for any strategy of Adam, all clauses have at least one literal which holds. Hence, Eve can ensure to visit every clause gadget. If Adam helps Eve in any of these gadgets then she can ensure a payoff of $\frac{11}{3}$ as explained above. Otherwise, she arrives at Φ, gets mean-payoff 2 and, since Adam did not help her in any of the clause gadgets, she is sure that no alternative strategy can achieve a payoff value of 4. Hence, the regret of the game is less than 2.

Conversely, if the QBF is false then Adam can make sure Eve does not visit at least one clause gadget that corresponds to a clause that is false in the constructed valuation. Additionally, in every clause gadget she does visit, he does not help her. She can now ensure a mean-payoff value of 2 by going to Φ but an alternative strategy of Eve can force a visit to the gadget not visited (as each clause contains at least one existentially quantified variable) and there Adam can now fully cooperate and ensures a payoff of 4 to Eve. ◀

▶ **Lemma 8.** *For $r \in \mathbb{Q}$, weighted arena G and payoff function* Inf, Sup, *or* LimSup, *determining whether* $\mathbf{Reg}_{\mathfrak{S}_\exists,\Sigma^1_\forall}(G) \lhd r$, *for* $\lhd \in \{<, \leq\}$, *is* coNP*-hard.*

Proof. We provide a reduction from the complement of the 2-DISJOINT-PATHS PROBLEM on directed graphs [11]. As the problem is known to be NP-complete, the result follows. In other words, we sketch how to translate a given instance of the 2-DISJOINT-PATHS PROBLEM into a weighted arena in which Eve can ensure regret value strictly less than 1 if and only if the answer to the 2-DISJOINT-PATHS PROBLEM is negative.

Consider a directed graph G and distinct vertex pairs (s_1, t_1) and (s_2, t_2). W.l.o.g. we assume that for all $i \in \{1, 2\}$:

(i) t_i is reachable from s_i, and
(ii) t_i is a sink (i.e. has no outgoing edges)

in G. We now describe the changes we apply to G in order to get the underlying graph structure of the weighted arena and then comment on the weight function. Let all vertices from G be Adam vertices and s_1 be the initial vertex. We replace all edges (v, t_1) incident on t_1 by a copy of the gadget shown in Figure 8. Next, we add self-loops on t_1 and t_2 with weights 1 and 2, respectively. Finally, the weights of all remaining edges are 0.

We claim that, in this weighted arena, Eve can ensure regret strictly less than 1 – for payoff functions Sup and LimSup – if and only if in G the vertex pairs (s_1, t_1) and (s_2, t_2) cannot be joined by vertex-disjoint paths. Indeed, we claim that the strategy that minimizes the regret of Eve is the strategy that, in states where she has a choice, tells her to go to t_1.

First, let us prove that this strategy has regret strictly less than 1 if and only if no two disjoint paths in the graph exist between the pairs of states (s_1, t_1) and (s_2, t_2). Assume the latter is the case. Then if Adam chooses to always avoid t_1, then clearly the regret is 0. If t_1 is eventually reached, then the choice of Eve secures a value of 1 (for all payoff functions). Note that if she had chosen to go towards s_2 instead, as there are no two disjoint paths, we know that either the path constructed from s_2 by Adam never reaches t_2, and then the value of the path is 0 – and the regret is 0 for Eve– or the path constructed from s_2 reaches t_1 again – and, again, the regret is 0 for Eve. Now assume that two disjoint paths between the source-target pairs exist. If Eve changed her strategy to go towards s_2 (instead of choosing t_1) then Adam has a strategy to reach t_2 and achieve a payoff of 2. Thus, her regret would be equal to 1.

Second, we claim that any other strategy of Eve has a regret greater than or equal to 1. Indeed, if Eve decides to go towards s_2 (instead of choosing to go to t_1) then Adam can choose to loop on the state before s_2 and the payoff in this case is 0. Hence, the regret of Eve is at least 1.

Note that minimal changes are required for the same construction to imply the result for Inf. Further, the weight function and threshold r can be accommodated so that Eve wins for the non-strict regret threshold. Hence, the general result follows. ◀

Memory requirements for Eve. It follows from our algorithms for computing regret in this variant that Eve only requires strategies with exponential memory. Examples where exponential memory is necessary can be easily constructed.

▶ **Corollary 9.** *For all payoff functions* Sup, Inf, LimSup, LimInf, $\underline{\text{MP}}$ *and* $\overline{\text{MP}}$, *for all game graphs G, there exists m which is $2^{\mathcal{O}(|G|)}$ such that:* $\mathbf{Reg}_{\mathfrak{S}_\exists,\Sigma^1_\forall}(G) = \mathbf{Reg}_{\Sigma^m_\exists,\Sigma^1_\forall}(G)$.

5 Variant III: Adam plays word strategies

For this variant, we provide tight upper and lower bounds for all the payoff functions: the regret threshold problem is EXPTIME-complete for Sup, Inf, LimSup, and LimInf, and undecidable for $\underline{\text{MP}}$ and $\overline{\text{MP}}$. For the later case, the decidability can be recovered when we fix a priori the size of the memory that Eve can use to play, the decision problem is then NP-complete. Finally, we show that our notion of regret minimization for word strategies generalizes the notion of *good for games* introduced by Henzinger and Piterman in [14], and we also formalize the relation that exists with the notion of determinisation by pruning for weighted automata introduced by Aminof et al. in [1].

Additional definitions. We say that a strategy of Adam is a *word strategy* if his strategy can be expressed as a function $\tau : \mathbb{N} \to [\max\{deg^+(v) : v \in V\}]$, where $[n] = \{i : 1 \leq i \leq n\}$. Intuitively, we consider an order on the successors of each Adam vertex. On every turn, the strategy τ of Adam will tell him to move to the i-th successor of the vertex according to the fixed order. We denote by $\mathfrak{W}_\forall$ the set of all such strategies for Adam. When considering word strategies, it is more natural to see the arena as a (weighted) automaton.

A *weighted automaton* is a tuple $\Gamma = (Q, q_I, A, \Delta, w)$ where A is a finite alphabet, Q is a finite set of states, q_I is the initial state, $\Delta \subseteq Q \times A \times Q$ is the transition relation, $w : \Delta \to \mathbb{Z}$ assigns weights to transitions. A *run* of Γ on a word $a_0 a_1 \ldots \in A^\omega$ is a sequence $\rho = q_0 a_0 q_1 a_1 \ldots \in (Q \times A)^\omega$ such that $(q_i, a_i, q_{i+1}) \in \Delta$, for all $i \geq 0$, and has *value* $\mathbf{Val}(\rho)$ determined by the sequence of weights of the transitions of the run and the payoff function. The value Γ assigns to a word is the supremum of the values of all its runs on the word. We say the automaton is deterministic if Δ is functional.

A game in which Adam plays word strategies can be reformulated as a game played on a weighted automaton $\Gamma = (Q, q_I, A, \Delta, w)$ and strategies of Adam – of the form $\tau : \mathbb{N} \to A$ – determine a sequence of input symbols to which Eve has to react by choosing Δ-successor states starting from q_I. In this setting a strategy of Eve which minimizes regret defines a run by resolving the non-determinism of Δ in Γ, and ensures the difference of value given by the constructed run is minimal w.r.t. the value of the best run on the word spelled out by Adam. The following result summarizes the results of this section:

▶ **Theorem 10.** *Deciding if the regret value is less than a given threshold (strictly or non-strictly) playing against word strategies of Adam is* EXPTIME*-complete for* Inf*,* Sup*,* LimInf*, and* LimSup*; it is undecidable for* $\underline{\text{MP}}$ *and* $\overline{\text{MP}}$*.*

Upper bounds. There is an EXPTIME algorithm for solving the regret threshold problem for Inf, Sup, LimInf, and LimSup. This algorithm is obtained by a reduction to parity games.

▶ **Lemma 11.** *For $r \in \mathbb{Q}$, weighted automaton Γ and payoff function* Inf*,* Sup*,* LimInf*, or* LimSup*, determining whether $\mathbf{Reg}_{\mathfrak{S}_\exists, \mathfrak{W}_\forall}(\Gamma) \lhd r$, for $\lhd \in \{<, \leq\}$, can be done in exponential time.*

Sketch. We focus, for this sketch, on the LimInf payoff function. Our decision algorithm consists in first building a deterministic automaton for $\Gamma = (Q_1, q_I, A, \Delta_1, w_1)$ using the construction provided in [6]. We denote by $D_\Gamma = (Q_2, s_I, A, \Delta_2, w_2)$ this deterministic automaton and we know that it is at most exponentially larger than Γ. Then we shift the weights of the automaton D_Γ by $-r$, and we denote it by D_Γ^{-r}. Finally, we need to decide if Eve is able to simulate D_Γ^{-r} on Γ in the sense of [6]: she must be able to resolve non-determinism in Γ on letters given by Adam in a way that the run constructed in Γ has a

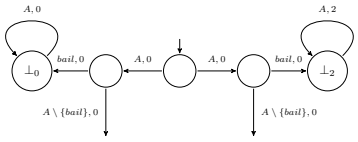


Figure 9 Initial gadget used in reduction from countdown games.

value greater than or equal to the value of the unique run of D_Γ^{-r} on the word constructed by Adam. This simulation problem can be reduced to determining the winner in a parity game. In our case, the parity game that we need is linear in the size of the product between D_Γ^{-r} and Γ, and so exponential in the size of Γ, but uses a polynomial number of priorities. Solving this parity game can be done in exponential time in the size of Γ, giving us the required upper bound for the regret threshold problem. Details for LimInf and the other measures are given in the technical report [16]. ◀

Lower bounds. We first establish EXPTIME-hardness for the payoff functions Inf, Sup, LimInf, and LimSup by giving a reduction from countdown games [18]. That is, we show that given a countdown game, we can construct a game where Eve ensures regret less than 2 if and only if Counter wins in the original countdown game.

▶ **Lemma 12.** *For $r \in \mathbb{Q}$, weighted automaton Γ and payoff function* Inf, Sup, LimInf, *or* LimSup, *determining whether $\mathbf{Reg}_{\mathfrak{S}_\exists,\mathfrak{W}_\forall}(\Gamma) \lhd r$, for $\lhd \in \{<, \leq\}$, is* EXPTIME*-hard.*

To show undecidability of the problem for the mean-payoff function we give a reduction from the threshold problem in *mean-payoff games with partial-observation*. This problem was shown to be undecidable in [9, 15].

▶ **Lemma 13.** *For $r \in \mathbb{Q}$, weighted automaton Γ and payoff function* $\underline{\mathsf{MP}}$ *or* $\overline{\mathsf{MP}}$*, determining whether $\mathbf{Reg}_{\mathfrak{S}_\exists,\mathfrak{W}_\forall}(\Gamma) \lhd r$, for $\lhd \in \{<, \leq\}$, is undecidable even if Eve is only allowed to play finite memory strategies.*

Fixed memory for Eve. Since the problem is EXPTIME-hard for most payoff functions and already undecidable for $\underline{\mathsf{MP}}$ and $\overline{\mathsf{MP}}$, we now fix the memory Eve can use.

▶ **Theorem 14.** *For $r \in \mathbb{Q}$, weighted automaton Γ and payoff function* Inf, Sup, LimInf, LimSup, $\underline{\mathsf{MP}}$, *or* $\overline{\mathsf{MP}}$*, determining whether $\mathbf{Reg}_{\Sigma_\exists^m,\mathfrak{W}_\forall}(\Gamma) \lhd r$, for $\lhd \in \{<, \leq\}$, can be done in* NTIME$(m^2|\Gamma|^2)$.

Sketch. Guess the strategy σ of Eve. Consider the non-deterministic automaton constructed from the synchronous product of the original machine and the deterministic automaton defined by the automaton restricted to transitions allowed by σ, this clearly has size at most $m|\Gamma|$. The weights of the transitions of the new automaton are set to the difference of the values of the functions of the two original automata. The language of the new machine is empty (for accepting threshold r) if and only if the desired property holds. As emptiness of a weighted automaton $\mathcal{A}$ can be decided in $O(|\mathcal{A}|^2)$ time [6], the result follows. ◀

We provide a matching lower bound. The proof is an adaptation of the NP-hardness proof from [1] (all the details are provided in the technical report [16]).

▶ **Theorem 15.** *For $r \in \mathbb{Q}$, weighted automaton Γ and payoff function* Inf, Sup, LimInf, LimSup, $\underline{\mathsf{MP}}$, *or* $\overline{\mathsf{MP}}$*, determining whether $\mathbf{Reg}_{\Sigma_\exists^1,\mathfrak{W}_\forall}(\Gamma) \lhd r$, for $\lhd \in \{<, \leq\}$, is* NP*-hard.*

Relation to other works. Let us first extend the definitions of *approximation, embodiment* and *refinement* from [1] to the setting of ω-words. Consider two weighted automata $\mathcal{A} = (Q_\mathcal{A}, q_I, A, \Delta_\mathcal{A}, w_\mathcal{A})$ and $\mathcal{B} = (Q_\mathcal{B}, q_I, A, \Delta_\mathcal{B}, w_\mathcal{B})$ and let $d : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be a *metric*.[3] We say $\mathcal{B}$ *(strictly) α-approximates* $\mathcal{A}$ *(with respect to d)* if $d(\mathcal{B}(w), \mathcal{A}(w)) \leq \alpha$ (resp. $d(\mathcal{B}(w), \mathcal{A}(w)) < \alpha$) for all words $w \in \Sigma^\omega$. We say $\mathcal{B}$ *embodies* $\mathcal{A}$ if $Q_\mathcal{A} \subseteq Q_\mathcal{B}$, $\Delta_\mathcal{A} \subseteq \Delta_\mathcal{B}$ and $w_\mathcal{A}$ agrees with $w_\mathcal{B}$ on $\Delta_\mathcal{A}$. For an automaton $\mathcal{A} = (Q, q_I, A, \Delta, w)$ and an integer $k \geq 0$, the k-refinement of $\mathcal{A}$ is the automaton obtained by refining the state space of $\mathcal{A}$ using k boolean variables. The automaton $\mathcal{A}$ is said to be *(strictly) (α, k)-determinisable by pruning* if the k-refinement of $\mathcal{A}$ embodies a deterministic automaton which (strictly) α-approximates $\mathcal{A}$. The next result follows directly from the above definitions.

▶ **Proposition 16.** *For non-negative $\alpha \in \mathbb{Q}$, $k \in \mathbb{N}$, a weighted automaton Γ is (strictly) (α, k)-DBP (w.r.t. the difference metric) iff $\mathbf{Reg}_{\Sigma^{2^k}_\exists, \mathfrak{W}_\forall}(\Gamma) \leq \alpha$ (resp. $\mathbf{Reg}_{\Sigma^{2^k}_\exists, \mathfrak{W}_\forall}(\Gamma) < \alpha$).*

In [14] the authors define *good for games automata*. Their definition is based on a game which is played on an ω-automaton by Spoiler and Simulator. We propose the following generalization of the notion of good for games automata for weighted automata. A weighted automaton $\mathcal{A}$ is *(strictly) α-good for games* if Simulator, against any word $w \in A^\omega$ spelled by Spoiler, can resolve non-determinism in $\mathcal{A}$ so that the resulting run has value v and $d(v, \mathcal{A}(w)) \leq \alpha$ (resp. $d(v, \mathcal{A}(w)) < \alpha$), for some metric d. We summarize the relationship that follows from the definition in the following result:

▶ **Proposition 17.** *For non-negative $\alpha \in \mathbb{Q}$, a weighted automaton Γ is (strictly) α-good for games (w.r.t. the difference metric) iff $\mathbf{Reg}_{\mathfrak{S}_\exists, \mathfrak{W}_\forall}(\Gamma) \leq \alpha$ (resp. $\mathbf{Reg}_{\mathfrak{S}_\exists, \mathfrak{W}_\forall}(\Gamma) < \alpha$).*

Acknowledgements. We thank Udi Boker for his comments on how to determinise LimSup automata.

References

1 Benjamin Aminof, Orna Kupferman, and Robby Lampert. Reasoning about online algorithms with weighted automata. *ACM Transactions on Algorithms*, 2010.

2 David E. Bell. Regret in decision making under uncertainty. *Operations Research*, 30(5):961–981, 1982.

3 Lubos Brim, Jakub Chaloupka, Laurent Doyen, Raffaella Gentilini, and Jean-François Raskin. Faster algorithms for mean-payoff games. *Formal Methods in System Design*, 38(2):97–118, 2011.

4 Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource interfaces. In *EMSOFT*, volume 2855 of *LNCS*, pages 117–133. Springer, 2003.

5 Krishnendu Chatterjee, Laurent Doyen, Emmanuel Filiot, and Jean-François Raskin. Doomsday equilibria for omega-regular games. In *VMCAI*, volume 8318, pages 78–97. Springer, 2014.

6 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM Trans. Comput. Log.*, 11(4), 2010.

7 Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Generalized mean-payoff and energy games. In *FSTTCS*, pages 505–516, 2010.

8 Werner Damm and Bernd Finkbeiner. Does it pay to extend the perimeter of a world model? In *FM*, volume 6664 of *LNCS*, pages 12–26. Springer, 2011.

[3] The metric used in [1] is the ratio measure.

9 Aldric Degorre, Laurent Doyen, Raffaella Gentilini, Jean-François Raskin, and Szymon Toruńczyk. Energy and mean-payoff games with imperfect information. In *CSL*, pages 260–274, 2010.

10 A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8:109–113, 1979.

11 Tali Eilam-Tzoreff. The disjoint shortest paths problem. *Discrete Applied Mathematics*, 85(2):113–138, 1998.

12 Emmanuel Filiot, Tristan Le Gall, and Jean-François Raskin. Iterated regret minimization in game graphs. In *MFCS*, volume 6281 of *LNCS*, pages 342–354. Springer, 2010.

13 Joseph Y. Halpern and Rafael Pass. Iterated regret minimization: A new solution concept. *Games and Economic Behavior*, 74(1):184–207, 2012.

14 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In *CSL*, pages 395–410, 2006.

15 Paul Hunter, Guillermo A. Pérez, and Jean-François Raskin. Mean-payoff games with partial-observation (extended abstract). In *Reachability Problems*, pages 163–175, 2014.

16 Paul Hunter, Guillermo A. Pérez, and Jean-François Raskin. Reactive synthesis without regret. *CoRR*, abs/1504.01708, 2015.

17 Marcin Jurdziński. Deciding the winner in parity games is in $\mathsf{UP} \cap \mathsf{coUP}$. *Information Processing Letters*, 68(3):119–124, 1998.

18 Marcin Jurdzinski, Jeremy Sproston, and François Laroussinie. Model checking probabilistic timed automata with one or two clocks. *LMCS*, 4(3), 2008.

19 Christos H. Papadimitriou and Mihalis Yannakakis. Shortest paths without a map. *Theoretical Computer Science*, 84(1):127–150, 1991.

20 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190. ACM, ACM Press, 1989.

21 Min Wen, Ruediger Ehlers, and Ufuk Topcu. Correct-by-synthesis reinforcement learning with temporal logic constraints. *arXiv preprint arXiv:1503.01793*, 2015.

22 Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In *NIPS*, pages 905–912, 2008.

23 Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *TCS*, 158(1):343–359, 1996.

Synthesis of Bounded Choice-Free Petri Nets

Eike Best[1] **and Raymond Devillers**[2]

1 **Department of Computing Science, Carl von Ossietzky Universität Oldenburg, Germany**
eike.best@informatik.uni-oldenburg.de

2 **Département d'Informatique, Université Libre de Bruxelles, Belgium**
rdevil@ulb.ac.be

Abstract

This paper describes a synthesis algorithm tailored to the construction of choice-free Petri nets from finite persistent transition systems. With this goal in mind, a minimised set of simplified systems of linear inequalities is distilled from a general region-theoretic approach, leading to algorithmic improvements as well as to a partial characterisation of the class of persistent transition systems that have a choice-free Petri net realisation.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Choice-Freeness, Labelled Transition Systems, Persistence, Petri Nets, System Synthesis

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.128

1 Introduction, some examples, and basic notation

In system analysis, the main task is to examine a given system's properties by means of a behavioural description. By contrast, in system synthesis, the task is to construct – preferably automatically – an implementing system from a given behavioural specification. The benefit of such an approach is that a successfully synthesised system is "correct by design". There is no need to re-examine its behavioural properties, because they are known to hold by construction. If synthesis fails, this may also help to delineate the true reasons of the failure, paving the way to modifications of the given input behaviour allowing for a more successful subsequent synthesis.

Synthesis is being applied in many different areas (e.g., [11, 19]). In general, however, since behavioural descriptions may be extremely (even infinitely) large, synthesis algorithms could be impossible to obtain by theoretical undecidability [14], or at least be very time-consuming. Also, synthesis suffers from nondeterminism, since for a given behavioural specification, many different implementations may exist. Moreover, if there is a desire for an implementation to enjoy further properties, detecting the existence of a suitable one (if possible) tends to increase the difficulty of a synthesis problem.

We investigate a special, decidable instance of system synthesis. It is assumed that a behavioural specification is given in the form of a finite, edge-labelled transition system, or *lts*, for short. For example, we could be interested in the transition system TS_1 shown on the left-hand side of Figure 1. We shall be asking whether or not such an lts can be implemented by an unlabelled Petri net having a specific shape. The shape we shall be aiming at is *choice-freeness*, meaning that every place has at most one outgoing transition. For example, both Petri nets N_1 and N_1' shown in Figure 1 implement TS_1, in the sense that their reachability graphs are isomorphic to TS_1. However, N_1 is choice-free while N_1' is not.

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 128–141

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

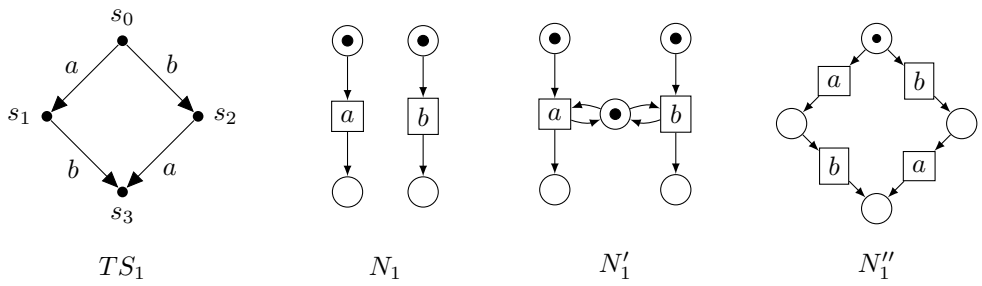


Figure 1 The lts TS_1 is solved by the Petri net N_1. It is also solved by N_1'. The net N_1'' is not (in this paper) accepted as a solution of TS_1 because its transitions are non-injectively labelled.

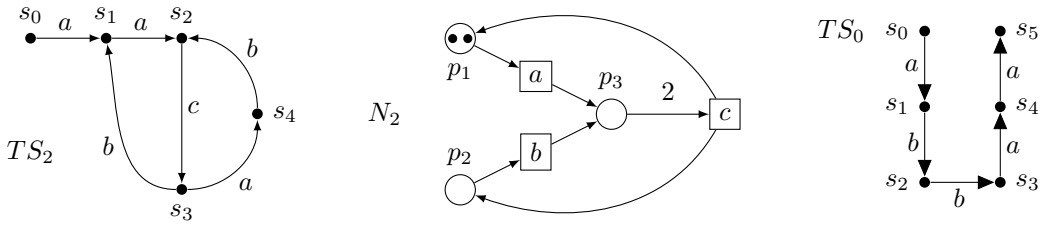


Figure 2 The net N_2 solves TS_2. No plain solution of TS_2 exists. No solution of TS_0 exists.

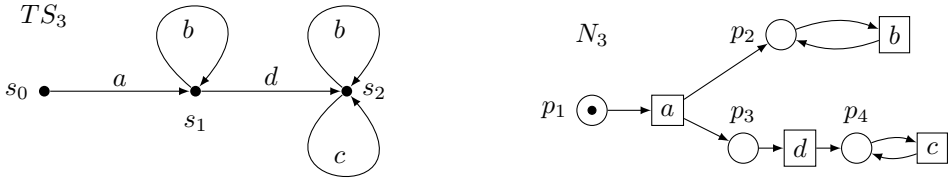


Figure 3 The Petri net N_3 solves the lts TS_3. No pure solution of TS_3 exists.

Being related to arbiter-freeness [16], choice-freeness is interesting in a digital design context [11]. Choice-free Petri nets are also precisely the class of nets allowing a fully distributed [1] implementation. The problem has been addressed and solved for special classes of choice-free nets in previous papers by the present authors, as follows: for connected marked graphs and T-systems in [5, 7]; for bounded, reversible choice-free nets (i.e., where it is always possible to come back to the initial state) in [6, 8]; and for connected, bounded, live choice-free nets (i.e., where no transition may become dead) in [9]. In the present paper, this framework will be generalised to bounded choice-free nets, also allowing for non-live transitions.

We shall be concerned with exact synthesis, disallowing that two or more transitions carry the same label. This excludes nets such as N_1'' in Figure 1 as implementations. Moreover, we shall take into consideration the full class of place/transition systems [18]. For example, the lts TS_2 depicted in Figure 2 can be solved by N_2 with an arc having weight 2 from p_3 to c, but not by any plain (meaning: having arc weights at most 1) Petri net. Similarly, the lts TS_3 shown in Figure 3 can be solved by N_3, but not by a pure (meaning: side-place free) Petri net. Observe that there are also specifications which cannot be implemented by any unlabelled Petri net, such as the lts TS_0 shown on the right-hand side of Figure 2 ([3]). The proofs of partial or full unsolvability are not hard and are left to the reader; [10] may help.

For easy reference, basic formal definitions are summarised in the remainder of this section. Important concepts with strong impact on the formal development of this paper will

be introduced in-line, that is, in the text explaining their relevance. To facilitate spotting them, such notions will be emphasised in *italic* at the point of their formal introduction.

▶ **Definition 1.1** (Basic notations and conventions used in this paper)**.** A *finite labelled transition system* with initial state is a tuple $TS = (S, \rightarrow, T, s_0)$ with nodes S (a finite set of states), edge labels T, edges $\rightarrow \subseteq (S \times T \times S)$, and an initial state $s_0 \in S$. A label t is enabled at $s \in S$, written formally as $s[t\rangle$, if $\exists s' \in S\colon (s,t,s') \in \rightarrow$, and backward enabled at s, written as $[t\rangle s$, if $\exists s' \in S\colon (s',t,s) \in \rightarrow$. A state s' is reachable from s through the execution of $\sigma \in T^*$, denoted by $s[\sigma\rangle s'$, if there is a directed path from s to s' whose edges are labelled consecutively by σ. The set of states reachable from s is denoted by $[s\rangle$. A (firing) sequence $\sigma \in T^*$ is allowed from a state s, denoted by $s[\sigma\rangle$, if there is some state s' such that $s[\sigma\rangle s'$. The language of TS is the set $L(TS) = \{\sigma \in T^* \mid s_0[\sigma\rangle\}$. Two lts $TS_1 = (S_1, \rightarrow_1, T, s_{01})$ and $TS_2 = (S_2, \rightarrow_2, T, s_{02})$ are language-equivalent if $L(TS_1) = L(TS_2)$, and isomorphic if there is a bijection $\zeta\colon S_1 \rightarrow S_2$ with $\zeta(s_{01}) = s_{02}$ and $(s,t,s') \in \rightarrow_1 \Leftrightarrow (\zeta(s),t,\zeta(s')) \in \rightarrow_2$, for all $s, s' \in S_1$.

An *initially marked Petri net* is denoted as $N = (P, T, F, M_0)$ where P is a finite set of places, T is a finite set of transitions, F is the flow function $F\colon ((P \times T) \cup (T \times P)) \rightarrow \mathbb{N}$ specifying the arc weights, and M_0 is the initial marking (where a marking is a mapping $M\colon P \rightarrow \mathbb{N}$, indicating the number of tokens in each place). N is plain if no arc weight exceeds 1; pure or side-place free if $\forall p \in P\colon (p^\bullet \cap {}^\bullet p) = \emptyset$, where $p^\bullet = \{t \in T \mid F(p,t){>}0\}$ and ${}^\bullet p = \{t \in T \mid F(t,p){>}0\}$; and CF (*choice-free* [12, 20]) or ON (place-output-nonbranching [8]) if $\forall p \in P\colon |p^\bullet| \leq 1$. A transition $t \in T$ is enabled at a marking M, denoted by $M[t\rangle$, if $\forall p \in P\colon M(p) \geq F(p,t)$. The firing of t leads from M to M', denoted by $M[t\rangle M'$, if $M[t\rangle$ and $M'(p) = M(p) - F(p,t) + F(t,p)$. This can be extended, as usual, to $M[\sigma\rangle M'$ for sequences $\sigma \in T^*$, and $[M\rangle$ denotes the set of markings reachable from M. The reachability graph $RG(N)$ of N is the labelled transition system with the set of vertices $[M_0\rangle$, initial state M_0, label set T, and set of edges $\{(M,t,M') \mid M, M' \in [M_0\rangle \wedge M[t\rangle M'\}$. If an lts TS is isomorphic to the reachability graph of a Petri net N, we say that N *solves* TS. All notions defined for labelled transition systems apply to Petri nets through their reachability graphs. ◀ 1.1

2 Basic synthesis of Petri nets, recapitulated

Certain classes of lts can be excluded from consideration up front. For instance, it might happen that in some transition system $TS = (S, \rightarrow, T, s_0)$, some state s is not reachable from the initial state s_0. Such an lts can never be solved by a Petri net, since the reachability graph is defined by adding all reachable markings (and no others). Hence we shall adopt, for TS, *total reachability*, meaning that $\forall s \in S\colon s \in [s_0\rangle$.

Nondeterminism can never occur in the reachability graph of a Petri net, because if $M[t\rangle M'$, then the successor marking M' is uniquely determined by M and t. Generalising this, let $\Psi(\sigma)$ denote the *Parikh vector* of a sequence $\sigma \in T^*$, i.e., the vector with index set T which returns the number of occurrences of $t \in T$ in σ, and call an lts *strongly deterministic* if, whenever $\Psi(\tau) = \Psi(\tau')$ and either $s[\tau\rangle s'$ and $s[\tau'\rangle s''$ or $s'[\tau\rangle s$ and $s''[\tau'\rangle s$, then $s' = s''$.

If some sequence τ is cyclic in the reachability graph of a Petri net, i.e., leads from some marking M to itself, $M[\tau\rangle M$, then (according to standard Petri net theory) any sequence which is Parikh-proportional to τ is also cyclic, at any marking at which it is enabled. Let us call an lts *strongly cycle-consistent* if the same property holds for it.

For the sake of brevity, let us call an lts *decent* if it is totally reachable, strongly deterministic, and strongly cycle-consistent. A non-decent lts has no possible Petri net

solution, and it can therefore be interesting to first check this structural constraint in order to avoid applying uselessly a costly regional analysis. This observation is exploited by some tools (e.g. [21]).

A very general (but also expensive) algorithm for synthesising a Petri net from a labelled transition system can be described as follows.

- For a given finite and decent lts $TS = (S, \rightarrow, T, s_0)$, we start constructing a net by letting it have T as its set of transitions, and no places. Such a net has the language T^*, which (normally) contains many more words than $L(TS)$. In order to exclude words disallowed in TS, and in order to guarantee a bijection between TS and the reachability graph of the hoped-for net N, a place will be introduced for every separation problem in TS as follows.
- An *event/state separation problem* consists of an ordered pair $(s,t) \in S \times T$ with $\neg(s[t\rangle)$. There are at most $|S| \cdot |T|$ such problems. For every event/state separation problem, N needs to have at least one place p such that $M(p) < F(p,t)$ for the marking M corresponding to state s, where $F(p,t)$ is the weight of the arc from p to t.
- A *state separation problem* consists of a pair of states $\{s, s'\}$ with $s \neq s'$. There are $\frac{1}{2} \cdot (|S| \cdot (|S|-1))$ such problems. For every state separation problem, N needs to contain at least one place p such that $M(p) \neq M'(p)$ for the markings M and M' corresponding to states s and s', respectively.
- The notion of a "place" is not known for TS. A *region* [2] of an lts $(S, \rightarrow, T, s_0)$ is a triple $(\mathbb{R}, \mathbb{B}, \mathbb{F}) \in (S \rightarrow \mathbb{N}, T \rightarrow \mathbb{N}, T \rightarrow \mathbb{N})$ such that for all $s[t\rangle s'$, $\mathbb{R}(s) \geq \mathbb{B}(t)$ and $\mathbb{R}(s') = \mathbb{R}(s) - \mathbb{B}(t) + \mathbb{F}(t)$. A region models a place p, in the sense that $\mathbb{B}(t)$ models $F(p,t)$, $\mathbb{F}$ models $F(t,p)$, and $\mathbb{R}(s)$ models the token count of p in the marking corresponding to s.
- A straightforward algorithm inspects every separation problem in turn and tries to solve a linear inequality system for it. The unknowns are the arc weights of a place p with respect to every transition in T, and the initial marking $M_0(p)$. The inequality system arises from the need to guarantee the region properties (giving rise to many inequalities), and from the need to guarantee a separation property (giving rise to one or two additional inequalities). If these systems are solvable for every separation problem, we find a net which is isomorphic to TS, otherwise such a net does not exist. If they are solvable for every event/state separation problem, then we can construct a Petri net which is language-equivalent to TS.

Thus, in general, we need to solve $O(|S|^2)$ inequality systems, each with more than $2 \cdot |T|$ unknowns. In the present paper, we ask whether a given finite, decent lts has a choice-free Petri net solution. If such a requirement is added, the algorithm could become more complex; but the aim of this paper is to demonstrate that, knowing what solutions we are looking for may also work the other way, namely focussing the search and speeding up the region-based general algorithm.

The reachability graphs of choice-free Petri nets necessarily satisfy an additional set of properties which are not shared by all finite labelled transition systems, even if they are decent. This excludes many decent ones from consideration. In the next Section 3, we shall gather a set of such properties. In Section 4, we describe how these properties and the special shape of the places (and regions) of a choice-free net can be exploited in order to simplify the region inequalities.

3 Persistent transition systems, small cycles, and CF Petri nets

The reachability graphs of choice-free Petri nets are persistent, and persistent lts enjoy a property of small cycles, as will be described next.

An lts $TS = (S, \to, T, s_0)$ is called *persistent* [17] if, whenever $s[a\rangle$ and $s[b\rangle$ with $a \neq b$, then also $s[ab\rangle s'$ and $s[ba\rangle s'$ for some common state s'. For example, all of the lts shown in figures 1–3 (including TS_0) are persistent. Any choice-free net $N = (S, T, F, M_0)$ is persistent, because if $a \neq b$ for $a, b \in T$, then there is no common pre-place p of a and b, i.e., for all $p \in P$, either $F(p, a) = 0$ or $F(p, b) = 0$, or both, which directly entails the persistence property, if we add the strong determinism of any Petri net.

The property of small cycles generalises the following observations. First, define a *home state* of TS to be a state $\widetilde{s} \in S$ which satisfies $\forall s \in [s_0\rangle \colon \widetilde{s} \in [s\rangle$. Finite persistent lts always have at least one home state (with an easy proof; see, e.g. corollary 2 of [4]). The sets of home states of TS_0, TS_1, TS_2 and TS_3 of figures 1–3 are, respectively, $\{s_5\}$, $\{s_3\}$, $\{s_1, s_2, s_3, s_4\}$, and $\{s_2\}$. Next, define a nontrivial cycle $s[\sigma\rangle s$ around a state $s \in [s_0\rangle$ to be *small* if there is no nontrivial cycle $s'[\sigma'\rangle s'$ with $s' \in [s_0\rangle$ and $\Psi(\sigma') \lneqq \Psi(\sigma)$, where $\lneqq = \ (\leq \cap \neq)$. For example, in TS_2, both $s_3[bac\rangle s_3$ and $s_3[abc\rangle s_3$, and also $s_1[acb\rangle s_1$ (and others) are small cycles, but $s_1[acabcb\rangle s_1$ (and others) are not. Notice that in TS_2, all small cycles have the same Parikh vector. In TS_3, by contrast, $s_1[b\rangle s_1$ is small, $s_2[b\rangle s_2$ is small, $s_2[c\rangle s_2$ is small, but $s_2[bc\rangle s_2$ is not. Notice that in TS_3, all small cycles either have the same Parikh vectors, or are label-disjoint, where two Parikh vectors are *label-disjoint* if their supports are disjoint, and the *support* of a Parikh vector $\Psi \colon T \to \mathbb{N}$ is defined as the set $\{t \in T \mid \Psi(t) > 0\}$.

This property is general, as follows.

▶ **Theorem 3.1** (Cycle decomposition at home states). *Let $TS = (S, \to, T, s_0)$ be a finite, decent, persistent lts. Then there exist a state $\widetilde{s} \in [s_0\rangle$ and a finite set $\mathcal{C} = \{\widetilde{s}[\rho_i\rangle\widetilde{s} \mid 1 \leq i \leq n\}$ of mutually label-disjoint small cycles around $\widetilde{s}$, with $n \leq |T|$, such that for any state $s \in [s_0\rangle$, the Parikh vector of any cycle $s[\rho\rangle s$ decomposes as $\Psi(\rho) = \sum_{i=1}^{n} k_i \cdot \Psi(\rho_i)$ for some $k_i \in \mathbb{N}$.*

Proof. From theorem 2 of [4], observing that the preconditions of this result are implied by finiteness, decency, and persistence, and keeping in mind that small cycles have been called hypersimple in [4]. ◀

In fact, the results of [4] also show that for $\widetilde{s}$, any home state can be chosen and that the set of Parikh vectors in $\mathcal{C}$ is independent of the choice of home state. For example, in TS_2, we may choose $\widetilde{s} = s_1$ and $\mathcal{C} = \{s_1[acb\rangle s_1\}$ with $|\mathcal{C}| = 1$, and in TS_3, $\widetilde{s} = s_2$ is the only possible choice, and we get $\mathcal{C} = \{s_2[b\rangle s_2, s_2[c\rangle s_2\}$ with $|\mathcal{C}| = 2$. In TS_0 and TS_1, $\widetilde{s} = s_5$ and $\widetilde{s} = s_3$ (respectively), and we have $\mathcal{C} = \emptyset$ in both cases. If a persistent Petri net $N = (S, T, F, M_0)$ is *bounded*, i.e., has a finite reachability graph $RG(N)$, then it satisfies the premises of the previous theorem. Hence the cycles of $RG(N)$ can be decomposed in the same way.

The decomposition theorem has implications for the distribution of live transitions in TS. A label $t \in T$ is *live* if $\forall s \in [s_0\rangle \exists s' \in [s\rangle \colon s'[t\rangle$. If TS is finite, decent, and persistent, then live transitions are exactly those that can be found in one of the cycles in $\mathcal{C}$. All others (for instance, a and d in TS_3, but also a and b in TS_0) can never again be executed, once a home state has been reached. We cast these notions in the following definition.

▶ **Definition 3.2** (The small cycles property $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$). Let $TS = (S, \to, T, s_0)$ be an lts. For some $n \in \mathbb{N}$ and for all $1 \leq i \leq n$, let Υ_i be a function $\Upsilon_i \colon T \to \mathbb{N}$ such that these functions are mutually label-disjoint. TS satisfies property $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$ iff $\{\Upsilon_1, \ldots, \Upsilon_n\}$ is the set of Parikh vectors of small cycles of TS. ◀ 3.2

If an lts TS satisfies $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$, then we shall henceforth denote by T_i the support of Υ_i and by T_0 the set $T \setminus (T_1 \cup \ldots \cup T_n)$. By definition, these $n+1$ sets are mutually disjoint. In case TS is finite, decent, and persistent, the set on non-live transitions is exactly T_0. For example, TS_2 and TS_0 in Figure 2 satisfy $\mathbf{P}\{\Upsilon_1\}$ and $\mathbf{P}\emptyset$, respectively, where Υ_1 maps a, b and c to 1. Also, T_0 is $\emptyset$ in TS_2 and $\{a, b\}$ in TS_0. In Figure 3, TS_3 satisfies $\mathbf{P}\{\Upsilon_1, \Upsilon_2\}$ where $\Upsilon_1 = (a \mapsto 0, b \mapsto 1, c \mapsto 0, d \mapsto 0)$, $\Upsilon_2 = (a \mapsto 0, b \mapsto 0, c \mapsto 1, d \mapsto 0)$, and $T_0 = \{a, d\}$.

A coherent notion of distance between states can be introduced as follows.

▶ **Definition 3.3** (Modulo vectors and distances). Let $TS = (S, \to, T, s_0)$ be a finite, decent, and persistent lts satisfying $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$.
For a natural T-vector Υ, let

$$\Upsilon \bmod \{\Upsilon_1, \ldots, \Upsilon_n\} \quad = \quad \Upsilon - \sum_{i \in \{1,\ldots,n\}} \left(\min_{t \in T_i} \left(\Upsilon(t) \div \Upsilon_i(t) \right) \right) \cdot \Upsilon_i$$

be the natural T-vector obtained by subtracting each of the vectors Υ_i as often as possible from Υ (in the formula, $\div$ denotes integer division). Let $r \in S$, $s \in [r\rangle$ and $r[\alpha\rangle s$ be a path of TS. Then $\Delta_{r,s} = \Psi(\alpha) \bmod \{\Upsilon_1, \ldots, \Upsilon_n\}$ is called the *distance* between r and s. ◀ 3.3

The following lemma shows that $\Delta_{r,s}$ does not depend on the path chosen between r and s.

▶ **Lemma 3.4** (istances are well-defined). *Let $TS = (S, \to, T, s_0)$ be a finite, decent, and persistent lts satisfying $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$.*
Let $r[\sigma\rangle s$ and $r[\sigma'\rangle s$ be two paths between the same states $r, s \in [s_0\rangle$.
Then $\Psi(\sigma) \bmod \{\Upsilon_1, \ldots, \Upsilon_n\} = \Psi(\sigma') \bmod \{\Upsilon_1, \ldots, \Upsilon_n\}$.

For the proof of this lemma, we use *Keller's theorem* [15], a basic tool for analysing persistent systems. For sequences $\sigma, \tau \in T^*$, let $\tau \stackrel{\bullet}{-} \sigma$ denote the sequence left after erasing successively in τ the leftmost occurrences of all symbols from σ, read from left to right. Keller's theorem states that in a deterministic and persistent lts, if $s[\tau\rangle$ and $s[\sigma\rangle$ for some $s \in [s_0\rangle$, then $\Psi(\tau(\sigma \stackrel{\bullet}{-} \tau)) = \Psi(\sigma(\tau \stackrel{\bullet}{-} \sigma))$ and $s[\tau(\sigma \stackrel{\bullet}{-} \tau)\rangle \widehat{s}$ and $s[\sigma(\tau \stackrel{\bullet}{-} \sigma)\rangle \widehat{s}$ for some state $\widehat{s} \in [s_0\rangle$.

Proof. Applied to $r[\sigma\rangle s$ and $r[\sigma'\rangle s$, Keller's theorem yields $s[\sigma \stackrel{\bullet}{-} \sigma'\rangle \widehat{s}$ and $s[\sigma' \stackrel{\bullet}{-} \sigma\rangle \widehat{s}$. The definition of $\stackrel{\bullet}{-}$ implies that $\sigma \stackrel{\bullet}{-} \sigma'$ and $\sigma' \stackrel{\bullet}{-} \sigma$ are label-disjoint. Lemma 4 in [4] states that in a finite, deterministic, strongly cycle-consistent and persistent lts, two paths between two different states always have a common label. This implies $s = \widehat{s}$.

Thus, both $s[\sigma \stackrel{\bullet}{-} \sigma'\rangle s$ and $s[\sigma' \stackrel{\bullet}{-} \sigma\rangle s$ are cyclic, and by Theorem 3.1, both $\Psi(\sigma \stackrel{\bullet}{-} \sigma')$ and $\Psi(\sigma' \stackrel{\bullet}{-} \sigma)$ are linear combinations of $\Upsilon_1, \ldots, \Upsilon_n$. On T_0, they both must be null, so that $\Psi(\sigma)$ and $\Psi(\sigma')$ coincide on T_0. On each T_i with $i \in \{1, \ldots, n\}$, since $\sigma \stackrel{\bullet}{-} \sigma'$ and $\sigma' \stackrel{\bullet}{-} \sigma$ are label-disjoint, at least one of them must be the empty sequence; hence, on T_i, one of $\Psi(\sigma)$ or $\Psi(\sigma')$ must be greater or equal to the other, by a multiple of Υ_i. ◀

In Figure 4, TS_4 satisfies $\mathbf{P}\{\Upsilon_1, \Upsilon_2\}$ where $\Upsilon_1 = (a \mapsto 0, b \mapsto 1, c \mapsto 1, d \mapsto 0, e \mapsto 0) = \Psi(bc)$ and $\Upsilon_2 = (a \mapsto 0, b \mapsto 0, c \mapsto 0, d \mapsto 1, e \mapsto 1) = \Psi(de)$. There are two Parikh-incomparable paths $s_0[bac\rangle s$ and $s_0[dae\rangle s$, and no smaller ones from s_0 to s. Yet the distance $\Delta_{s_0,s}$ is uniquely defined as the Parikh vector $\Delta_{s_0,s} = (a \mapsto 1, b \mapsto 0, c \mapsto 0, d \mapsto 0, e \mapsto 0) = \Psi(a)$, and it can be obtained either by subtracting Υ_1 from $\Psi(bac)$ or by subtracting Υ_2 from $\Psi(dae)$.

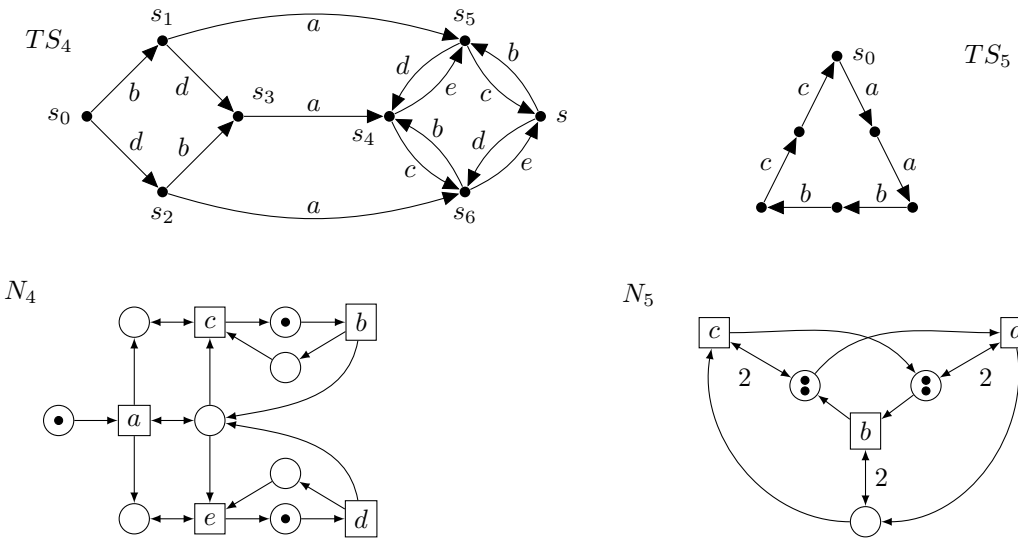


Figure 4 Upper part: Two lts which are PN-solvable but have no choice-free Petri net solutions. Lower part: Solutions N_4 of TS_4 (left-hand side) and N_5 of TS_5 (right-hand side).

The properties defined thus far (total reachability, determinism, cycle-consistency, persistence, and the small cycle property) are shared by all reachability graphs of bounded persistent Petri nets. The reachability graphs of bounded choice-free Petri nets enjoy further (stronger) properties, two of which, the prime cycle property and the distance property, are described in the remainder of this section. If any of these properties is violated for some lts, then it is certain that choice-free synthesis (to be defined in the next section) will fail for it.

▶ **Definition 3.5** (Prime cycles). Let $TS = (S, \to, T, s_0)$ be any lts and $s[\sigma\rangle s$ a cycle in it. This cycle is called *prime* if $\gcd\{\Psi(\sigma)(t) \mid t \in T\} = 1$ (where gcd denotes the greatest common divisor). ◀ 3.5

▶ **Lemma 3.6** (Prime cycle property). *In the reachability graph of a bounded choice-free net, all small cycles are prime.*

Proof. Lemma 16 in [20] states that in a choice-free net (P, T, F, M_0), if there is a T-semiflow X (i.e., a T-indexed vector $X \geq 0$ such that $\forall p \in P\colon \sum_{t \in T}(F(t,p) - F(p,t)) \cdot X(t) = 0$), and a firing sequence $M_0[\sigma\rangle M$ with $\Psi(\sigma) \geq X$, then it is possible to rearrange σ in such a way that $M_0[\sigma'\rangle M[\rho\rangle M$ with $\Psi(\rho) = X$ and $\Psi(\sigma'\rho) = \Psi(\sigma)$. This implies that at home states, non-prime cycles can be factored out into a nontrivial initial part followed by a prime cycle executing exactly X, and hence are not small. ◀

This is illustrated by TS_5 shown on the right-hand side of Figure 4. It is finite, decent, persistent, and satisfies $\mathbf{P}\{\Upsilon_1\}$ with $\Upsilon_1 = (a \mapsto 2, b \mapsto 2, c \mapsto 2)$. It is PN-solvable by N_5, as also shown in Figure 4, but, as a consequence of Lemma 3.6, may not be solved by a choice-free net.

▶ **Definition 3.7** (Parikh-minimal paths). Let $TS = (S, \to, T, s_0)$ be any lts, let $r \in S$, let $s \in [r\rangle$, and let $r[\sigma\rangle s$ be a path from r to s. The latter is called *Parikh-minimal* if there is no path $r[\sigma'\rangle s$ with $\Psi(\sigma') \lneqq \Psi(\sigma)$. ◀ 3.7

The following lemma implies that in choice-free nets, the distance $\Delta_{r,s}$ between two states r and s agrees with the minimal Parikh vector of any path from r to s. In particular, unlike in TS_4, there are no Parikh-incomparable Parikh-minimal paths between the same states.

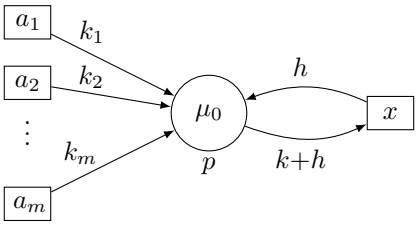


Figure 5 A general pure ($h = 0$) or non-pure ($h > 0$) choice-free place with initial marking μ_0.

▶ **Lemma 3.8** (Distance property). *In the reachability graph $RG(N)$ of a bounded choice-free net N, if $r[\sigma\rangle s$ is a Parikh-minimal path, then $\Psi(\sigma) = \Delta_{r,s}$. Also, if $q[\rho\rangle q$ is a small cycle in $RG(N)$, then $\Psi(\sigma) \not\geq \Psi(\rho)$.*

Proof. This again follows from an iterated application of Lemma 16 in [20]. ◀

For example, the lts TS_4 shown in Figure 4 is PN-solvable (by N_4, also shown in the figure), finite, decent, persistent, and satisfies $\mathbf{P}\{\Upsilon_1, \Upsilon_2\}$ as already mentioned, but may not be solved by a choice-free net, since, for instance, $s_0[bac\rangle s$ is Parikh-minimal but $\Psi(bac) \neq \Delta_{s_0,s} = \Psi(a)$; also, $\Psi(bac) \geq \Upsilon_1 = (a \mapsto 0, b \mapsto 1, c \mapsto 1, d \mapsto 0, e \mapsto 0)$.

4 Choice-free synthesis

In this section, it will be shown how the special form of the target places of a hoped-for choice-free Petri net solving a given lts can be exploited in order to reduce the number, and to simplify the shape, of the linear inequality systems that need to be solved. Concretely, let us consider an lts $TS = (S, \to, T, s_0)$ which is finite, decent, persistent, and satisfies $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$. We shall analyse when and how one can synthesise a corresponding choice-free net $N = (P, T, F, M_0)$. As before, the set of transitions of N is the same as the set of labels of TS. Since N is intended to be choice-free, every place $p \in P$ has the general form shown in Figure 5, with $x \in T$ as its only outgoing transition, and $\{a_1, \ldots, a_m\} = T \setminus \{x\}$. All arc weight parameters $k, h, k_1, k_2, \ldots, k_m$ and the initial marking μ_0 of p are required to be semipositive, and they are the unknowns of the synthesis. If p is used for preventing x at some state s, i.e., for solving some event/state separation problem $\neg(s[x\rangle)$, we must have $k + h > 0$; but, for the time being, any of these parameters could also be zero.
Let $T = T_0 \uplus T_1 \uplus \ldots \uplus T_n$ be the partition of T induced by $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$. For $0 \leq i \leq n$, we denote by $I_i = \{j \mid a_j \in T_i\}$ the indices of transitions a_j for which $\Upsilon_i(a_j) > 0$.
In the following, we shall also denote by ℓ the unique index such that $x \in T_\ell$.

Ensuring that cycles are preserved. Since the net effect of firing x on p is $-k = (-(k+h)+h)$ and all Parikh vectors in $\{\Upsilon_1, \ldots, \Upsilon_n\}$ are cyclic, we must have

$$\forall i \in \{1, \ldots, n\} : \quad \sum_{j \in I_i} k_j \cdot \Upsilon_i(a_j) \quad = \quad k \cdot \Upsilon_i(x) \tag{1}$$

ensuring that if every transition t is fired $\Upsilon_i(t)$ times, the marking on p is reproduced. Note that this implies $k \geq 0$, and even $k > 0$ unless all the k_j's for $j \in I_i$ are null.

If $x \in T_0$, i.e., $\ell = 0$, all the right-hand sides of (1) are null, so that all k_j for $j \notin I_0$ must be null too; in other words, if $p^\bullet \subseteq T_0$, then ${}^\bullet p \subseteq T_0$. Thus, if x is non-live, then all transitions in ${}^\bullet p$ are also non-live. If $x \in T_\ell$ for $\ell \in \{1, \ldots, n\}$, the right-hand sides of (1) are

null when $i \in \{1,\dots,n\} \setminus \{\ell\}$, so that all k_j for $j \notin I_0 \cup I_\ell$ must be null too; in other words, if $p^\bullet \subseteq T_\ell$, then ${}^\bullet p \subseteq T_0 \cup T_\ell$. Thus, if x is live and part of a small cycle, then all transitions in ${}^\bullet p$ are either non-live, or live and part of the same small cycle.

Ensuring that the marking on p does not prevent enabled transitions. By the shape of the place shown in Figure 5, by ${}^\bullet p \subseteq T_0 \cup T_\ell$, and by the firing rule, the marking of place p at (the marking corresponding to) an arbitrary state $r \in [s_0\rangle$ is

$$M_r(p) \;=\; \mu_0 + \sum_{j \in I_0 \cup I_\ell} k_j \cdot \Delta_{s_0,r}(a_j) - k \cdot \Delta_{s_0,r}(x) \tag{2}$$

This sum must always be nonnegative. Let us start by analysing what it means that p may never prevent any enabled x. Thus, consider an edge $r[x\rangle r'$ in TS. Then the marking $M_r(p)$ has to be at least $k+h$, and $M_{r'}(p)$ is at least h. More generally, if $l > 0$ and $r[x^l\rangle r'$, then we must have $M_r(p) \geq l \cdot k + h$, or equivalently, $M_{r'}(p) \geq h$. For this reason, when considering the marking of p at a state r with $r[x\rangle$, we first try to follow x-chains in forward direction as long as possible. If $r[x\rangle r'[x\rangle r''$ with $r \neq r'$, then by strong determinism, r'' is necessarily different from r and from r', so that x-chains starting with two different states can never hit an x-cycle, nor an x-branch, and necessarily have a unique last state. We may have infinite x-paths, but only in case $r[x\rangle r$ (like those for b or for c in Figure 3), where state r can never be left by an x-edge. Thus, we are interested in the following subset of states:

$$XNX(x) \;=\; \{r \in S \mid [x\rangle r \wedge \neg r[x\rangle\} \;\cup\; \{r \in S \mid r[x\rangle r\}$$

which either are produced by x but do not enable x, or have an x-loop. The above considerations amount to a proof of the following corollary:

▶ **Corollary 4.1** (XNX and enabling condition). $\big(\forall r \in S\colon r[x\rangle \Rightarrow M_r(p) \geq k+h\big) \iff \big(\forall r \in XNX(x)\colon M_r(p) \geq h\big)$.

In other words, we only need to require $M_r(p) \geq h$ for states $r \in XNX(x)$ in order to guarantee that place p allows x whenever it is enabled. But we can do more. Suppose that $r, r' \in XNX(x)$ and $r \in [r'\rangle$, and consider the case that $r'[\alpha\rangle r$ where α is x-free. Then α acts semipositively on p, and we only need to require $M_{r'}(p) \geq h$ on r', in order to have $M_r(p) \geq h$ on r. For this reason, before imposing the requirement $M_r(p) \geq h$ on some $r \in XNX(x)$, we may try to follow x-free backward chains starting at r, hoping to find some $r' \in XNX(x)$ such that imposing $M_{r'}(p) \geq h$ on r' implies $M_r(p) \geq h$ for r. In doing so, we may hit a cycle. However, by Theorem 3.1, such a cycle may not contain any transition from $T_0 \cup T_\ell$. This is because cycles may never contain transitions from T_0 anyway, and because, while a cycle might intersect T_ℓ if $\ell > 0$, it would then also have to contain x (but we only follow x-free backward chains). Thus, the cycle we may be hitting could at most be formed by transitions in $T \setminus (T_0 \cup T_\ell)$; however, this has no importance since – as we already know – such cycles do not modify the marking of p. Let us therefore consider the following equivalence relation between states q, q':

$$q \equiv_\ell q' \quad\iff\quad q[\beta\rangle q' \text{ and } q'[\beta'\rangle q \text{ with } \beta, \beta' \in (T \setminus (T_0 \cup T_\ell))^*$$

and let us define

$$MXNX(x) = \{r \in XNX(x) \mid \nexists r' \in XNX(x) : r' \not\equiv_\ell r \text{ and } r'[\alpha\rangle r \text{ with } \alpha \in (T \setminus \{x\})^+\}$$

i.e., we only consider states in $XNX(x)$ which do not lie x-freely after another (non-$\equiv_\ell$-equivalent) one. This set may contain many $\equiv_\ell$-equivalent states, but we need only keep one of them. Let us therefore choose some set

$$\boxed{mXNX(x) \subseteq MXNX(x) \text{ with a single representative of each } \equiv_\ell\text{-equivalent class in it}}$$

These considerations amount to a proof of

▶ **Corollary 4.2** ($mXNX$ and enabling condition). $\big(\forall r \in S\colon r[x\rangle \Rightarrow M_r(p) \geq k + h\big) \iff \big(\forall r \in mXNX(x)\colon M_r(p) \geq h\big)$.

In other words, we only need to require $M_r(p) \geq h$ for states $r \in mXNX(x)$ in order to guarantee that place p allows x whenever it is enabled. Combining Corollary 4.2 with the formula (2) relating any $M_r(p)$ to μ_0 yields the following set of constraints:

$$\forall r \in mXNX(x)\colon \quad \mu_0 \quad \geq \quad k \cdot \Delta_{s_0,r}(x) - \sum_{j \in I_0 \cup I_\ell} k_j \cdot \Delta_{s_0,r}(a_j) + h \tag{3}$$

Let us now re-examine the constraint that $\forall r \in [s_0\rangle\colon M_r(p) \geq 0$. By $r \in [s_0\rangle$, there is a path $s_0[\alpha\rangle r$. If α contains an x, let s be the visited state before the last x; from the previous constraints, $M_s(p) \geq k + h$ and $M_r(p) \geq h \geq 0$. If α contains no x, then it has a semipositive effect on p and thus, $\mu_0 \leq M_r(p)$. It is then enough to impose $\mu_0 \geq 0$ to get the desired property $M_r(p) \geq 0$. However, $\mu_0 \geq 0$ can always be ensured by adding, if necessary, an adequate shift to all markings of p, as well as to h.

In sum, requiring the non-negative solvability of (3) suffices in order to find parameters $k, h, k_1, \ldots, k_m, \mu_0$ in such a way that the corresponding region (and then, a corresponding place p with initial marking $M_0(p) = \mu_0$) allows all the paths that are possible in TS.

Ensuring that place p solves an event/state separation problem. For each state s not enabling x, there should be a place p which does not have enough tokens to allow to perform x when reaching the state corresponding to s. That is, in addition to the constraints derived in the previous section, p should satisfy $M_s(p) < k + h$, i.e., using (2) with $r = s$:

$$\mu_0 \quad < \quad k + h + k \cdot \Delta_{s_0,s}(x) - \sum_{j \in I_0 \cup I_\ell} k_j \cdot \Delta_{s_0,s}(a_j) \tag{4}$$

Again, it is possible to reduce the number of such inequalities. For any $\alpha \in (T \setminus \{x\})^*$ with $s'[\alpha\rangle s$, we have $M_{s'}(p) \leq M_s(p)$. Hence, if (4) is ascertained for s, it is no longer necessary to bother about s'. Again, we may have cycles of equivalent states allowing to progress indefinitely while staying in states non-enabling x. Note that, by persistence, if $s \equiv_\ell s'$, then $s[x\rangle \iff s'[x\rangle$; moreover, if $s[a\rangle r$ with $r \not\equiv_\ell s$, then $s'[a\rangle r'$ with $r \equiv_\ell r'$; i.e., $\equiv_\ell$-equivalent states behave equivalently, as far as (non-) enabling of x is concerned. Hence, it makes sense to define $MNX(x) = \{s \in S \mid \neg s[x\rangle \text{ and } \forall s[a\rangle r\colon (s \equiv_\ell r) \vee r[x\rangle\}$ and

$$\boxed{mNX(x) \subseteq MNX(x) \text{ with a single representative of each } \equiv_\ell\text{-equivalence class in it}}$$

i.e., we consider states not allowing x such that no non-equivalent successor still excludes performing x, and we keep one representative of each class. Thus, if the event/state separation problems can be solved for the states s in $mNX(x)$, then they are solved for all $s \in S$ with $\neg s[x\rangle$. Hence, for every $s \in mNX(x)$, we need to find a place satisfying (3) and (4).

Combining the constraints (3) and (4) allows to eliminate both μ_0 and h:

$$\boxed{\forall r \in mXNX(x)\colon 0 < k \cdot [1 + \Delta_{s_0,s}(x) - \Delta_{s_0,r}(x)] + \textstyle\sum_{j \in I_0 \cup I_\ell} k_j \cdot [\Delta_{s_0,r}(a_j) - \Delta_{s_0,s}(a_j)]} \tag{5}$$

input a finite, decent, persistent lts $TS = (S, \to, T, s_0)$ satisfying $\mathbf{P}\{\Upsilon_1, \ldots, \Upsilon_n\}$;
initially T is the set of transitions, and P is empty;
for every $x \in T_\ell$ $(0 \le \ell \le n)$ and $s \in mNX(x)$ **do**
 construct a set $mXNX(x)$ and the corresponding system (5/6);
 if there is no natural solution to this system **then**
  {**output** "TS is not CF-solvable, due to x, s, and system (5/6)"; **stop**};
 choose a set of natural numbers $(k, k_1, \ldots, k_m)$ satisfying (5), as well as (1) if $\ell \neq 0$,
  and compute h and μ_0;
 add to P a place as in Fig. 5, with weights $k_1, \ldots, k_m, k+h, h$, and initial marking μ_0;
end for
output "The net with transitions T and places P CF-solves TS".

Figure 6 An algorithm checking CF-solvability and constructing an adequate solution.

If the system (5) is solvable in the domain of natural numbers (with $k_j = 0$ if $j \notin I_0 \cup I_\ell$), let us define $\mu = \max\{k \cdot \Delta_{s_0,r}(x) - \sum_{j \in I_0 \cup I_\ell} k_j \cdot \Delta_{s_0,r}(a_j) \mid r \in mXNX(x)\}$. If $\mu \ge 0$, by choosing $h = 0$ and $\mu_0 = \mu$ we shall get a solution to the systems (3) and (4), with $\mu_0 \ge 0$. If $\mu < 0$, it is not possible to create a suitable pure place from this solution, but we may choose $h = -\mu$ and $\mu_0 = 0$ (realising the "adequate shift" referred to above), and we shall again get a solution to the systems (3) and (4), with $\mu_0 \ge 0$.

If $x \notin T_0$, the constraints (1) need to be fulfilled as well. Combining (1) and (5), we get

$$\begin{array}{l}\forall r \in mXNX(x):\\ 0 < \sum_{j \in I_0 \cup I_\ell} k_j \cdot [\Upsilon_\ell(a_j) \cdot (1 + \Delta_{s_0,s}(x) - \Delta_{s_0,r}(x)) - \Upsilon_\ell(x) \cdot (\Delta_{s_0,s}(a_j) - \Delta_{s_0,r}(a_j))]\end{array} \quad (6)$$

If the system (6) is solvable in the domain $\mathbb{N}$, it is also possible to find a natural solution to both (1) and (5), by choosing a suitable value for k using (1), and, if necessary, multiplying the solution found by a common factor. Then we may choose h and μ_0 as described above.

Figure 6 summarises the resulting algorithm, where by "system (5/6)" we mean "system (5)" if $x \in T_0$ and "system (6)" if $x \in T_\ell$ for $1 \le \ell \le n$.

▶ **Theorem 4.3** (Validity of the construction). *If, for some $x \in T$ and $s \in mNX(x)$, the corresponding system (5/6) is not solvable, then TS has no CF solution. Otherwise, the constructed net is a CF solution of TS.*

Proof. If the system (5/6) associated with some $x \in T$ and $s \in mNX(x)$ is not solvable, then from the analysis above there is no place of a CF net both allowing all valid evolutions and excluding the invalid transition x from s. Let us thus assume all those systems are solvable and let us consider the net constructed as above.

We have seen that for any $s \in S$, if $s[x\rangle$, there is a state $r \in mXNX(x)$ and a state $s' \in XNX(x)$ such that $r[\alpha\rangle s'$ and $s[x^\ell\rangle s'$, with $\ell > 0$ and $\alpha \in (T \setminus \{x\})^*$. By the construction of each place p, $M_r(p) \ge h$, $M_{s'}(p) \ge h$, and $M_s(p) \ge k + h$. We have also seen that, for any $s \in S$, there is an x-free sequence α such that either $s_0[\alpha\rangle s$ and $M_s(p) \ge M_0(p) \ge 0$, or there is some $r \in XNX(x)$ with $r[\alpha\rangle s$ so that $M_s(p) \ge M_r(p) \ge h$. As a consequence, place p allows all valid evolutions specified by the lts. If $\neg s'[x\rangle$, we know there is some $s \in mNX(x)$ such that $M_{s'}(p) \le M_s(p)$. From the choice of $M_0(p)$ for the corresponding place p, and from (5), we have (4) whatever h, which excludes to perform x from s as well as from s'.

Thus, the solvability of all systems (5/6) implies that all event/state separation problems can be solved, and we get a CF net N with the same language as TS. The only way to

have non-isomorphism is that some state separation problem cannot be solved, i.e., that two states s_1 and s_2 correspond to the same marking. In that case, let $s_1[\beta\rangle q_1$ be a path to a home state q_1 of TS. Since s_1 and s_2 correspond to the same marking and $L(N) = L(TS)$, $s_2[\beta\rangle q_2$ for some state q_2 corresponding to the same marking as q_1. Since q_1 is a home state, there is a path $q_2[\alpha\rangle q_1$. Since the language is the same and q_1, q_2 correspond to the same marking, we have $q_2[\alpha\rangle q_1[\alpha\rangle q_3[\alpha\rangle q_4 \ldots$, and from finiteness and cycle consistency, we must have $q_1 = q_2$. But then, by strong determinism, we also have $s_1 = s_2$. ◀

Two examples

- Consider TS_3 (Figure 3). For $x = a$, we get $x \in T_0 = \{a, d\}$, $I_\ell = I_0 = \{1\}$ if $a_1 = d$, $mXNX(a) = \{s_1\}$, $mNX(a) = \{s_2\}$ and equation (5) reduces to $0 < k \cdot [1+1-1] + k_1 \cdot [0-1]$, which leads to the solution $k = 1$, $k_1 = 0$, $\mu_0 = 1$ and $h = 0$, corresponding to place p_1 in N_3. For $x = b$, we get $x \in T_1 = \{b\}$, $T_0 = \{a, d\}$, $I_0 = \{1, 2\}$ if $a_1 = a$ and $a_2 = d$, $I_1 = \emptyset$, $mXNX(b) = \{s_1\}$, $mNX(b) = \{s_0\}$ and equation (6) reduces to $0 < k_1 \cdot [0 - 1 \cdot (0-1)] + k_2 \cdot [0 - 1 \cdot (0-0)]$, which leads to the solution $k_1 = 1$, $k = k_2 = 0$, $\mu_0 = 0$ and $h = 1$, corresponding to place p_2 in N_3. The treatments of c and d are similar.
- Consider TS_4 (Figure 4). For $x = a$, we get $x \in T_0 = \{a\}$, $I_\ell = I_0 = \emptyset$, $MXNX(a) = \{s_4, s_5, s_6\}$ (all states are $\equiv_0$-equivalent), $mXNX(a) = \{s_4\}$ (for example), $MNX(a) = \{s_0, s_4, s_5, s_6, s\}$, $mNX(a) = \{s_0, s\}$ (for example), and for $x = s_0$, equation (5) reduces to $0 < k \cdot [1 + 0 - 1] + 0$, which is unsolvable.

Some special cases

- All transitions are live. Then $T_0 = \emptyset$; $\equiv_0$ is identity; and any $a_j \in {}^\bullet p$ corresponds to the same small cycle as $x \in p^\bullet$. If synthesis succeeds, the resulting CF net is a disjoint composition of n individual CF nets, each one connected by itself and corresponding to one of the Parikh vectors Υ_i (for $1 \leq i \leq n$). In essence, therefore, this reduces to the next case.
- All transitions are live and $n = 1$ (cf. [9]). Then both $\equiv_0$ and $\equiv_1$ reduce to identity. All sets $mXNX(x)$ and $mNX(x)$ are unique and can be simplified as follows:

$$\begin{array}{lcl} XNX(x) & = & \{s \in S \mid [x\rangle s \wedge \neg s[x\rangle\} \\ mXNX(x) & = & \{s \in XNX(x) \mid \nexists s' \in XNX(x) : s'[\alpha\rangle s \text{ with } \alpha \in (T \setminus \{x\})^+\} \\ mNX(x) & = & \{s \in S \mid \neg s[x\rangle \wedge \forall s' \in S, a \in T : s[a\rangle s' \Rightarrow s'[x\rangle\} \end{array}$$

  Such transition systems are "almost reversible" (i.e., they consist of an initial acyclic part, followed by a single strongly connected component, such as TS_2 in Figure 2).
- TS is reversible, i.e., s_0 is a home state (cf. [6, 8]). All previous simplifications hold and, additionally, if there is a CF solution, then there is also a pure solution. Nevertheless (despite the simpler setting), it is possible to construct reversible persistent transition systems which can be solved by a plain and pure Petri net, but not by a CF net.
- TS belongs to a marked graph (a plain net with $\forall p \in P : |p^\bullet| = 1 \wedge |{}^\bullet p| = 1$) or to a T-system (a plain net with $\forall p \in P : |p^\bullet| \leq 1 \wedge |{}^\bullet p| \leq 1$) (cf. [5, 7]). Such transition systems have full characterisations for bounded as well as for unbounded Petri nets.

5 Concluding remarks

The first part of this paper describes a partial characterisation – more precisely, an upper approximation – of the state spaces of bounded choice-free Petri nets. The reachability graphs of such Petri nets are finite, totally reachable, strongly deterministic, strongly cycle-consistent,

persistent, and enjoy the small cycle, the prime cycle, and the distance properties. A full structural characterisation seems to be very difficult to obtain. Even for finite words, such as TS_0 in Figure 2, it is very difficult to obtain exact structural (i.e., so to speak, free of linear algebra) conditions characterising the PN-solvable ones amongst them [3].

In its second part (starting with Section 4), the paper describes how choice-freely solvable transition systems can be detected and their Petri net solutions be constructed if possible. The aim here was to use structural knowledge in order to limit the set of states for which event/state separation problems need to be solved, and also to reduce the number of linear inequalities needed for each one of these problems.

The algorithmic gains are threefold: (1) It is possible to check, before starting synthesis, some of the properties given by the upper state space characterisation and to discard any given lts failing to satisfy one of them as not being solvable choice-freely. Such an algorithm has already been included for marked graphs in APT [10] and performs very satisfactorily. In general, though, these properties may not be easy to check. (2) State separation problems all but disappear (though this came as no surprise, given the result described in [13]). (3) The number of unknowns and the number of systems is reduced for the event/state separation problems that remain. This allows more efficient synthesis, and our first experiments confirm that our algorithm exhibits interesting performances. However, even for the special cases discussed at the end of Section 4, it seems difficult to estimate exactly how much can be gained, and in particular how the size of the essential sets $mXNX(x)$ and $mNX(x)$ evolves with the size of the lts, and possibly with some of its specific characteristics (like the out- and in-degrees of its nodes, symmetries of the structure, etc.).

There are many extensions and potential applications of choice-free synthesis. One particularly promising generalisation is to allow partially non-injective transition labellings for Petri nets (for instance, by forbidding equally labelled transitions in parallel components).

Acknowledgements. We are indebted to the reviewers for valuable comments.

References

1 Eric Badouel, Benoît Caillaud, and Philippe Darondeau. Distributing finite automata through Petri net synthesis. *Formal Asp. Comput.*, 13(6):447–470, 2002.

2 Eric Badouel and Philippe Darondeau. Theory of regions. In *Lectures on Petri Nets I: Basic Models, LNCS Vol. 1491*, pages 529–586, 1998.

3 Kamila Barylska, Eike Best, Evgeny Erofeev, Łukasz Mikulski, and Marcin Piątkowski. On binary words being Petri net solvable. In *Algorithms and Theories for the Analysis of Event Data (ATAED 2015)*, pages 1–15, 2015.

4 Eike Best and Philippe Darondeau. A decomposition theorem for finite persistent transition systems. *Acta Inf.*, 46(3):237–254, 2009.

5 Eike Best and Raymond R. Devillers. Characterisation of the state spaces of live and bounded marked graph Petri nets. In *8th International Conference on Language and Automata Theory and Applications (LATA 2014)*, pages 161–172, 2014.

6 Eike Best and Raymond R. Devillers. Synthesis of persistent systems. In *35th International Conference on Application and Theory of Petri Nets and Concurrency (ICATPN 2014)*, pages 111–129, 2014.

7 Eike Best and Raymond R. Devillers. State space axioms for T-systems. *Acta Inf.*, 52(2-3):133–152, 2015.

8 Eike Best and Raymond R. Devillers. Synthesis and reengineering of persistent systems. *Acta Inf.*, 52(1):35–60, 2015.

9 Eike Best and Raymond R. Devillers. Synthesis of live and bounded persistent systems. *Fundamenta Informaticae*, 140:39–59, 2015.

10 Eike Best and Uli Schlachter. Analysis of Petri nets and transition systems. In *Proc. 8th Interaction and Concurrency Experience (ICE), Grenoble*, 2015.

11 Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alex Yakovlev. *Logic Synthesis for Asynchronous Controllers and Interfaces*, volume 8 of *Advanced Microelectronics*. Springer, 2002.

12 Stefano Crespi-Reghizzi and Dino Mandrioli. A decidability theorem for a class of vector-addition systems. *Inf. Process. Lett.*, 3(3):78–80, 1975.

13 Philippe Darondeau. Equality of languages coincides with isomorphism of reachable state graphs for bounded and persistent Petri nets. *Inf. Process. Lett.*, 94(6):241–245, 2005.

14 Paul Gastin and Nathalie Sznajder. Fair synthesis for asynchronous distributed systems. *ACM Trans. Comput. Log.*, 14(2):9, 2013.

15 Robert M. Keller. A fundamental theorem of asynchronous parallel computation. In *Sagamore Computer Conference, August 20-23 1974, LNCS Vol. 24*, pages 102–112, 1975.

16 Leslie Lamport. Arbitration-free synchronization. *Distributed Computing*, 16(2-3):219–237, 2003.

17 Lawrence H. Landweber and Edward L. Robertson. Properties of conflict-free and persistent Petri nets. *J. ACM*, 25(3):352–364, 1978.

18 Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

19 Peter J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

20 Enrique Teruel, José Manuel Colom, and Manuel Silva. Choice-free Petri nets: a model for deterministic concurrent systems with bulk services and arrivals. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 27(1):73–83, 1997.

21 Harro Wimmel and Karsten Wolf. Applying CEGAR to the Petri net state equation. In *17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2011)*, pages 224–238, 2011.

Polynomial Time Decidability of Weighted Synchronization under Partial Observability

Jan Křetínský[1], Kim Guldstrand Larsen[2], Simon Laursen[2], and Jiří Srba[2]

1 **IST Austria, Austria**
2 **Aalborg University, Department of Computer Science, Denmark**

Abstract

We consider weighted automata with both positive and negative integer weights on edges and study the problem of synchronization using adaptive strategies that may only observe whether the current weight-level is negative or nonnegative. We show that the synchronization problem is decidable in polynomial time for deterministic weighted automata.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases weighted automata, partial observability, synchronization, complexity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.142

1 Introduction

The problem of synchronizing automata [4, 13, 14] studies the following natural question: *"how can we gain control over a device when its current state is unknown?"* Synchronizing automata have classically been studied in the setting of deterministic finite automata (DFA), aiming at finding short synchronizing words, i.e. finite sequences of input symbols that will bring the automaton from any (unknown) state into a unique state. Here the existence of a synchronizing word is NLOGSPACE-complete [4, 14], and polynomial bounds were given on the length of the shortest synchronizing word. Yet, establishing a tight bound on the length of the shortest synchronizing word has been an open problem for the last 50 years, with Černý [4] conjecturing that words of length at most $(n-1)^2$, where n is the number of states in the DFA, are sufficient.

We consider synchronization of deterministic weighted automata (WA), where their states are composed of locations and integer weights, and where transitions have their associated weights from $\mathbb{Z}$. In this setting, weights are simply accumulated during the run of the system, and thus it is impossible to find a word that will ensure synchronization to a single state: for any two states with identical locations but different weights, e.g. (ℓ, z) and $(\ell, z+1)$, any word will – by the assumption of determinism – maintain the relative difference in their weights. We therefore assume that during the synchronization, the controller has some (minimal) information available concerning the current weight of the system; in particular, we assume that the controller is able to observe whether the current weight is negative or nonnegative. Under this assumption, a solution to the synchronization problem becomes an *adaptive* strategy, in the sense that the next input to be selected may be based on the previous weight-observations made by the controller.

Our main result is that the existence of a synchronizing strategy, using only observations of the sign of the current weight-level, is decidable in polynomial time for deterministic WA. This result relies on a polynomial time algorithm for detecting cycles of weight $+1$ and -1 in a given weighted graph.

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 142–154

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

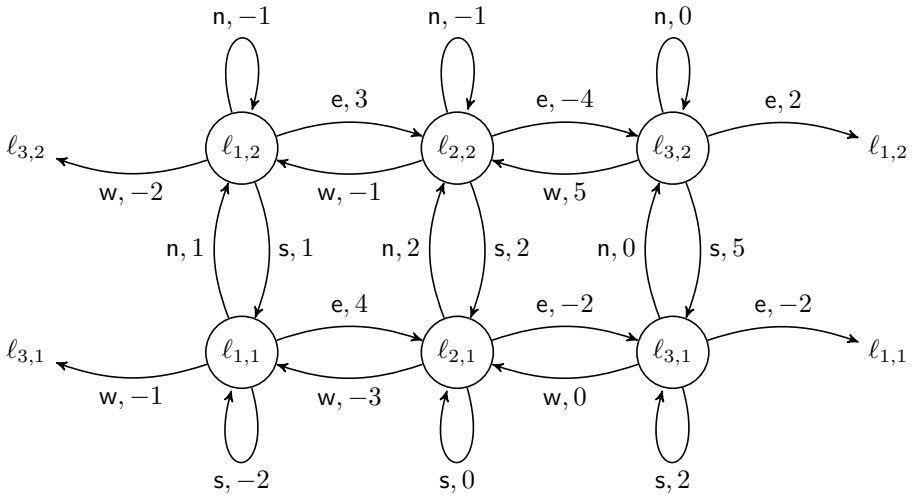


Figure 1 **B**lind **P**ackman with moves north (n), south (s), east (e) and west (w).

Fig. 1 illustrates BP (Blind Packman), a WA with 6 locations and 4 actions. We have to find a strategy that will (under partial observability of weights) synchronize infinitely many states of the form $(\ell_{i,j}, z)$ where $\ell_{i,j}$ is one of the 6 locations and $z \in \mathbb{Z}$ is the starting weight. First, we note that after an n-input, BP will be in one of the 3 (top-row) locations $\ell_{i,2}$ for $i = 1, 2, 3$. Given the cyclic, horizontal structure, it is also clear that no further sequence of inputs from the set $\{\mathsf{n}, \mathsf{e}, \mathsf{s}, \mathsf{w}\}$ can provide any additional information in which of the three locations we are located. However, assuming the weight-level is observed to be nonnegative (a similar case applies if the weight-level is observed to be negative), we may infer that BP is in a state of the form $(\ell_{i,2}, z)$ with $z \geq 0$. Noting the -1-loops from $\ell_{1,2}$ and $\ell_{2,2}$, it is tempting to repeatedly offer n as input until the weight-level becomes negative. However, the presence of the 0-loop at $\ell_{3,2}$ makes it possible that such a strategy will not terminate. Instead, we observe that the input-word $(\mathsf{n} \cdot \mathsf{e})^3$ will constitute a (composite) cycle in BP that makes the weight-level drop exactly by -1, regardless as to from which of the three top-locations the word was executed. Thus, by repeating this input-word while constantly observing the sign of the weight-level and terminating as soon as the weight-level becomes negative, we are able to infer that the BP automaton is either in the location $\ell_{3,2}$ in case the observation changed after the input e, or in one of the states $(\ell_{1,2}, -1)$ or $(\ell_{2,2}, -1)$ if the change happened after the input n. In the former case, we exercise the cycle e^3 until a change in observation brings us to $(\ell_{3,2}, 0)$. In the latter case, an e-input followed by a test of the weight-level will reveal the true identity of the state; using ± 1 cycles, it is now easy to reach $(\ell_{3,2}, 0)$, thus completing the synchronization.

As illustrated by the above example, the presence of cycles with weights $+1$ and -1 is essential for the synchronization under partial observability. As we shall demonstrate, the existence of such cycles is decidable in polynomial time, and constitutes, with a few other polynomial time checks, a necessary and sufficient condition for the synchronization of a WA.

Related work

Survey of results and applications for classical synchronizing words may be found in [13, 14]. Recently, there has been an increasing interest in novel extensions of the synchronization problem. Volkov [9] studied synchronization games and priced synchronization on weighted

automata with positive weights and finds a synchronizing word where the worst accumulated cost is below a given bound. In [6, 7, 8], (infinite) synchronizing words were studied in the probabilistic settings. Synchronization of weighted timed automata was studied in [5], where location synchronization with safety conditions on the weight was considered, though without the requirement on the weight synchronization. Finally, synchronization under partial observability was recently studied in [12] but only in the context of finite automata without weights.

2 Definitions

We shall now formally define the synchronization problem on deterministic and complete weighted automata.

▶ **Definition 1** (Weighted Automaton). A (deterministic) *weighted automaton (WA)* is a tuple $\mathcal{A} = (L, Act, T, W)$ where

- L is a finite set of locations,
- Act is a finite set of actions,
- $T : L \times Act \to L$ is a transition function, and
- $W : L \times Act \to \mathbb{Z}$ is a weight function.

A *state* of $\mathcal{A}$ is a pair $(\ell, z) \in L \times \mathbb{Z}$ where ℓ is the current location and z the current weight. Let $S(\mathcal{A})$ be the set of all states of $\mathcal{A}$. We write $(\ell, z) \xrightarrow{a,w} (\ell', z')$ if $T(\ell, a) = \ell'$, $W(\ell, a) = w$ and $z' = z + w$.

A *path* in $\mathcal{A}$ is a finite sequence of states $\pi = s_0 s_1 \dots s_n$ such that for all i, $0 \leq i < n$, we have $s_i \xrightarrow{a_i,w_i} s_{i+1}$ for some $a_i \in Act$ and $w_i \in \mathbb{Z}$. The last state s_n in the path π is referred to as $last(\pi)$. The set of all paths is denoted by $Paths(\mathcal{A})$. For the complexity analysis in the rest of this paper, we assume a binary encoding of integers in $\mathcal{A}$.

▶ **Definition 2** (Observation Function). An *observation function* $\gamma : S(\mathcal{A}) \to \mathcal{O}$ maps each state of $\mathcal{A}$ to an observation from an observations set $\mathcal{O}$.

Assume now a given observation function γ to the set of observations $\mathcal{O}$. Let $\pi = s_0 s_1 \dots s_n$ be a path in $\mathcal{A}$. The observation function γ is naturally extended to an *observation sequence* for π by

$$\gamma(\pi) = \gamma(s_1)\gamma(s_2)\dots\gamma(s_n) \ .$$

▶ **Definition 3** (Strategy). A *strategy* is a function $\delta : \mathcal{O}^+ \to Act \cup \{\mathsf{done}\}$ that maps a nonempty sequence of observations to a proposed action or the symbol $\mathsf{done} \notin Act$, signaling that no further actions will be proposed.

A path $\pi = s_0 s_1 \dots s_n$ *follows* a strategy δ if $s_i \xrightarrow{a_i,w_i} s_{i+1}$ for $a_i = \delta(\gamma(s_0 s_1 \dots s_i))$ and $w_i \in \mathbb{Z}$, for all i, $0 \leq i < n$. A strategy δ is *terminating* if it does not generate any infinite path, in other words there is no infinite sequence where all its finite prefixes follow δ.

Given a subset of states $X \subseteq S(\mathcal{A})$ and a terminating strategy δ, the set of all maximal paths that follow the strategy δ in $\mathcal{A}$ and start from some state in X, denoted by $\delta[X]$, is defined as follows:

$$\delta[X] = \{\pi = s_0 s_1 \dots s_n \in Paths(\mathcal{A}) \mid s_0 \in X, \pi \text{ follows } \delta \text{ and } \delta(\gamma(\pi)) = \mathsf{done}\} \ .$$

The set of final states reached when following δ starting from X is defined as $last(\delta[X]) = \{last(\pi) \mid \pi \in \delta[X]\}$. Assuming a given observation function γ, we can now define a synchronizing strategy that will bring the system from any unknown initial state to the same single synchronizing state.

▶ **Definition 4** (Synchronization). Given a WA $\mathcal{A}$, a strategy δ is *synchronizing* if δ is terminating and $|last(\delta[S(\mathcal{A})])| = 1$. Further, $\mathcal{A}$ is *synchronizable* if it admits a synchronizing strategy.

We limit our study to systems where we see no information about the current location and have a partial observability of the current weight so that we can distinguish whether its value is negative or nonnegative. This is the minimal possible observation as if we cannot observe anything about the weight then synchronization is impossible. Hence, we define the observation function γ to the set of observations $\mathcal{O} = \{<0, \geq 0\}$ by

$$\gamma((\ell, z)) = \begin{cases} <0 & \text{if } z < 0 \\ \geq 0 & \text{if } z \geq 0 \end{cases}.$$

We are interested in deciding whether a given WA is synchronizable under this observation function γ.

3 Polynomial Time Algorithm for Synchronizing

Let $\mathcal{A} = (L, Act, T, W)$ be a WA. We write $\ell \xrightarrow{a,w} \ell'$ for $\ell, \ell' \in L$ whenever $T(\ell, a) = \ell'$ and $w = W(\ell, a)$. A *cycle* in $\mathcal{A}$ starting in ℓ_0 is a path of the form $\ell_0 \xrightarrow{a_0,w_0} \ell_1 \xrightarrow{a_1,w_1} \dots \ell_n \xrightarrow{a_n,w_n} \ell_0$. The *weight* of the cycle is $\sum_{i=0}^{n} w_i$.

We now perform a series of checks in order to test whether we can synchronize from any possible initial state. The tests will give a necessary and sufficient condition for synchronization. At the end, we will argue that all the checks can be done in polynomial time. Therefore, we provide a polynomial time algorithm for deciding synchronizability. Furthermore, if all the checks succeed, we also construct a synchronizing strategy. It works in several phases, each concerning some of the tests. However, we note that although our algorithm decides synchronizability in polynomial time, the construction of a synchronizing strategy as an explicit function is not possible due to the fact that the lengths of the synchronizing sequences proposed by the strategy are unbounded (they depend on the initial weight values). We instead provide an algorithm, describing the unbounded strategy from any given initial state.

First, we check if the given $\mathcal{A}$, viewed as a labelled directed graph, has the following property.

Property 1. The graph $\mathcal{A}$ has a strongly connected component that is reachable from any location in $\mathcal{A}$.

If Property 1 is not satisfied, and such a bottom strongly connected component does not exist, clearly there is no synchronizing strategy for $\mathcal{A}$. From now on, assume that Property 1 holds. Taking advantage of this property, we define the first phase of the constructed synchronizing strategy δ.

▷ **Phase 1.** For every location ℓ let $home(\ell)$ be a sequence of actions that will bring ℓ into this strongly connected component and for any sequence of actions x let $\ell[x]$ be the location that we will reach from ℓ after performing the sequence x (note that this is well defined due to the fact that $\mathcal{A}$ is deterministic). Our synchronizing strategy will start by performing the action sequence $x_1 x_2 x_3 \dots x_n$ where

$$x_1 = home(\ell_1), x_2 = home(\ell_2[x_1]), x_3 = home(\ell_3[x_1 x_2]), \dots, x_n = home(\ell_n[x_1 x_2 \dots x_{n-1}])$$

assuming that $L = \{\ell_1, \ell_2, \ldots, \ell_n\}$. We shall refer to this technique as *sequentialization*: intuitively, even if the initial location is unknown, we can perform given actions for each possible initial location, meanwhile tracking where we move if the actual initial location was different, and execute these steps in a sequence for each possible location. ◁

▶ **Example 1.** We illustrate Phase 1 on the system below in Figure 2. We first execute $home(\ell_1) = a$. Meanwhile both ℓ_2 and ℓ_3 move to ℓ_2. Therefore, we proceed with $home(\ell_2) = ba$. Therefore, if we started in ℓ_3 we are now in ℓ_4, too. Since ℓ_4 is in a bottom strongly connected component, $home(\ell_4)$ is the empty sequence and we are done.

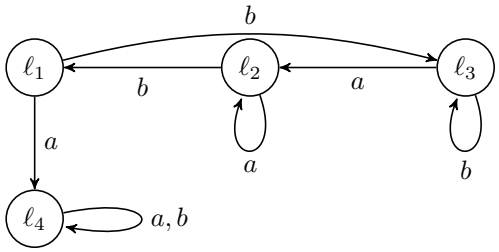


Figure 2 Example of sequentialization (the word *aba* will bring all locations to ℓ_4).

Consequently, after Phase 1, we are for sure in the strongly connected component. Within this component, we check the following property.

Property 2. Let $\mathcal{A}$ be strongly connected. In $\mathcal{A}$, there is a cycle with weight 1 and a cycle with weight -1.

▶ **Lemma 5.** *If Property 2 is not satisfied then there is no synchronizing strategy.*

Proof. Assume that $\mathcal{A}$ is synchronizable. Then there must be a positive and a negative cycle in $\mathcal{A}$. Further, for any location ℓ, the states $(\ell, 0)$ and $(\ell, 1)$ can be synchronized. Therefore, there is a path from both these states to some state (ℓ', z) for some $z \in \mathbb{Z}$. Since $\mathcal{A}$ is strongly connected, there is also a path from (ℓ', z) back to the state (ℓ, z') for some $z' \in \mathbb{Z}$. Consequently, there are two cycles from ℓ with weights z' and $z' - 1$, respectively; moreover, these weights can be chosen non-zero due to existence of a positive cycle. Hence the weights of these two cycles are relative primes and in combination with the presence of a positive and negative cycle in $\mathcal{A}$, this implies the existence of cycles with the weights $+1$ and -1. ◀

From now on, assume that $\mathcal{A}$ is strongly connected and Property 2 holds. Observe that, consequently, there are $+1$ and -1 cycles starting and ending in each location ℓ. Let us denote the corresponding sequences of actions by ℓ^+ and ℓ^-. The first, and rather naive, use of these cycles is to get the weight component of the state close to zero.

▷ **Phase 2.** We extend our strategy δ by performing the ± 1 cycles until we see a change in our observation. Assuming we start with nonnegative observation, Phase 2 ends at the moment when a negative observation is reached (and symmetrically for the other case). To this end, assuming $L = \{\ell_1, \ldots, \ell_n\}$, we employ the sequentialization technique again. We first execute the word ℓ_1^- for the -1 cycle from ℓ_1 and keep track of the resulting locations $\{\ell'_1, \ldots, \ell'_n\}$. Note that their weights could have increased instead, say by at most c_1. Next we execute ${\ell'_2}^-$ exactly $(c_1 + 1)$-times, so that even if the initial location was ℓ_2, after this

many cycles the weight decreased no matter how it increased by performing ℓ_1^-. Meanwhile ℓ_3' changes to ℓ_3'' and its weight could have in total increased by at most c_2. We thus execute $\ell_3''^-$ exactly (c_2+1)-times and so on for all locations cyclically (starting again at the first location once we went through all of them) until the weight decreases below zero. This process terminates since whenever performing cycles for a particular location, its weight (if we indeed started in the respective location) drops below any previous value. ◁

▶ **Example 2.** We illustrate Phase 2 on the system below in Figure 3 when the observation is nonnegative. We first execute $\ell^- = a$. Meanwhile r loops under a and increases by $c_1 = 3$. Therefore, we proceed with repeating $r^- = bab$ for 4 times. This in turn makes ℓ return again to ℓ with value increased by $c_2 = 4 \cdot 3 = 12$. Next we repeat ℓ^- for 13 times etc.

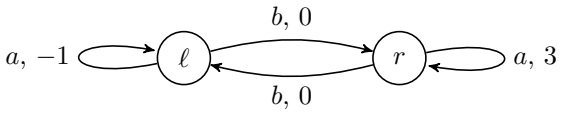


Figure 3 Example illustrating Phase 2.

We are now guaranteed that right after Phase 2, the observation has just changed. Therefore, we are now in a state (ℓ, z) for some $\ell \in L$ and

$$0 \leq z < M \text{ if } \gamma((\ell, z)) = \geq 0 \quad \text{or} \quad -M \leq z < 0 \text{ if } \gamma((\ell, z)) = <0$$

where M is the largest absolute value of any weight used in $\mathcal{A}$. Therefore, there are finitely many states we can be at. Now that we have a bound how far we are from zero, we can make a better use of ± 1 cycles and derive for each location, where we possibly might be at, its weight.

▷ **Phase 3.** Once again, we employ sequentialization. Assuming first that we are in location $\ell_1 \in L$ of a state (ℓ_1, z) with $-M \leq z < M$, we can perform a sequence of actions corresponding to -1 or $+1$ cycle from ℓ_1 (depending on whether $\gamma((\ell_1, z)) = \geq 0$ or $\gamma((\ell_1, z)) = <0$) until the observation changes at the end of the cycle when we are again in ℓ_1. If we indeed started in the location ℓ_1, we know that we are now in the state $(\ell_1, -1)$ if $\gamma((\ell_1, z)) = \geq 0$, or $(\ell_1, 0)$ if $\gamma((\ell_1, z)) = <0$. If the weight in the reached states did not change from nonnegative to negative (or the other way round) even after performing M cycles, we know for sure that we were not in the location ℓ_1. The situation where we started in a different location than ℓ_1 and the weight observation still changed as expected simply adds an extra (false) hypothesis that can be eliminated (as shown later on in this section).

Now consider what would happen, until now, if we were instead in location $\ell_2 \in L$ at the moment before we started to perform the -1 or $+1$ cycles for ℓ_1. After playing according to the strategy above, we would be now in a possibly different location ℓ_2' with weight in the range $[-M', M']$ where M' can be computed from M and the strategy performed so far. We can now start performing -1 or $+1$ cycle from ℓ_2' exactly as before in order to determine the exact weight in this location (provided we started in ℓ_2) and we continue like this with handling ℓ_3 etc., for each location in L. ◁

▶ **Example 3.** We illustrate Phase 3 on the system below in Figure 4. If the observation is nonnegative, the current weight is at most 2 and we start with repeating $bbaa$, a -1 cycle

for ℓ, for at most $M = 3$ times. If the observation remains nonnegative after this sequence (case 1) we must be in r. Otherwise (case 2), we stop when the observation changes and if we are in ℓ the current weight is -1. Meanwhile r returned back to r and could have increased its weight to at most 5 in case 2. Then we proceed with repeating $bbaa$ at most 6 times to get for sure to $(r, -1)$. In case 1, the observation is negative and the weight is at least -2. Hence, we repeat aab, a $+1$ cycle for r. Say that the observation changes after two repetitions. Then we are either in $(r, 0)$ or in the meanwhile achieved $(\ell, -3)$. (The latter is, however, impossible here since the observation would remain negative.)

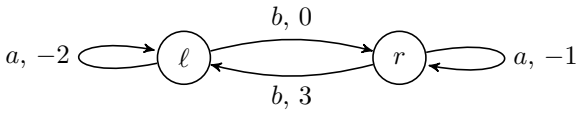


Figure 4 Example illustrating Phase 3.

We conclude that after Phase 3 we must be in one of the states from the set

$$\{(\ell_1, z_1), (\ell_2, z_2), \ldots, (\ell_n, z_n)\}$$

called the *hypothesis set*, where all z_i's are exactly known. We can w.l.o.g. assume that all locations in the assumption are pairwise different. Indeed, we can perform a number of ± 1 cycles from the location that appears in the hypothesis set more times and determine which one of the weights is still feasible (at most one is). Note that the size of the hypothesis set is thus at most $|L|$.

The next task is to distinguish between these hypotheses. For each pair of locations, assuming their weights from the hypothesis set, there must be a way to synchronize them. We present three tests such that at least one of them must be passed by each pair. All tests refer to the following notion of difference graph.

▶ **Definition 6** (Difference Graph)**.** The *difference graph* of a WA $\mathcal{A}$ is a weighted graph $G^{\mathcal{A}} = (V, E)$ with $E \subseteq V \times \mathbb{Z} \times V$ such that $V = L \times L$, and for every $a \in Act$ we have $\big((\ell, \ell'), W(\ell, a) - W(\ell', a), (T(\ell, a), T(\ell', a))\big) \in E$.

In other words, $G^{\mathcal{A}}$ is a synchronous product of two $\mathcal{A}$'s, where each edge weight is the difference of edge weights in the first and the second component.

▶ **Example 4.** Consider the system on the upper part of Figure 5, parametrized by $k \in \mathbb{Z}$. We depict a part of its difference graph on the lower part of the figure.

We have already seen how to distinguish states with the same location using ± 1 cycles. For all pairs of locations (ℓ_1, ℓ_2) where $\ell_1 \neq \ell_2$, we run the following three tests.

Property 3. There is a path in $G^{\mathcal{A}}$ from (ℓ_1, ℓ_2) to (ℓ, ℓ) for some $\ell \in L$.

Property 4. There is a path in $G^{\mathcal{A}}$ from (ℓ_1, ℓ_2) to (ℓ_1', ℓ_2') such that there is a cycle of nonzero weight (positive or negative) in $G^{\mathcal{A}}$ starting in (ℓ_1', ℓ_2').

In order to define the last Property 5, we need additional notions and reasoning. If Property 4 is not satisfied for a given pair (ℓ_1, ℓ_2) then every cycle in $G^{\mathcal{A}}$ reachable from (ℓ_1, ℓ_2) has zero weight. Therefore, whenever any cycle C in $G^{\mathcal{A}}$ is performed in $\mathcal{A}$ starting from location ℓ_1 or ℓ_2, the weight changes in both cases by the same value, called the *projected*

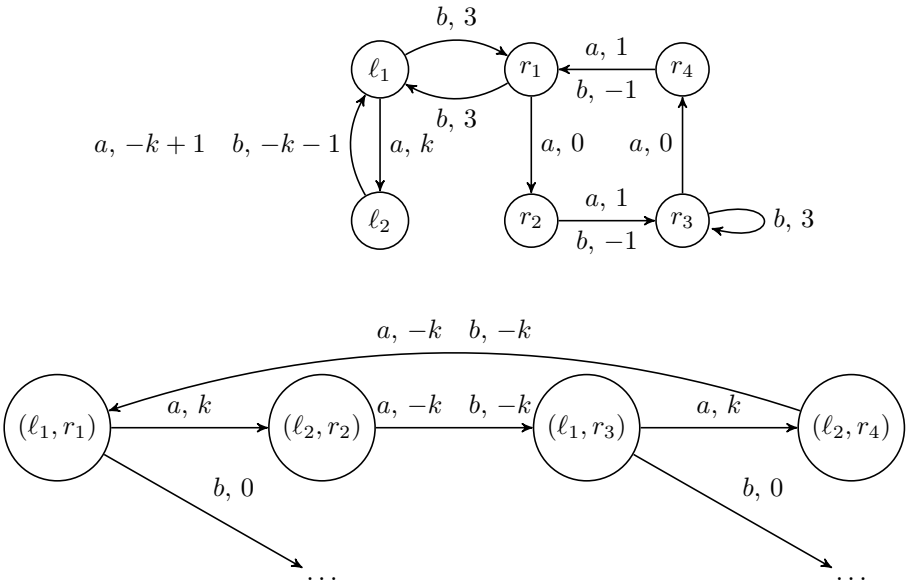


Figure 5 Example of difference graph.

weight $\mathcal{P}(C)$ of C. Although $G^{\mathcal{A}}$ may be disconnected, we may w.l.o.g. assume that (ℓ_1, ℓ_2) is a node in $G^{\mathcal{A}}$ that is a part of some strongly connected component (otherwise, we bring it there, using sequentialization).

▶ **Lemma 7.** *For every strongly connected component S of $G^{\mathcal{A}}$ satisfying Property 2 and not satisfying Property 4, there is a number p such that $1 \leq p \leq |L|$ and for any node (pair of locations) in S there are cycles C^+, C^- starting in this node with $\mathcal{P}(C^+) = p$ and $\mathcal{P}(C^-) = -p$. Moreover, such p can be computed in polynomial time.*

Proof. Let (ℓ_1, ℓ_2) be an arbitrary node in S. Due to Property 2, there is a cycle from ℓ_1 in $\mathcal{A}$ with weight $+1$. We repeatedly perform this $+1$ cycle and follow the same behaviour from (ℓ_1, ℓ_2) in $G^{\mathcal{A}}$. At the end of each cycle, the first component in $G^{\mathcal{A}}$ will be in the location ℓ_1 and the second component in one of the $|L|$ possible locations. By the pigeon-hole principle, after performing the $+1$ cycle at most $|L|$ times, we will find a repeated pair in $G^{\mathcal{A}}$. Hence we found a cycle C^+ in $G^{\mathcal{A}}$ with zero weight in $G^{\mathcal{A}}$, due to the violation of Property 4, and with the projected weight $0 < \mathcal{P}(C^+) \leq |L|$. By the same arguments, but using the fact about the existence of -1 cycle in $\mathcal{A}$, we can find a cycle C^- in $G^{\mathcal{A}}$ with the projected weight $0 > \mathcal{P}(C^-) \geq -|L|$.

Let us now argue that for each node we can choose cycles with absolute weights equal to a fixed integer. Let $p_{\min} := \min\{|\mathcal{P}(C)| \mid C \text{ is a cycle in } S\}$ denote the smallest projection over all cycles in the strongly connected component S. We claim that for any pair of states (ℓ_1, ℓ_2) in S, there are cycles in S starting in (ℓ_1, ℓ_2) with projected weights $p_{\min}$ and $-p_{\min}$. Indeed, note that there is a cycle C in S with $|\mathcal{P}(C)| = p_{\min}$ and there are cycles D^+ and D^- from (ℓ, ℓ') that visit some state of C and have positive and negative projected weight, respectively. Now by repeating C on the way either in D^+ or in D^-, we construct cycles E^+ and E^- with $0 < \mathcal{P}(E^+) \leq p_{\min}$ and $0 > \mathcal{P}(E^-) \geq -p_{\min}$, respectively. By minimality of $p_{\min}$, we obtain $\mathcal{P}(E^+) = p_{\min}$ and $\mathcal{P}(E^-) = -p_{\min}$. Note that, moreover, for each cycle C is S, $\mathcal{P}(C)$ is a multiple of $p_{\min}$. And vice versa, for each multiple of $p_{\min}$, there is a cycle with such projected weight in any node of S. Therefore, we can perform the pigeon-hole

construction of a cycle (in polynomial time), obtaining a weight $0 < p \leq |L|$, and we are guaranteed that from each node there are cycles with projected weights p and $-p$, respectively (although p is not necessarily minimal). ◀

Consequently, for each strongly connected component S, we have $0 < p \leq |L|$. For S, we define a reachability problem on a graph $\mathcal{A}_S = (V, \to)$ where

- $V = (L \times \{0, \ldots, p-1\}) \times (L \times \{0, \ldots, p-1\}) \cup \{\mathit{separated}\}$, and
- for each $v = \big((\ell_1, z_1), (\ell_2, z_2)\big)$ and $a \in \mathit{Act}$, let $v' = \big((\ell_1', z_1'), (\ell_2', z_2')\big)$ where $\ell_1' = T(\ell_1, a)$, $\ell_2' = T(\ell_2, a)$ and $z_1' = z_1 + W(\ell_1, a) + \alpha \cdot p$ and $z_2' = z_2 + W(\ell_2, a) + \alpha \cdot p$ for the unique $\alpha \in \mathbb{Z}$ such that the larger of z_1', z_2' lies in the interval $[0, p-1]$; we set
  - $v \to v'$ if $v' \in V$, i.e. the lower weight is also nonnegative, and
  - $v \to \mathit{separated}$, otherwise, i.e. the lower weight is negative.

We say that the graph $\mathcal{A}_S$ is *distinguishing* for a pair of locations $(\ell_1, \ell_2) \in S$ if from any initial node $\big((\ell_1, z_1), (\ell_2, z_2)\big)$, for each $0 \leq z_1, z_2 \leq p-1$, we can reach the node *separated*. Note that the size of $\mathcal{A}_S$ is at most $|L|^4 + 1$, hence polynomial in $|\mathcal{A}|$. Now we state the final test.

Property 5. If (ℓ_1, ℓ_2) belongs to a strongly connected component S of $G^{\mathcal{A}}$ then the graph $\mathcal{A}_S$ is distinguishing for (ℓ_1, ℓ_2).

▶ **Example 5.** Consider the difference graph of Figure 5. Observe that there is no path from (ℓ_1, r_1) to a pair with identical components, as well as no nonzero cycle. The length p is equal 2 here. If $k \geq 2$ then we have $((\ell_1, 0), (r_1, 0)) \to \mathit{separated}$ as action a immediately creates a large enough difference. If $k = 1$, then *separated* is still reachable from $((\ell_1, 0), (r_1, 0))$, but only after aa is taken. Then both weights are 1 and the next action a creates the distinguishing difference as the weights would now be $2, 1$, i.e. transformed to $0, -1$.

Supposing each pair of locations satisfies Property 3 or Property 4 or Property 5, we can iteratively decrease the size of the hypothesis set until it becomes a singleton as shown in the next phase.

▷ **Phase 4.** We employ sequentialization again. We pick any two states from the current hypothesis set and eliminate at least one of them as described below. Meanwhile, we update all remaining states from the hypothesis set to their current states. We repeat this procedure until the hypothesis set becomes a singleton. Let (ℓ_1, z_1) and (ℓ_2, z_2) be the currently explored pair from the hypothesis set.

First, if Property 3 holds we perform the sequence of actions that brings both locations into a single location. Afterwards, if their respective weights are different, using the ± 1 cycles, we detect at least one of the weights impossible as above. Thus we decrease the size of the hypothesis set.

Second, if Property 4 holds then we can extend our strategy δ by executing the sequence of actions that brings (ℓ_1, ℓ_2) to some (ℓ_1', ℓ_2') where we can repeatedly execute actions on the nonzero cycle in $G^{\mathcal{A}}$ until the weights in the pair of states reached after this sequence are sufficiently (see below) far away from each other. Assume w.l.o.g. that the weights z_1', z_2' of the two reached states are both positive and $z_1' < z_2'$ (the other situations are symmetric). Now from the location with the lower weight (ℓ_1'), we enter a simple cycle in $\mathcal{A}$ with the minimal (negative) weight and start executing it. This ensures that if we started from ℓ_1 or ℓ_2, then the observation will change to negative in n_1 or n_2 steps, respectively, where $n_1 < n_2$ (since $|z_2' - z_1'|$ was sufficiently large) and we can compute these numbers. If the observation changes after exactly n_1 steps, we eliminate the state corresponding to ℓ_2 from

the hypothesis set. If the observation changes after exactly n_2 steps, we eliminate the state corresponding to ℓ_1 from the hypothesis set. If the observation changes after a different number of steps, we eliminate both.

Third, let Property 5 hold (and Property 4 not) and (ℓ_1, ℓ_2) be in a strongly connected component S of $G^{\mathcal{A}}$. By Lemma 7, we have zero cycles in $\mathcal{A}_S$ with projected weights p and $-p$ where $0 < p \leq |L|$. We perform these cycles until the larger weight is in $[0, p-1]$. If the lower weight is negative at this moment, the current observation eliminates one of the hypotheses. Otherwise, the weights in both states are in $[0, p-1]$. Due to Property 5 we have a strategy to reach *separated* in $\mathcal{A}_S$, inducing a strategy in $\mathcal{A}$ by inserting the $-p$ and p cycles. Upon reaching *separated* in $\mathcal{A}_S$, the observation in $\mathcal{A}$ proves one of the two hypotheses impossible. (If at any moment throughout the process, an unexpected change of observation occurs, we eliminate the respective hypothesis from the set immediately.)

Once the hypothesis set is a singleton, we know precisely the current state. Finally, we deterministically reach a fixed location and fixed weight (by performing ± 1 cycles) and thus synchronize. ◁

The stated properties are not only sufficient, but also necessary conditions for synchronizability:

▶ **Lemma 8.** *Let $\mathcal{A}$ be a strongly connected WA satisfying Property 2. Then $\mathcal{A}$ is synchronizable if and only if for each pair of locations (ℓ_1, ℓ_2) either Property 3 or Property 4 or Property 5 is satisfied.*

Proof. The "if"-part follows from the previously constructed synchronizing strategy. For the "only-if"-part, assume that there is a pair (ℓ_1, ℓ_2) satisfying neither Property 3, nor Property 4, nor Property 5. By the last one, there are weights $z_1, z_2 \in [0, p-1]$ such that the node *separated* is not reachable from the configuration $init = \big((\ell_1, z_1), (\ell_2, z_2)\big)$ in the graph $\mathcal{A}_p$. For a contradiction, assume that $\mathcal{A}$ admits a synchronizing strategy σ. When σ is applied to initial states (ℓ_1, z_1) and (ℓ_2, z_2), we obtain two paths π_1 and π_2, inducing two sequences of observations $\gamma(\pi_1)$ and $\gamma(\pi_2)$. Comparing the respective elements in the two sequences, there are two cases.

In the first case, observations will never differ. Since σ is synchronizing, it brings both states (ℓ_1, z_1) and (ℓ_2, z_2) eventually into the same state, in particular to the same location, witnessing Property 3 and contradicting to our assumption.

In the second case, after a certain number of steps, the observations of the current states (ℓ_1', z_1') and (ℓ_2', z_2') of the two path will differ, w.l.o.g. $z_1' < 0 \leq z_2'$. Since Property 4 is not satisfied, by Lemma 7 there are cycles increasing and decreasing weight in both ℓ_1 and ℓ_2 by p. The two paths π_1, π_2 produced by the strategy σ in $\mathcal{A}$ induce two sequences $\hat{\pi}_1, \hat{\pi}_2$ where the ith elements are both increased/decreased by $\alpha_i \cdot p$ for some $\alpha_i \in \mathbb{Z}$ so that the larger one is in $[0, p-1]$. These sequences straightforwardly induce a path in $\mathcal{A}_S$, where S is the strongly connected component of *init*. Since *separated* cannot be reached from *init*, the smaller weight is always in $[0, p-1]$, too. Let $\hat{z}_1', \hat{z}_2'$ denote the weights in $\mathcal{A}_S$ when σ achieves z_1', z_2'. Since $z_2' \geq 0$, $\hat{z}_2' < p$, and $\hat{z}_2' \equiv z_2' \pmod{p}$, we obtain $\hat{z}_2' \leq z_2'$. Therefore, by $\hat{z}_2' - \hat{z}_1' = z_2' - z_1'$ we also get $\hat{z}_1' \leq z_1'$. Since $z_1' < 0$, we obtain $\hat{z}_1' < 0$, a contradiction. ◀

4 Complexity

We can now state our main theorem:

▶ **Theorem 9.** *The synchronizability problem for deterministic weighted automata is decidable in polynomial time.*

Proof. Properties 1-5 form sufficient and necessary conditions for the existence of a synchronizing strategy for $\mathcal{A}$ by Lemma 5 and Lemma 8. Moreover, all properties can be verified in polynomial time. Indeed, the size of $G^{\mathcal{A}}$ is polynomial in $|\mathcal{A}|$ and p necessary for constructing $\mathcal{A}_S$ is computable in polynomial time by Lemma 5, and the presence of ± 1 cycles is decided in polynomial time by Theorem 11 as discussed in the rest of this section. ◀

We now prove that the presence of ± 1 cycles can be decided in polynomial time. We assume a weighted graph $G = (V, E)$ where V is a finite set of nodes and $E \subseteq V \times \mathbb{Z} \times V$ are the edges written as $u \xrightarrow{w} v$ whenever $(u, w, v) \in E$. A *path* in G is a sequence of edges $v_0 \xrightarrow{w_0} v_1 \xrightarrow{w_1} \dots \xrightarrow{w_{n-1}} v_n$. A *weight of a path* π is defined as $|\pi| = \sum_{i=0}^{n-1} w_i$. A *$k$-cycle* is a path π where $v_0 = v_n$ such that $k = |\pi|$.

▶ Remark. We first briefly discuss related problems and point to severe differences, preventing us from adapting the existing results. On the one hand, we note that the problem whether there is a k-cycle, where k is a part of the input, is NP-hard (see full version of the paper). On the other hand, it is a classical result [11] that existence of 0-cycles is decidable in polynomial time. The result can be proven by a reduction to linear programming. The idea is the following. For each transition, there is a variable encoding the frequency of the transition on the desired cycle. Encoding of Kirchhoff's flow-preservation laws then ensures that the frequencies indeed induce a cycle. Finally, the sum of transition weights multiplied by the frequencies is required to be 0. From every rational solution, we can by multiplication obtain an integer solution, and thus a realizable cycle. Since 0 multiplied by any number remains 0, we thus obtain a 0-cycle. In contrast, in our setting, this idea cannot be used. Indeed, suppose we require the frequency-weighted sum of edge-weights to be 1. Since the frequencies and thus also the number 1 must be multiplied by an a priori unknown integer, in order to obtain an integer solution, the resulting total weight is not 1. Asking instead directly for an integer solution to the system is an instance of integer linear programming, which is an NP-hard problem. Instead of using linear programming, we employ (as shown in the full version of the paper) a number theoretic arguments and exploit Dijkstra's shortest path algorithm on graphs where weights are counted modulo various numbers. Finally, note that although we can decide the existence of ± 1-cycles in polynomial time, the length (number of edges) of the shortest one may still be exponential. For instance, consider a single vertex with two self-loops labelled by $2^n + 1$ and -2.

The discussion suggests that number theoretic techniques have to be applied. We reduce our problem to the problem whether the greatest common divisor of all cycles in a graph is 1. Formally, for a weighted graph G, let the *period* $\gcd(G)$ denote $\gcd\{k \mid k \in \mathbb{Z}, G \text{ has a } k\text{-cycle}\}$.

▶ **Proposition 10.** For every strongly connected weighted graph G, there is a 1-cycle and a -1-cycle in G if and only if there is a positive and a negative cycle in G and $\gcd(G) = 1$.

Proof. The 'Only-if' direction is trivial. For the 'If' direction, $\gcd(G) = 1$ yields by Bézout's identity an equality

$$1 = \alpha_1 \cdot k_1 + \dots + \alpha_n \cdot k_m \tag{1}$$

for some $m \in \mathbb{N}$, $\alpha_i \in \mathbb{Z}$, and k_i being the weight of some cycle c_i in G, and where, moreover, some $k_p > 0$ and some $k_n < 0$. Note that these numbers can be extracted using the extended Euclidean algorithm. First, we argue, we can choose all $\alpha_i \geq 0$ so that Equation (1) still holds.

Whenever $\alpha_i < 0$ with k_i positive, we increase α_i by $x \cdot (-k_n)$ for some $x \in \mathbb{N}$ so that it becomes positive. Further, we increase α_n by $x \cdot k_i$, thus preserving Equation (1). For negative k_i, we proceed similarly, using k_p and α_p instead. Since this procedure only increases α's, they all eventually become positive.

Nonnegative coefficients α_i determine the number of repetitions of each cycle c_i. Since these cycles may be disconnected, this does not yield a single 1-cycle yet. To this end, we consider a negative cycle visiting each vertex of G, guaranteed by assumptions. Let $-\omega$ denote its weight. We construct a 1-cycle by executing this cycle and on the way, whenever reaching a vertex where the cycle c_i originates, we execute c_i for $(\omega+1)\cdot\alpha_i$ times. A -1-cycle is constructed similarly. ◀

▶ **Theorem 11.** *The presence of both a* 1*-cycle and at the same time a* -1*-cycle in a weighted graph* G *is decidable in polynomial time. Moreover, such cycles can be effectively constructed.*

Proof. Deciding presence of a negative cycle and producing a witness can be done in polynomial time using, for instance, Bellman-Ford algorithm (see e.g. [3]); the same holds for positive cycles by swapping the signs.

The period of a graph can be computed in polynomial time, too. Indeed, the result for unweighted graphs (all weights are one) was proven in [10]. Further, [1] suggests an extension of the technique to weighted graphs. Since [10] is to the best of our knowledge not accessible electronically (the only hardcopy of the report is located at library of Stanford University) and the correctness of the extension to weighted graphs is not proven in [1], we also provide our own proof, using supposedly different techniques. Full version of the paper gives the details. ◀

▶ Remark. The polynomial time algorithm for deciding synchronizability is relying only a single observation, testing whether the accumulated weight is negative or nonnegative. In a more general setting, we may consider a richer set of observations checking whether the weights are less-than/greater-or-equal to a given number of integer values. The techniques in this paper can be directly reused to handle this more generation situation and the only check that must be modified is Property 5. Here, if some observations are far away from each other (the integers that they test have distance more than p) then it is sufficient to check if at least of them succeeds, otherwise the graph $\mathcal{A}_S$ is extended to include weights in the range $[0, kp-1]$ where k is the number of observations that are close to each other so that all of them are considered in the check for distinguishability of a given pair of locations. As the observations are part of the input (of the problem description), this still creates a graph with only polynomially many nodes.

5 Conclusion

We have shown that the synchronization problem for deterministic WA under (minimal) partial observability is decidable in polynomial time. This result is based on a polynomial time algorithm for deciding the existence of $+1$ and -1 cycles in a weighted graph and states five necessary and sufficient conditions for synchronizability. All conditions are verifiable in polynomial time, despite the fact that the length of the resulting synchronization strategy is unbounded (as it depends on the initial weight values). The presented techniques are general and allow for a straightforward adaptation to the situation when more observations become available. Future research will include nontrivial extensions to nondeterministic WA and synchronization under safety constraints, e.g. constraints on the weight-levels encountered during the synchronization.

Acknowledgments. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement 601148 (CASSTING), EU FP7 FET project SENSATION, Sino-Danish Basic Research Center IDAE4CPS, the European Research Council (ERC) under grant agreement 267989 (QUAREM), the Austrian Science Fund (FWF) project S11402-N23 (RiSE) and Z211-N23 (Wittgenstein Award), the Czech Science Foundation under grant agreement P202/12/G061, and People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) REA Grant No 291734.

References

1 Esther M. Arkin, Christos H. Papadimitriou, and Mihalis Yannakakis. Modularity of cycles and paths in graphs. *J. ACM*, 38(2):255–274, 1991.

2 Paolo Baldan and Daniele Gorla, editors. *CONCUR 2014 – Concurrency Theory – 25th International Conference, CONCUR 2014, Rome, Italy, September 2–5, 2014. Proceedings*, volume 8704 of *Lecture Notes in Computer Science*. Springer, 2014.

3 Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer, 2nd edition, 2008.

4 Ján Černý. Poznámka k. homogénnym experimentom s konecnými automatmi. *Mat. fyz. čas SAV*, 14:208–215, 1964.

5 Laurent Doyen, Line Juhl, Kim Guldstrand Larsen, Nicolas Markey, and Mahsa Shirmohammadi. Synchronizing words for weighted and timed automata. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014*, volume 29 of *LIPIcs*, pages 121–132. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2014.

6 Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. Infinite synchronizing words for probabilistic automata. In Filip Murlak and Piotr Sankowski, editors, *Mathematical Foundations of Computer Science 2011*, volume 6907 of *Lecture Notes in Computer Science*, pages 278–289. Springer, 2011.

7 Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. Synchronizing objectives for Markov decision processes. In Johannes Reich and Bernd Finkbeiner, editors, *iWIGP*, volume 50 of *EPTCS*, pages 61–75, 2011.

8 Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. Robust synchronization in Markov decision processes. In Baldan and Gorla [2], pages 234–248.

9 Fedor Fominykh and Mikhail Volkov. P(l)aying for synchronization. In *Implementation and Application of Automata*, volume 7381 of *Lecture Notes in Computer Science*, pages 159–170. Springer, 2012.

10 Donald Knuth. Strong components. Technical Report 004639, Comput. Sci. Dept., Stanford University, Stanford, Calif., 1973.

11 S. Rao Kosaraju and Gregory F. Sullivan. Detecting cycles in dynamic graphs in polynomial time (preliminary version). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 398–406. ACM, 1988.

12 Kim Guldstrand Larsen, Simon Laursen, and Jiří Srba. Synchronizing strategies under partial observability. In Baldan and Gorla [2], pages 188–202.

13 Sven Sandberg. Homing and synchronizing sequences. In *Model-Based Testing of Reactive Systems*, volume 3472 of *Lecture Notes in Computer Science*, pages 5–33. Springer, 2004.

14 Mikhail V. Volkov. Synchronizing automata and the Černý conjecture. In *Language and automata theory and applications*, pages 11–27. Springer, 2008.

SOS Specifications of Probabilistic Systems by Uniformly Continuous Operators

Daniel Gebler[1] and Simone Tini[2]

1 **VU University Amsterdam, The Netherlands**
2 **University of Insubria, Italy**

Abstract

Compositional reasoning over probabilistic systems wrt. behavioral metric semantics requires the language operators to be uniformly continuous. We study which SOS specifications define uniformly continuous operators wrt. bisimulation metric semantics. We propose an expressive specification format that allows us to specify operators of any given modulus of continuity. Moreover, we provide a method that allows to derive from any given specification the modulus of continuity of its operators.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases SOS, probabilistic process algebra, bisimulation metric semantics, compositional metric reasoning, uniform continuity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.155

1 Introduction

Probabilistic programming languages are languages that incorporate probabilistic choice as a primitive. They allow us to describe probabilistic concurrent communicating systems. The operational semantics of those languages is usually described by Structural Operational Semantics (SOS) specifications. A SOS specification assigns to each language expression a transition system with transitions inductively defined by means of SOS rules [19, 7].

As behavioral semantics we consider bisimulation metric [9, 22], which is the quantitative analogue of bisimulation equivalence [17] and assigns to each pair of processes a distance which measures the proximity of their quantitative properties. Compositional reasoning over probabilistic processes and probabilistic programs requires that the language operators are uniformly continuous [12]. Uniform continuity ensures that a small variance in the behavior of the parts leads to a bounded small variance in the behavior of the composed processes.

A successful approach to study systematically compositionality properties is the structural analysis of SOS language specifications [1, 19]. In this approach one analyses SOS specifications that satisfy desired compositionality properties and proposes syntactic SOS rule and specification templates that ensure by construction the compositionality property.

In this paper we develop an expressive SOS specification format guaranteeing that the specified operators are uniformly continuous. The format allows us to specify for each operator its respective modulus of continuity. Our fundamental insight is that an operator is uniformly continuous if it is Lipschitz continuous for each finite projection. The SOS specification format derives then from the definition of Lipschitz factors of the finite projections the guarantee that the specified operator is uniformly continuous. Furthermore, we develop a method to derive from any modulus of continuity the respective syntactic requirements on the specifications ensuring that the specified operators satisfy this modulus of continuity. Moreover, we develop a novel method to derive from any SOS specification the modulus of

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 155–168

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

continuity of its operators. The Lipschitz factor of some operator wrt. the k-th projection, i.e., wrt. the up-to-k bisimulation metric, is determined by the replication of processes in the first k steps, the probabilistic choices in those steps, and the (step) discount of the bisimulation metric. Hence, our analysis provides further insights in the interplay between those determining factors. Our key contributions are:

1. We develop an expressive SOS specification format guaranteeing that all specified operators are uniformly continuous (Thm. 28).
2. We provide a method that allows us to derive for any uniformly continuous operator its respective modulus of continuity from its specification rules (Thms. 27 and 28).
3. We provide a method that, given any modulus of continuity, determines sufficient syntactic requirements s.t. any specification satisfying these requirements defines an operator with that modulus of continuity (Thm. 32).
4. We show by appropriate examples that our SOS specification formats and syntactic requirements cannot be relaxed in any obvious way (Exs. 10–13).
5. We apply those results and derive an upper bound on the distance between language expressions from the syntactic properties of the operators (Thm. 35). This enables metric compositional reasoning over partial program specification [12].

The paper is organized as follows. In Section 2 we recall the necessary technical definitions. In Section 3 we prove that an operator is uniformly continuous if it is Lipschitz continuous for each finite projection. In Section 4 we discuss which structural patterns of SOS rules define uniformly continuous operators. In Section 5 we present our format for uniformly continuous operators. In Section 6 we develop our method to derive from any modulus of continuity the respective syntactic requirements on the specifications ensuring that the specified operators satisfy this modulus of continuity. In Section 7 we show how to apply our results to derive an upper bound on the distance between language expressions from the syntactic properties of the operators. We conclude in Section 8 and discuss possible future work.

2 Preliminaries

The operational semantics of programming languages and process algebras is usually given as a transition system with language expressions (terms) as states and a transition relation inductively defined by means of SOS rules.

Probabilistic Transition Systems. A *signature* is a structure $\Sigma = (F, \mathsf{r})$, where F is a countable set of *operators*, and $\mathsf{r}\colon F \to \mathbb{N}$ is a *rank function*. We will use n for $\mathsf{r}(f)$ if it is clear from the context. By $f \in \Sigma$ we mean $f \in F$. We assume an infinite set of *state variables* $\mathcal{V}_s$. The set of *state terms* over a signature Σ and a set of state variables $V \subseteq \mathcal{V}_s$, notation $\mathsf{T}(\Sigma, V)$, is defined as usual. The set of *closed state terms* $\mathsf{T}(\Sigma, \emptyset)$ is abbreviated as $\mathsf{T}(\Sigma)$. The set of *open state terms* $\mathsf{T}(\Sigma, \mathcal{V}_s)$ is abbreviated as $\mathbb{T}(\Sigma)$.

Probabilistic transition systems extend transition systems by allowing for probabilistic choices in the transitions. We consider probabilistic nondeterministic labelled transition systems [20]. As state space we take the set of all closed terms $\mathsf{T}(\Sigma)$. Probability distributions over this space are mappings $\pi\colon \mathsf{T}(\Sigma) \to [0,1]$ with $\sum_{t \in \mathsf{T}(\Sigma)} \pi(t) = 1$ that assign to each term $t \in \mathsf{T}(\Sigma)$ its respective probability $\pi(t)$. By $\Delta(\mathsf{T}(\Sigma))$ we denote the set of all probability distributions on $\mathsf{T}(\Sigma)$. We let π, π' range over $\Delta(\mathsf{T}(\Sigma))$.

▶ **Definition 1** (PTS [20]). A *probabilistic nondeterministic labeled transition system* (*PTS*) is given by a triple $(\mathsf{T}(\Sigma), A, \to)$, where Σ is a signature, A is a countable set of *actions*, and $\to \subseteq \mathsf{T}(\Sigma) \times A \times \Delta(\mathsf{T}(\Sigma))$ is a *transition relation*.

We write $t \xrightarrow{a} \pi$ for $(t, a, \pi) \in \rightarrow$. Let $der(t, a) = \{\pi \in \Delta(\mathsf{T}(\Sigma)) \mid t \xrightarrow{a} \pi\}$.

Bisimulation metric. Bisimulation metrics are the quantitative analogue to bisimulation equivalences. Let $([0,1]^{\mathsf{T}(\Sigma)\times\mathsf{T}(\Sigma)}, \sqsubseteq)$ be the complete lattice of functions $d, d' \colon \mathsf{T}(\Sigma)\times\mathsf{T}(\Sigma) \to [0,1]$ ordered by $d \sqsubseteq d'$ iff $d(t,t') \leq d'(t,t')$ for all terms $t, t' \in \mathsf{T}(\Sigma)$. The bottom element $\mathbf{0}$ is the constant zero function $\mathbf{0}(t,t') = 0$. A function $d \colon \mathsf{T}(\Sigma) \times \mathsf{T}(\Sigma) \to [0,1]$ is a *1-bounded pseudometric* if $d(t,t) = 0$, $d(t,t') = d(t',t)$ and $d(t,t'') \leq d(t,t') + d(t',t'')$ for all $t, t', t'' \in \mathsf{T}(\Sigma)$. Intuitively, the bisimilarity metric will be a 1-bounded pseudometric d with $d(t,t')$ measuring the maximal distance of quantitative properties between t and t'.

We define now the bisimilarity metric as least fixed point of a monotone function over $([0,1]^{\mathsf{T}(\Sigma)\times\mathsf{T}(\Sigma)}, \sqsubseteq)$ [8]. A pseudometric on terms $\mathsf{T}(\Sigma)$ is lifted to a pseudometric on distributions $\Delta(\mathsf{T}(\Sigma))$ by the Kantorovich pseudometric. This lifting corresponds to the lifting of bisimulation equivalence relations on terms to bisimulation equivalence relations on distributions [22]. A *matching* for a pair of distributions $(\pi, \pi') \in \Delta(\mathsf{T}(\Sigma)) \times \Delta(\mathsf{T}(\Sigma))$ is a distribution over the product state space $\omega \in \Delta(\mathsf{T}(\Sigma) \times \mathsf{T}(\Sigma))$ with π and π' as left and right marginal, i.e., $\sum_{t' \in \mathsf{T}(\Sigma)} \omega(t,t') = \pi(t)$ and $\sum_{t \in \mathsf{T}(\Sigma)} \omega(t,t') = \pi'(t')$ for all terms $t, t' \in \mathsf{T}(\Sigma)$. Let $\Omega(\pi, \pi')$ denote the set of all matchings for (π, π'). The *Kantorovich pseudometric* $\mathbf{K}(d) \colon \Delta(\mathsf{T}(\Sigma)) \times \Delta(\mathsf{T}(\Sigma)) \to [0,1]$ of pseudometric $d \colon \mathsf{T}(\Sigma) \times \mathsf{T}(\Sigma) \to [0,1]$ is defined for all distributions $\pi, \pi' \in \Delta(\mathsf{T}(\Sigma))$ by

$$\mathbf{K}(d)(\pi, \pi') = \min_{\omega \in \Omega(\pi,\pi')} \sum_{t,t' \in \mathsf{T}(\Sigma)} d(t,t') \cdot \omega(t,t').$$

In order to capture nondeterministic choices, we need to lift pseudometrics on distributions to pseudometrics on sets of distributions. The *Hausdorff pseudometric* $\mathbf{H}(\hat{d}) \colon P(\Delta(\mathsf{T}(\Sigma))) \times P(\Delta(\mathsf{T}(\Sigma))) \to [0,1]$ is defined for $\Pi_1, \Pi_2 \subseteq \Delta(\mathsf{T}(\Sigma))$ and $\hat{d} \colon \Delta(\mathsf{T}(\Sigma)) \times \Delta(\mathsf{T}(\Sigma)) \to [0,1]$ by

$$\mathbf{H}(\hat{d})(\Pi_1, \Pi_2) = \max\left\{ \sup_{\pi_1 \in \Pi_1} \inf_{\pi_2 \in \Pi_2} \hat{d}(\pi_1, \pi_2), \sup_{\pi_2 \in \Pi_2} \inf_{\pi_1 \in \Pi_1} \hat{d}(\pi_2, \pi_1) \right\}$$

with $\inf \emptyset = 1$, $\sup \emptyset = 0$.

Now we define $\mathbf{B} \colon [0,1]^{\mathsf{T}(\Sigma)\times\mathsf{T}(\Sigma)} \to [0,1]^{\mathsf{T}(\Sigma)\times\mathsf{T}(\Sigma)}$ by

$$\mathbf{B}(d)(t,t') = \sup_{a \in A} \{\mathbf{H}(\lambda \cdot \mathbf{K}(d))(der(t,a), der(t',a))\}$$

for $d \colon \mathsf{T}(\Sigma) \times \mathsf{T}(\Sigma) \to [0,1]$, $t, t' \in \mathsf{T}(\Sigma)$, $\lambda \in (0,1]$ a discount factor[1], and $(\lambda \cdot \mathbf{K}(d))(\pi, \pi') = \lambda \cdot \mathbf{K}(d)(\pi, \pi')$. $\mathbf{B}$ is a monotone function over $([0,1]^{\mathsf{T}(\Sigma)\times\mathsf{T}(\Sigma)}, \sqsubseteq)$. Prefixed points $\mathbf{B}(d) \sqsubseteq d$ are pseudometrics satisfying the bisimulation transfer condition (for all pairs t and t' each transition from t can be mimicked by an equally labelled transition from t' s.t. the distance between the accessible distributions does not exceed the distance between t and t'). By the Knaster-Tarski theorem $\mathbf{B}$ has a least fixed point, which forms the bisimilarity metric.

▶ **Definition 2** (Bisimilarity metric [9, 8])**.** We call $\mathbf{d}_k = \mathbf{B}^k(\mathbf{0})$ the *up-to-k bisimilarity metric*, and $\mathbf{d} = \lim_{k\to\infty} \mathbf{d}_k$ the *bisimilarity metric*.

By bisimulation distance between t and t' we mean $\mathbf{d}(t,t')$. Bisimulation equivalence [20] is the kernel of the bisimilarity metric [9].

[1] By means of the *discount factor* $\lambda \in (0,1]$ we allow to specify how much the behavioral distance of future transitions is taken into account.

▶ **Example 3.** Consider the probabilistic CCS [23, 15, 12] terms $s = a.a^\omega$ and $t_e = a.([1-e]a^\omega \oplus [e]0)$, with $e \in (0,1)$. Process a^w performs a forever. The transitions $s \xrightarrow{a} \pi_s$, with $\pi_s(a^\omega) = 1$, and $t_e \xrightarrow{a} \pi_t$, with $\pi_t(a^\omega) = 1-e$ and $\pi_t(0) = e$, are derivable. Then $\mathbf{d}(a^\omega, a^\omega) = 0$ and $\mathbf{d}(a^\omega, 0) = 1$. Hence $\mathbf{K}(\mathbf{d})(\pi_s, \pi_t) = e$. Thus, $\mathbf{d}_0(s, t_e) = 0$ and $\mathbf{d}_k(s, t_e) = \lambda e$ if $k \geq 1$. Finally, we get $\mathbf{d}(s, t_e) = \lambda e$.

Algebra of probability distributions. By $\delta(t)$ with $t \in \mathsf{T}(\Sigma)$ we denote the *Dirac distribution* defined by $(\delta(t))(t) = 1$. The convex combination $\sum_{i\in I} p_i\pi_i$ of a family $\{\pi_i\}_{i\in I}$ of distributions $\pi_i \in \Delta(\mathsf{T}(\Sigma))$ with $p_i \in (0,1]$ and $\sum_{i\in I} p_i = 1$ is defined by $(\sum_{i\in I} p_i\pi_i)(t) = \sum_{i\in I}(p_i\pi_i(t))$ for all $t \in \mathsf{T}(\Sigma)$. The expression $f(\pi_1, \ldots, \pi_n)$ with $f \in \Sigma$ and $\pi_i \in \Delta(\mathsf{T}(\Sigma))$ denotes the product distribution defined by $f(\pi_1, \ldots, \pi_n)(f(t_1, \ldots, t_n)) = \prod_{i=1}^n \pi_i(t_i)$.

In order to describe probabilistic behavior, we need syntactic expressions that denote probability distributions. We assume an infinite set of *distribution variables* $\mathcal{V}_d$. We let μ, ν range over $\mathcal{V}_d$. We denote by $\mathcal{V}$ the set of state and distribution variables $\mathcal{V} = \mathcal{V}_s \cup \mathcal{V}_d$. We let ζ, ζ' range over $\mathcal{V}$. The set of *distribution terms* over a set of state variables $V_s \subseteq \mathcal{V}_s$ and a set of distribution variables $V_d \subseteq \mathcal{V}_d$, notation $\mathsf{DT}(\Sigma, V_s, V_d)$, is the least set satisfying [6]:

(i) $V_d \subseteq \mathsf{DT}(\Sigma, V_s, V_d)$,
(ii) $\{\delta(t) \mid t \in \mathsf{T}(\Sigma, V_s)\} \subseteq \mathsf{DT}(\Sigma, V_s, V_d)$,
(iii) $\sum_{i\in I} p_i\theta_i \in \mathsf{DT}(\Sigma, V_s, V_d)$ whenever $\theta_i \in \mathsf{DT}(\Sigma, V_s, V_d)$ and $p_i \in (0,1]$ with $\sum_{i\in I} p_i = 1$, and
(iv) $f(\theta_1, \ldots, \theta_n) \in \mathsf{DT}(\Sigma, V_s, V_d)$ whenever $f \in \Sigma$ and $\theta_i \in \mathsf{DT}(\Sigma, V_s, V_d)$.

We write $\theta_1 \oplus_p \theta_2$ for $\sum_{i=1}^2 p_i\theta_i$ with $p_1 = p$ and $p_2 = 1-p$. Furthermore, we write $\theta_1\, f\, \theta_2$ for $f(\theta_1, \theta_2)$. We write $\mathbb{DT}(\Sigma)$ for $\mathsf{DT}(\Sigma, \mathcal{V}_s, \mathcal{V}_d)$ (set of all *open distribution terms*), and $\mathsf{DT}(\Sigma)$ for $\mathsf{DT}(\Sigma, \emptyset, \emptyset)$ (set of all *closed distribution terms*).

Distribution terms have the following meaning. A *distribution variable* $\mu \in \mathcal{V}_d$ is a variable that takes values from $\Delta(\mathsf{T}(\Sigma))$. An *instantiable Dirac distribution* $\delta(t)$ instantiates to $\delta(t')$ if t instantiates to t'. Case (iii) allows to construct convex combinations of distributions. Case (iv) lifts the structural inductive construction of state terms to distribution terms. Substitutions are defined as usual [7].

SOS specification. We specify the operational semantics of operators by SOS rules. SOS rules are syntax-driven inference rules that define the behavior of complex expressions in terms of the behavior of their components. We employ SOS rules of the probabilistic GSOS format [4, 7, 18, 6]. This format uses triples of the form $t \xrightarrow{a} \theta$ that specify in a single literal all probabilistic choices of a transition. Earlier formats [3, 16, 21] used the old fashion quadruple $t \xrightarrow{a,p} t'$ that decorates the transitions with both the action label and the probability in order to partially specify a probabilistic jump. However, this approach required complicated consistency conditions on the set of all rules to ensure that the partially specified probabilistic jumps define in total probabilistic choices.

▶ **Definition 4** (SOS rule)**.** A *SOS rule* r has the form:

$$\frac{\{x_i \xrightarrow{a_{i,k}} \mu_{i,k} \mid i \in I, k \in K_i\} \qquad \{x_i \xrightarrow{b_{i,l}} \not\rightarrow \mid i \in I, l \in L_i\}}{f(x_1, \ldots, x_n) \xrightarrow{a} \theta}$$

with n the rank of operator $f \in \Sigma$, $I = \{1, \ldots, n\}$ the indices of the arguments of f, K_i, L_i finite index sets, $a_{i,k}, b_{i,l}, a \in A$ actions, $x_i \in \mathcal{V}_s$ state variables, $\mu_{i,k} \in \mathcal{V}_d$ distribution variables, and $\theta \in \mathbb{DT}(\Sigma)$ a distribution term. Furthermore, all $\mu_{i,k}$ for $i \in I, k \in K_i$ are pairwise different, all $x_1, \ldots, x_n$ are pairwise different, and all variables in θ are from $\{\mu_{i,k} \mid i \in I, k \in K_i\} \cup \{x_1 \ldots, x_n\}$.

The expressions $x_i \xrightarrow{a_{i,k}} \mu_{i,k}$, $x_i \xrightarrow{b_{i,l}}\!\!\!\!\!\!/\;\;$ and $f(x_1,\ldots,x_n) \xrightarrow{a} \theta$ are called, resp., *positive premises*, *negative premises* and *conclusion*. The set of all premises is denoted by $\mathrm{prem}(r)$. The term $f(x_1,\ldots,x_n)$ is called the *source*, the variables $x_1,\ldots,x_n$ are called *source variables*, and the distribution term θ is called the *target* (notation $\mathrm{trgt}(r)$). Let $\mathrm{der}(r,x_i) = \{\mu_{i,k} \mid x_i \xrightarrow{a_{i,k}} \mu_{i,k} \in \mathrm{prem}(r)\}$. We call $\mu \in \mathrm{der}(r,x_i)$ a *derivative* of source variable x_i.

A *probabilistic transition system specification* (PTSS) is a triple $P = (\Sigma, A, R)$, where Σ is a signature, A is a countable set of actions and R is a countable set of SOS rules. We denote by R_f the set of rules specifying operator f, i.e., all rules of R with source $f(x_1,\ldots,x_n)$. The unique model of P is a PTS $(\mathsf{T}(\Sigma), A, \to)$, with transitions in $\to$ all and only those for which P offers a justification [7].

Intuitively, a term $f(t_1,\ldots,t_n)$ represents the composition of terms $t_1,\ldots,t_n$ by operator f. A rule r specifies some transition $f(t_1,\ldots,t_n) \xrightarrow{a} \pi$ that represents the evolution of the composed term $f(t_1,\ldots,t_n)$ by action a to the distribution π. We say that a rule with conclusion $f(x_1,\ldots,x_n) \xrightarrow{a} \theta$ *delays* the evolution of the source term x_i if x_i appears in θ, and that the source term x_i *evolves* to $\mu \in \mathrm{der}(r,x_i)$ if μ appears in θ. We say that r *replicates* a source variable x_i if multiple instances of either x_i or x_i-derivatives in $\mathrm{der}(r,x_i)$ appear in the target θ of rule r.

3 Uniform continuity

In order to specify and reason about probabilistic systems in a compositional manner, it is necessary that the operators describing these systems are uniformly continuous [12]. A uniformly continuous operator ensures that a small variance in the behavior of a system component leads to a bounded small variance in the behavior of the composed system. We assume some fixed PTSS $P = (\Sigma, A, R)$.

▶ **Definition 5** (Modulus of continuity)**.** Let $f \in \Sigma$ be some n-ary operator and d be any 1-bounded pseudometric on $\mathsf{T}(\Sigma)$. A mapping $\omega\colon [0,1]^n \to [0,1]$ is an *upper bound on the distance between f-composed terms wrt. d* if for all terms $s_i, t_i \in \mathsf{T}(\Sigma)$

$$d(f(s_1,\ldots,s_n), f(t_1,\ldots,t_n)) \leq \omega(d(s_1,t_1),\ldots,d(s_n,t_n)).$$

An upper bound ω of f wrt. d is a *modulus of continuity of f wrt. d* if ω is continuous at $(0,\ldots,0)$, i.e., $\lim_{(\epsilon_1,\ldots,\epsilon_n)\to(0,\ldots,0)} \omega(\epsilon_1,\ldots,\epsilon_n) = \omega(0,\ldots,0)$, and $\omega(0,\ldots,0) = 0$.

▶ **Definition 6** (Uniformly continuous operator)**.** Let d be any 1-bounded pseudometric on $\mathsf{T}(\Sigma)$. An operator $f \in \Sigma$ is

1. *uniformly continuous wrt. d* if f admits some modulus of continuity wrt. d,
2. *L-Lipschitz continuous wrt. d* with $L \in \mathbb{R}_{\geq 0}$ if $\omega(\epsilon_1,\ldots,\epsilon_n) = L\sum_{i=1}^{n} \epsilon_i$ is a modulus of continuity of f wrt. d, and
3. *Lipschitz continuous wrt. d* if f is L-Lipschitz continuous wrt. d for some $L \in \mathbb{R}_{\geq 0}$.

▶ **Example 7.** Consider the synchronous parallel composition operator specified by

$$\frac{x \xrightarrow{a} \mu \quad y \xrightarrow{a} \nu}{x \mid y \xrightarrow{a} \mu \mid \nu}$$

and terms s and t_e as in Ex. 3. Recall that $\mathbf{d}(s,t_e) = \lambda e$. The transitions $s \mid s \xrightarrow{a} \pi_s$, with $\pi_s = \delta(a^\omega \mid a^\omega)$, and $t_{e_1} \mid t_{e_2} \xrightarrow{a} \pi_t$, with $\pi_t = (1-e_1)(1-e_2)\delta(a^\omega \mid a^\omega) + e_1(1-e_2)\delta(0 \mid a^\omega) + (1-e_1)e_2\delta(a^\omega \mid 0) + e_1e_2\delta(0 \mid 0)$, are derivable. Then, $\mathbf{d}(s \mid s, t_{e_1} \mid t_{e_2}) =$

$\lambda\mathbf{K}(\mathbf{d})(\pi_s,\pi_t) = \lambda(1-(1-e_1)(1-e_2)) \leq \lambda e_1 + \lambda e_2 = \mathbf{d}(s,t_{e_1}) + \mathbf{d}(s,t_{e_2})$. Thm. 27 below will confirm that $\omega(\epsilon_1,\epsilon_2) = \epsilon_1+\epsilon_2$ is a modulus of continuity of the synchronous parallel composition operator wrt. $\mathbf{d}$. Hence, this operator is 1-Lipschitz continuous.

The behavioral distance between two arbitrary terms s and t can be divided in the distance observable by the first k steps and the distance observable after step k. The distance observable after step k is bounded by λ^k.

▶ **Proposition 8.** *Let $s,t \in T(\Sigma)$. Then $\mathbf{d}(s,t) \leq \mathbf{d}_k(s,t) + \lambda^k$ for all $k \in \mathbb{N}$.*

A fundamental insight that we will use later to define the SOS specification format is that an operator is uniformly continuous wrt. the bisimilarity metric if this operator is Lipschitz continuous wrt. all up-to-k bisimilarity metrics.

▶ **Theorem 9.** *Assume $\lambda < 1$. If an operator $f \in \Sigma$ is Lipschitz continuous wrt. $\mathbf{d}_k$ for each $k \in \mathbb{N}$, then f is uniformly continuous wrt. $\mathbf{d}$.*

Hence, we assume now a strictly discounting bisimulation metric with $\lambda < 1$.

4 Analysis of uniformly continuous operators

We analyze now the structural patterns of SOS rules that define uniformly continuous operators and give representative examples of rules that specify operators that are not uniformly continuous. Moreover, we derive from the structural properties of the rules the moduli of continuity of the specified operators.

▶ **Example 10** (Non-recurring process replication)**.** Consider the rules

$$\frac{x \xrightarrow{a} \mu}{f(x) \xrightarrow{a} \theta} \qquad \frac{x \xrightarrow{a} \mu \quad y \xrightarrow{a} \nu}{x \mid y \xrightarrow{a} \mu \mid \nu}$$

with $\theta \in \mathbb{DT}(\Sigma)$ some distribution term. We analyze for various distribution terms θ the modulus of continuity of the specified operator f. We use again the terms s and t_e from Ex. 3. Recall that $\mathbf{d}(s,t_e) = \lambda e$.

Consider $\theta = \delta(x \mid x)$. The operator f replicates the source process x, delays both instances, and lets them evolve in parallel. The transitions $f(s) \xrightarrow{a} \delta(s \mid s)$ and $f(t_e) \xrightarrow{a} \delta(t_e \mid t_e)$ are derivable. It follows that $\mathbf{d}(f(s),f(t_e)) = \lambda\mathbf{K}(\mathbf{d})(\delta(s \mid s),\delta(t_e \mid t_e)) = \lambda\mathbf{d}(s \mid s,t_e \mid t_e) \leq 2\lambda\mathbf{d}(s,t_e)$ (c.f. Ex. 7). Thm. 27 below will confirm that $\omega(\epsilon) = (2\lambda)\epsilon$ is a modulus of continuity of this specification of f.

Consider $\theta = (\delta(x \mid x) \oplus_r \delta(0))$ for some $r \in (0,1)$. The operator f replicates and delays now only with probability r the source process x. The transitions $f(s) \xrightarrow{a} r\delta(s \mid s) + (1-r)\delta(0)$ and $f(t_e) \xrightarrow{a} r\delta(t_e \mid t_e) + (1-r)\delta(0)$ are derivable. Hence, $\mathbf{d}(f(s),f(t_e)) = \lambda r\mathbf{d}(s \mid s,t_e \mid t_e) \leq 2r\lambda\mathbf{d}(s,t_e)$. Thm. 27 below will confirm that $\omega(\epsilon) = (2r\lambda)\epsilon$ is a modulus of continuity of this specification of f.

Consider $\theta = (\mu \mid \mu) \oplus_r \delta(0)$. The operator f replicates (but does not delay) with probability r the source process x. The evolved instances proceed in parallel. The transitions $f(s) \xrightarrow{a} r\delta(a^\omega \mid a^\omega) + (1-r)\delta(0)$ and $f(t_e) \xrightarrow{a} r((1-e)^2\delta(a^\omega \mid a^\omega) + e(1-e)\delta(0 \mid a^\omega) + (1-e)e\delta(a^\omega \mid 0) + e^2\delta(0 \mid 0)) + (1-r)\delta(0)$ are derivable. Now $\mathbf{d}(f(s),f(t_e)) \leq 2r\lambda e = 2r\mathbf{d}(s,t_e)$. Thm. 27 below will confirm that $\omega(\epsilon) = (2r)\epsilon$ is a modulus of continuity of this specification of f.

In essence, Ex. 10 shows that the number of non-recurring process replications, weighted by the probability of their realization, and weighted by the discount factor if processes are delayed, determines the Lipschitz factor of the operator.

▶ **Example 11** (Linear process replication)**.** We proceed with the analysis of Ex. 10 and analyze the specification of recursive replication behavior.

Consider $\theta = \delta(f(x)) \mid \mu$. Note that this specification of f is precisely the π-calculus bang operator. The transitions $f(s) \xrightarrow{a} \pi_s$, with $\pi_s = \delta(f(s) \mid a^\omega)$, and $f(t_e) \xrightarrow{a} \pi_t$, with $\pi_t = (1-e)\delta(f(t_e) \mid a^\omega) + e\delta(f(t_e) \mid 0)$ are derivable. Then $\mathbf{d}(f(s), f(t_e)) = \lambda e + \lambda(1-e)\mathbf{d}(f(s) \mid a^\omega, f(t_e) \mid a^\omega)) = \lambda e + \lambda(1-e)\mathbf{d}(f(s), f(t_e))$. Hence, $\mathbf{d}(f(s), f(t_e)) = \frac{\lambda e}{1-\lambda+\lambda e} \leq \frac{\lambda e}{1-\lambda} = \frac{1}{1-\lambda}\mathbf{d}(s, t_e)$. Intuitively, the operator f spawns and delays in each computation step a new instance of the source process x. Thus, the total number of spawned (resp. discounted) process copies is $\sum_{k=0}^{\infty} \lambda^k = 1/(1-\lambda)$. Hence, $\omega(\epsilon) = \frac{1}{1-\lambda}\epsilon$ (formally shown below by Thm. 27) is a modulus of continuity of this specification of f.

Consider $\theta = \delta(f(x)) \mid \mu \mid \delta(x)$. The specified operator f has the modulus of continuity $\omega(\epsilon) = \frac{1+\lambda}{1-\lambda}\epsilon$. Similarly, if $\theta = \delta(f(x)) \mid \mu \mid \delta(x) \mid \delta(x)$, then $\omega(\epsilon) = \frac{1+2\lambda}{1-\lambda}\epsilon$ is a modulus of continuity of the specified operator f.

In essence, Ex. 11 shows that if the number of recurring process replications is finitely bounded, then the specified operator is Lipschitz continuous.

▶ **Example 12** (Non-linear process replication)**.** We analyze now the fork operation of operating systems specified by the copy operator of [5, 11] with the rules

$$\frac{x \xrightarrow{a} \mu}{\mathsf{cp}(x) \xrightarrow{a} \mu}(a \notin \{l, r\}) \qquad \frac{x \xrightarrow{l} \mu \quad x \xrightarrow{r} \nu}{\mathsf{cp}(x) \xrightarrow{s} \mathsf{cp}(\mu) \mid \mathsf{cp}(\nu)}$$

Actions l and r are the left and right forking actions, and s is the resulting split action. The fork of t is the process $\mathsf{cp}(t)$ evolving by t to the parallel composition of the left fork (l-derivative of t) and the right fork (r-derivative of t). For all other actions $a \notin \{l, r\}$ the process $\mathsf{cp}(t)$ mimics the behavior of t.

First, we show that the copy operator is not Lipschitz continuous. Formally, for any $L \in \mathbb{R}_{\geq 0}$, we show that $\mathbf{d}(\mathsf{cp}(s), \mathsf{cp}(t)) > L\mathbf{d}(s, t)$ for some CCS processes s, t. Let $s_1 = l.([1-e]a \oplus [e]0) + r.([1-e]a \oplus [e]0)$ and $t_1 = l.a + r.a$, and $s_{k+1} = l.s_k + r.s_k$ and $t_{k+1} = l.t_k + r.t_k$. Clearly $\mathbf{d}(s_k, t_k) = \lambda^k e$. Then $\mathbf{d}(\mathsf{cp}(s_k), \mathsf{cp}(t_k)) = \lambda^k(1-(1-e)^{2^k})$. Hence, for any k with $2^k > L$, $\mathbf{d}(\mathsf{cp}(s), \mathsf{cp}(t))/\mathbf{d}(s, t) = (1-(1-e)^{2^k})/e > L$ holds for $s = s_k$, $t = t_k$ and all $0 < e < (2^k - L)/(2^{k-1}(2^k - 1))$. Thus, the copy operator is not Lipschitz continuous.

However, Thm. 27 below will confirm that $\omega(\epsilon) = \inf_{k \in \mathbb{N}}(2^k\epsilon + \lambda^k)$ is a (non-linear) modulus of continuity of the copy operator. Intuitively, the copy operator creates in k steps at most 2^k copies of the source process x, i.e., the copy operator is 2^k-Lipschitz continuous for the up-to-k bisimilarity metric. Then, by Prop. 8 we derive the modulus of continuity wrt. bisimilarity metric from the moduli of continuity of the up-to-k bisimilarity metrics.

In essence, Ex. 12 shows that an operator is uniformly continuous if in each step only finitely many process copies are spawned.

▶ **Example 13** (Non-uniformly continuous operators)**.** Consider the unary operators f and g specified by the following rules for all $k \in \mathbb{N}$:

$$\frac{x \xrightarrow{a} \mu}{f(x) \xrightarrow{a} \underbrace{\mu \mid \ldots \mid \mu}_{k-\text{times}}} \qquad \frac{}{g(x) \xrightarrow{a} \delta(\underbrace{h(\ldots h(}_{k-\text{times}} x))} \qquad \frac{x \xrightarrow{a} \mu}{h(x) \xrightarrow{a} \mu \mid \mu}$$

We start with operator f. We get $\mathbf{d}(f(s), f(t_e)) = \sup_{k\in\mathbb{N}} \lambda(1-(1-e)^k) = \lambda$. The least upper bound on the distance between f-composed processes is $\omega(\epsilon) = \lambda$ if $\epsilon > 0$ and $\omega(0) = 0$. However, ω is not a modulus of continuity since it is not continuous at 0. Hence, operator f is not uniformly continuous.

We proceed with operator g. We get $\mathbf{d}(g(s), g(t_e)) = \sup_{k\in\mathbb{N}} \lambda^2(1-(1-e)^{2^k}) = \lambda^2$. Following the same line of reasoning as with operator f we conclude that operator g is not uniformly continuous.

In essence, Ex. 13 shows that an operator may be not uniformly continuous if there is no bound on the number of process copies it can spawn in a single step.

5 Specification of uniformly continuous operators

We develop now a specification format that allows us to specify uniformly continuous operators. We exploit Thm. 9 and specify uniformly continuous operators by defining suitable Lipschitz factors wrt. all up-to-k bisimilarity metrics.

5.1 Finite projection Lipschitz continuous operators

▶ **Definition 14** (Lipschitz factor assignment)**.** We call a mapping[2] $L\colon (\mathbb{N}\times\Sigma) \to \mathbb{R}^{\infty}_{\geq 0}$ a *Lipschitz factor assignment* (*LFA*, for short) for operators in Σ. Let $\mathcal{L}_\Sigma$ be the set of all LFAs for Σ, with $L, M \in \mathcal{L}_\Sigma$ ordered $L \sqsubseteq M$ iff $L_k(f) \leq M_k(f)$ for all $k \in \mathbb{N}$ and $f \in \Sigma$.

Intuitively, $L_k(f)$ is either the Lipschitz factor of operator $f \in \Sigma$ wrt. $\mathbf{d}_k$, or ∞ if f is not Lipschitz continuous wrt. $\mathbf{d}_k$.

▶ **Proposition 15.** *$(\mathcal{L}_\Sigma, \sqsubseteq)$ is a complete lattice.*

It is clear that the bottom element of the lattice $(\mathcal{L}_\Sigma, \sqsubseteq)$ is the LFA $0 \in \mathcal{L}_\Sigma$ given by $0_k(f) = 0$ for all $k \in \mathbb{N}$ and $f \in \Sigma$.

▶ **Definition 16** (Semantic consistency)**.** Let $L \in \mathcal{L}_\Sigma$ be a LFA and $k \in \mathbb{N}$. We call L *consistent with the up-to-k bisimilarity metric* $\mathbf{d}_k$ if

$$\mathbf{d}_k(f(s_1,\ldots,s_n), f(t_1,\ldots,t_n)) \leq L_k(f)\sum_{i=1}^{n} \mathbf{d}_k(s_i,t_i)$$

for all operators $f \in \Sigma$ and terms $s_i, t_i \in \mathsf{T}(\Sigma)$. Furthermore, we call L *consistent with the bisimilarity metric* $\mathbf{d}$ if L is consistent with $\mathbf{d}_k$ for all $k \in \mathbb{N}$.

Hence $L \in \mathcal{L}_\Sigma$ is consistent with $\mathbf{d}_k$ if each operator f with $L_k(f) < \infty$ is $L_k(f)$-Lipschitz continuous wrt. $\mathbf{d}_k$. We proceed by lifting LFAs from operators to terms.

▶ **Definition 17** (LFA on terms)**.** Let $L \in \mathcal{L}_\Sigma$ be a LFA. The lifting of L is a *Lipschitz factor assignment on terms* given as the mapping $L\colon (\mathbb{N}\times(\mathbb{T}(\Sigma)\cup\mathbb{DT}(\Sigma))\times\mathcal{V}) \to \mathbb{R}^{\infty}_{\geq 0}$ defined by:

$$L_k(t,\zeta) = \begin{cases} 1 & \text{if } t = \zeta \\ L_k(f)\displaystyle\sum_{i=1}^{n} L_k(t_i,\zeta) & \text{if } t = f(t_1,\ldots,t_n) \\ 0 & \text{otherwise} \end{cases}$$

[2] We will write the first argument of L as subscript, i.e., $L_k(f)$ for $L(k, f)$, to align with the notation $\mathbf{d}_k$ of up-to-k-bisimilarity metric.

$$L_k(\theta,\zeta) = \begin{cases} 1 & \text{if } \theta = \zeta \\ L_k(t,\zeta) & \text{if } \theta = \delta(t) \\ \sum_{i\in I} p_i \cdot L_k(\theta_i,\zeta) & \text{if } \theta = \sum_{i\in I} p_i\theta_i \\ L_k(f)\sum_{i=1}^{n} L_k(\theta_i,\zeta) & \text{if } \theta = f(\theta_1,\dots,\theta_n) \text{ and } \zeta \in \mathcal{V}_s \\ \overline{L_k(f)}\sum_{i=1}^{n} L_k(\theta_i,\zeta) & \text{if } \theta = f(\theta_1,\dots,\theta_n) \text{ and } \zeta \in \mathcal{V}_d \\ 0 & \text{otherwise} \end{cases}$$

with $\overline{L_k(f)} = \max(L_k(f), 1)$.

The Lipschitz factor of a state term arises from the functional composition of the Lipschitz moduli of continuity of the operators in the state term. Similarly, also for distribution terms except for operators with $L_k(f) < 1$ (case 5 of $L_k(\theta,\zeta)$). As shown in [13, Sec. 4.2], if f has a modulus of continuity on state terms below 1-Lipschitz continuity, then the modulus of continuity of f on distribution terms is 1-Lipschitz continuity (but not smaller).

The lifting of a LFA preserves consistency.

▶ **Proposition 18.** *Let $L \in \mathcal{L}_\Sigma$ be a LFA and $k \in \mathbb{N}$. If L is consistent with $\mathbf{d}_k$, then for any term $t \in \mathbb{T}(\Sigma)$ we have*

$$\mathbf{d}_k(\sigma_1(t),\sigma_2(t)) \le \sum_{x\in\mathcal{V}_s} L_k(t,x)\cdot \mathbf{d}_k(\sigma_1(x),\sigma_2(x))$$

for all closed substitutions $\sigma_1,\sigma_2\colon \mathcal{V} \to T(\Sigma)$.

The set of SOS rules R gives rise to a mapping $R\colon \mathcal{L}_\Sigma \to \mathcal{L}_\Sigma$ with $R(L)$ defined as the LFA obtained by applying the rules of R to L.

▶ **Definition 19** (R-extension)**.** The *R-extension of LFAs* is the mapping[3] $R\colon \mathcal{L}_\Sigma \to \mathcal{L}_\Sigma$ defined by

$$R(L)_0(f) = 0$$

$$R(L)_{k+1}(f) = \sup_{r\in R_f} \max_{i=1}^{r(f)} \left(\lambda \cdot L_k(\mathrm{trgt}(r), x_i) + \sum_{\mu\in \mathrm{der}(r,x_i)} L_k(\mathrm{trgt}(r),\mu) \right)$$

for all $L \in \mathcal{L}_\Sigma$ and $f \in \Sigma$.

Intuitively, the lifted LFA on terms (Def. 17) is obtained by structural induction over terms, while the R-extended LFA (Def. 19) is obtained by operational induction over rules. The R-extension of Lipschitz factor assignments preserves semantic consistency.

▶ **Proposition 20.** *Let $L \in \mathcal{L}_\Sigma$ be a LFA and $k \in \mathbb{N}$. If L is consistent with $\mathbf{d}_k$, then $R(L)$ is consistent with $\mathbf{d}_{k+1}$.*

▶ **Corollary 21.** *If L is consistent with $\mathbf{d}$, then $R(L)$ is consistent with $\mathbf{d}$.*

[3] The symbol R denotes both the set of rules of some specification and the R-extension mapping of LFAs induced by a set of rules R. The meaning of symbol R will always be clear from the application context.

The R-extension mapping allows us to specify a canonical LFA given as the least fixed-point of R. Existence and uniqueness follow by the Knaster-Tarski theorem using that $(\mathcal{L}_\Sigma, \sqsubseteq)$ is a complete lattice (Prop. 15) and that R is monotone (Prop. 22). Since the bottom LFA $0 \in \mathcal{L}_\Sigma$ is consistent with $\mathbf{d}_0$ and R preserves consistency of LFAs (Prop. 20), we get that the canonical LFA is consistent with $\mathbf{d}$. The canonical LFA provides the least restricting syntactic requirements for the specified operators.

▶ **Proposition 22.** *R is order-preserving on $(\mathcal{L}_\Sigma, \sqsubseteq)$.*

▶ **Definition 23** (Canonical LFA)**.** Let $P = (\Sigma, A, R)$ be a PTSS. We call $L_P = \lim_{n\to\infty} R^n(0)$ the *canonical LFA of P*.

Dual to the notion of semantic consistency of LFAs (Def. 16) we introduce now the notion of syntactic consistency of LFAs. Intuitively, a syntactically consistent LFA ensures that the Lipschitz factors are compatible with the rules.

▶ **Definition 24** (Syntactic consistency)**.** Let $P = (\Sigma, A, R)$ be a PTSS and $L \in \mathcal{L}_\Sigma$ some LFA. We call L *consistent with P* (or alternatively L is P-consistent) if $R(L) \sqsubseteq L$.

In other words, all prefixed points of R are consistent with P. In particular, the canonical LFA L_P is consistent with P. Moreover, L_P is the least LFA consistent with P. The syntactic consistency condition $R(L) \sqsubseteq L$ of LFA L with a specification $P = (\Sigma, A, R)$ is a syntactical invariance condition on P that mimics the semantical bisimulation invariance condition $\mathbf{B}(\mathbf{d}) \sqsubseteq \mathbf{d}$ on the induced model $(\mathsf{T}(\Sigma), A, \to)$.

Semantic consistency of a LFA L (Def. 16) means consistency of L with the bisimilarity metric $\mathbf{d}$ on the induced model $(\mathsf{T}(\Sigma), A, \to)$, whereas syntactic consistency of L (Def. 24) means consistency of L with the specification $P = (\Sigma, A, R)$ from which the model is derived. As expected, syntactic consistency implies semantic consistency.

▶ **Proposition 25** (Syntactic consistency implies semantic consistency)**.** *Let $P = (\Sigma, A, R)$ be a PTSS and $L \in \mathcal{L}_\Sigma$ a LFA. If L is consistent with P then L is also consistent with $\mathbf{d}$.*

5.2 Uniformly continuous operators

A P-consistent LFA allows for deriving for each operator f an upper bound on the distance between f-composed terms.

▶ **Definition 26** (Upper bound induced by a LFA)**.** Let $P = (\Sigma, A, R)$ be a PTSS and $L \in \mathcal{L}_\Sigma$ a LFA. We define for any n-ary operator $f \in \Sigma$ the *upper bound on the distance of f-composed processes induced by L* as the mapping $\omega_{L,f} \colon (\mathbb{R}_{\geq 0})^n \to \mathbb{R}^\infty_{\geq 0}$ defined by

$$\omega_{L,f}(\epsilon_1, \ldots, \epsilon_n) = \inf_{k\in\mathbb{N}} \left(L_k(f) \sum_{i=1}^{n} \epsilon_i + \lambda^k \right)$$

If L is consistent with P, then $\omega_{L,f}$ is an upper bound on the distance between f-composed terms wrt. $\mathbf{d}$.

▶ **Theorem 27.** *Let $P = (\Sigma, A, R)$ be a PTSS and $L \in \mathcal{L}_\Sigma$ a LFA consistent with P. Then*

$$\mathbf{d}(f(s_1, \ldots, s_n), f(t_1, \ldots, t_n)) \leq \omega_{L,f}(\mathbf{d}(s_1, t_1), \ldots, \mathbf{d}(s_n, t_n)).$$

Moreover, if L is consistent with P, then $\omega_{L,f}$ is a modulus of continuity of f wrt. $\mathbf{d}$ if all Lipschitz factors $L_k(f)$ of f are finite.

▶ **Theorem 28.** *Let $P = (\Sigma, A, R)$ be a PTSS and $L \in \mathcal{L}_\Sigma$ a LFA consistent with P. An operator $f \in \Sigma$ is*

1. *uniformly continuous if $L_k(f) < \infty$ for all $k \in \mathbb{N}$,*
2. *Lipschitz continuous if $\sup_{k\in\mathbb{N}} L_k(f) < \infty$, and*
3. *K-Lipschitz continuous if $L_k(f) \leq K$ for all $k \in \mathbb{N}$.*

Hence, if f is Lipschitz continuous, then $\sup_{k\in\mathbb{N}} L_k(f)$ is a Lipschitz factor of f. Since the canonical LFA L_P is the least LFA consistent with P it suffices to verify the conditions of Thm. 28 on the canonical LFA.

We provide now an example that shows how to derive the canonical LFA, how to compute the modulus of continuity, and how to determine the resp. compositionality property.

▶ **Example 29.** Let $P = (\Sigma, A, R)$ be the PTSS specifying the synchronous parallel composition operator (Ex. 7) and the copy operator (Ex. 12). Let $L \in \mathcal{L}_\Sigma$ be defined as $L_0(|) = 0$ and $L_0(\mathsf{cp}) = 0$, and $L_k(|) = 1$ and $L_k(\mathsf{cp}) = 2^k$ for any $k \in \mathbb{N}_{>0}$. First we show that L is the canonical LFA $L_P = \lim_{n\to\infty} R^n(0)$ (Def. 23). Observe that $R^{n+1}(0)_k = R^n(0)_k$ for all $k \leq n$. By induction over n. Base case $R^0(0)_0 = 0 = L_0(|)$ and $R^0(\mathsf{cp})_0 = 0 = L_0(\mathsf{cp})$ is obvious. The induction step is $R^{n+1}(0)_{n+1}(|) = \max(\lambda \cdot R^n(0)_n(\mu \mid \nu, x) + R^n(0)_n(\mu \mid \nu, \mu), \lambda \cdot R^n(0)_n(\mu \mid \nu, y) + R^n(0)_n(\mu \mid \nu, \nu)) =$ (inductive hypthesis) $\max(\lambda \cdot L_n(\mu \mid \nu, x) + L_n(\mu \mid \nu, \mu), \lambda \cdot L_n(\mu \mid \nu, y) + L_n(\mu \mid \nu, \nu)) = \max(0 + 1, 0 + 1) = 1 = L_{n+1}(|)$ and $R^{n+1}(0)_{n+1}(\mathsf{cp}) = \max(R^n(0)_n(\mu, \mu), R^n(0)_n(\mathsf{cp}(\mu) \mid \mathsf{cp}(\nu), \mu) + R^n(0)_n(\mathsf{cp}(\mu) \mid \mathsf{cp}(\nu), \nu)) =$ (inductive hypthesis) $\max(L_n(\mu, \mu), L_n(\mathsf{cp}(\mu) \mid \mathsf{cp}(\nu), \mu) + L_n(\mathsf{cp}(\mu) \mid \mathsf{cp}(\nu), \nu)) = \max(1, 2^n + 2^n) = 2^{n+1} = L_{n+1}(\mathsf{cp})$. Hence by Thm. 27 we get that $\omega_{(L,|)}(\epsilon_1, \epsilon_2) = \epsilon_1 + \epsilon_2$, $\omega_{(L,\mathsf{cp})}(\epsilon) = \inf_{k\in\mathbb{N}}(2^k\epsilon + \lambda^k)$ are upper bounds for $|$ and cp wrt. $\mathbf{d}$. By Thm. 28 get that the operator $|$ is 1-Lipschitz continuous and that the operator cp is uniformly continuous. Moreover, the upper bounds are indeed moduli of continuity.

6 From modulus of continuity to operator specifications

In reverse, we derive now from any modulus of continuity ω a LFA L s.t. any PTSS P consistent with L specifies an operator that has ω as modulus of continuity.. The derived LFA depends on ω and the underlying model of process replication. The model of process replication is given as a mapping $\chi\colon \mathbb{R}_{\geq 0} \times \mathbb{N} \to \mathbb{R}_{\geq 0}$ assigning to each step k an upper bound on the number of spawned process instances. The first argument is a fixed growth factor.

▶ **Definition 30** (Growth function)**.** We define the following *growth functions* $\chi\colon \mathbb{R}_{\geq 0} \times \mathbb{N} \to \mathbb{R}_{\geq 0}$:

1. $\chi(c, k) = c$ (constant),
2. $\chi(c, k) = c \cdot k$ (linear growth),
3. $\chi(c, k) = c^k$ (exponential growth).

The constant growth function expresses that at most c process instances are spawned irrespective of the number of steps performed by the combined process (cf. non-recurring process replication, Ex. 10). The linear growth function will be used to model operators with bounded stepwise replication (cf. recurring step-bounded process replication, Ex. 11). Similarly, the exponential growth function allows us to model continuously replicating operators (cf. recurring step-unbounded process replication, Ex. 12).

▶ **Definition 31** (LFA induced by ω and χ)**.** Assume a function $\omega\colon [0,1]^n \to [0,1]$ s.t. $\omega(0,\ldots,0) = 0$ and $\lim_{(\epsilon_1,\ldots,\epsilon_n)\to(0,\ldots,0)} \omega(\epsilon_1,\ldots,\epsilon_n) = \omega(0,\ldots,0)$, a growth function χ and an operator $f \in \Sigma$. The *LFA* $L^f_{\omega,\chi}$ *induced by* ω *and* χ *for* f is defined by

$$L^f_{\omega,\chi,k}(g) = \begin{cases} \chi(C,k) & \text{if } g = f \\ \infty & \text{if } g \neq f \end{cases}$$

with $C = \sup\{c \in \mathbb{R}_{\geq 0} \mid \forall L \in \mathcal{L}_\Sigma.\, ((\forall k \in \mathbb{N}. L_k(f) = \chi(c,k)) \Rightarrow \omega_{L,f} \leq \omega)\}$.

The LFA induced by the exponential growth function is the LFA arising from maximal recurring process replications. The recurring process replication factor C is the maximal process replication per single transition step (possibly repeated along the evolution of the combined process).

▶ **Theorem 32.** *Let* $P = (\Sigma, A, R)$ *be a PTSS and* $L^f_{\omega,\chi}$ *the LFA induced by* ω *and* χ *for* f*. If there exists a* P*-consistent LFA* $L \in \mathcal{L}_\Sigma$ *with* $L \sqsubseteq L^f_{\omega,\chi}$*, then* P *specifies* f *s.t.* f *admits* ω *as modulus of continuity.*

▶ **Example 33.** To define an operator that may not increase the behavioral distance of its argument, assume the modulus of continuity $\omega(\epsilon) = \epsilon$ (1-Lipschitz continuity). The LFA $L^f_{\omega,\chi}$ induced by ω and $\chi(1,k) = 1$ for f (Def. 30.1, Def. 31, Def. 26) gives $L^f_{\omega,\chi,k}(f) = 1$. Let the operator f be specified by the rule (with $\theta \in \mathbb{DT}(\Sigma)$ be any distribution term):

$$\frac{x \xrightarrow{a} \mu}{f(x) \xrightarrow{a} \theta}\,.$$

Clearly, $\theta = \mu$ specifies operator f s.t. $L^f_{\omega,\chi}$ is consistent with P (Def. 24) and that operator f admits ω as modulus of continuity (Thm. 32). Let $t \in \mathsf{T}(\Sigma)$ be any closed term describing some alternative process behavior. With the same argument, also $\theta = \delta(\mu \mid \mu) \oplus_p \delta(t)$ with $p \leq 1/2$ (2 instances proceed with probability at most $1/2$), $\theta = \delta(x \mid x) \oplus_p \delta(t)$ with $p \leq 1/(2\lambda)$ (2 instances proceed with one step delay with probability at most $1/(2\lambda)$), and $\theta = \delta(a^n.(x \mid x)) \oplus_p \delta(t)$ with $p \leq 1/(2\lambda^{n+1})$ ($a^n.\_$ is action prefix operator performing n-times action a followed by the argument process) specify each operator f admitting ω as modulus of continuity (Thm. 32).

We conclude by observing that $\theta = f(\mu) \mid \mu$ specifies operator f s.t. P is consistent with $L^f_{\omega,\chi}$ whereby $L^f_{\omega,\chi,k}(f) = 2k$ is obtained from the linear growth function $\chi(2,k) = 2k$ and the modulus of continuity $\omega(\epsilon) = \inf_{k\in\mathbb{N}}(2k\epsilon + \lambda^k)$. In the same way we can derive that $\theta = f(\mu) \mid f(\mu)$ specifies operator f s.t. P is consistent with $L^f_{\omega,\chi,k} = 2^k$ obtained from exponential growth function $\chi(2,k) = 2^k$ and the modulus of continuity $\omega(\epsilon) = \inf_{k\in\mathbb{N}}(2^k\epsilon + \lambda^k)$.

7 Syntactic and semantic compositionality

LFAs induced by moduli of continuity and growth functions (Def. 31) are compositional. This allows us to determine the LFA for multiple operators separately, and then to specify those operators simultaneously in a specification consistent with the composed LFAs.

▶ **Theorem 34.** *Let* $P = (\Sigma, A, R)$ *be a PTSS and* $G \subseteq \Sigma$ *be a set of operators. For each* $g \in G$ *let* $L^g_{\omega_g,\chi_g}$ *be the LFA induced by some* ω_g *and* χ_g *for* g*. If for each* $g \in G$ *the LFA* $L^g_{\omega_g,\chi_g}$ *is consistent with* P*, then also the LFA* $\inf_{g\in G} L^g_{\omega_g,\chi_g}$ *is consistent with* P*.*

Upper bounds of operators (Def. 5) are compositional. Hence, we define now an upper bound on the distance between two closed instances of a term by composing the moduli of continuity of the operators of that term. In essence, the following theorem lifts Thm. 27 to terms.

▶ **Theorem 35.** *Let $P = (\Sigma, A, R)$ be a PTSS, $L \in \mathcal{L}_\Sigma$ a LFA consistent with P and $t \in \mathbb{T}(\Sigma)$ any open term. For all closed substitutions $\sigma_1, \sigma_2 \colon \mathcal{V} \to \mathsf{T}(\Sigma)$ we get*

$$\mathbf{d}(\sigma_1(t), \sigma_2(t)) \leq \inf_{k \in \mathbb{N}} \left(\sum_{x \in \mathcal{V}_s} L_k(t, x) \cdot \mathbf{d}(\sigma_1(x), \sigma_2(x)) + \lambda^k \right) .$$

▶ **Example 36.** We start by exemplifying Thm. 34. We consider the specification $P = (\Sigma, A, R)$ of operators $G = \{\_ \mid \_, \mathsf{cp}(\_)\}$. As shown in Ex. 29 the LFAs $L^{|}_{\omega_|, \chi_|, k}(|) = 1$, $L^{|}_{\omega_|, \chi_|, k}(\mathsf{cp}) = \infty$ and $L^{\mathsf{cp}}_{\omega_{\mathsf{cp}}, \chi_{\mathsf{cp}}, k}(\mathsf{cp}) = 2^k$, $L^{\mathsf{cp}}_{\omega_{\mathsf{cp}}, \chi_{\mathsf{cp}}, k}(|) = \infty$ (Def. 31) are consistent with P. Then by Thm. 34 $L = \inf_{g \in G} L^g_{\omega_g, \chi_g}$ with $L_k(|) = 1$ and $L_k(\mathsf{cp}) = 2^k$ is consistent with P.

We proceed by exemplifying Thm. 35. Consider terms $t = \mathsf{cp}(x \mid x)$. By using $L_k(|) = 1$ and $L_k(\mathsf{cp}) = 2^k$ (Ex. 29), we get $L_k(\mathsf{cp}(x \mid x), x) = L_k(\mathsf{cp}) \cdot L_k(x \mid x, x) = 2^k \cdot (L_k(|) \cdot (L_k(x, x) + L_k(x, x)) = 2^{k+1}$ and $L_k(\mathsf{cp}(x) \mid \mathsf{cp}(x), x) = 2^{k+1}$. Hence, by Thm. 35 we get $\mathbf{d}(\sigma_1(t), \sigma_2(t)) \leq \inf_{k \in \mathbb{N}}(2^{k+1} \cdot \mathbf{d}(\sigma_1(x), \sigma_2(x)) + \lambda^k)$ for all closed substitutions $\sigma_1, \sigma_2 \colon \mathcal{V} \to \mathsf{T}(\Sigma)$. Equally, for $t = \mathsf{cp}(x) \mid \mathsf{cp}(x)$ we get $L_k(\mathsf{cp}(x) \mid \mathsf{cp}(x), x) = 2^{k+1}$ and $\mathbf{d}(\sigma_1(t), \sigma_2(t)) \leq \inf_{k \in \mathbb{N}}(2^{k+1} \cdot \mathbf{d}(\sigma_1(x), \sigma_2(x)) + \lambda^k)$. The nesting of the copy operator $\mathsf{cp}(\mathsf{cp}(x))$ induces $L_k(\mathsf{cp}(\mathsf{cp}(x)), x) = 2^{2k}$ with distance bound $\mathbf{d}(\sigma_1(\mathsf{cp}(\mathsf{cp}(x))), \sigma_2(\mathsf{cp}(\mathsf{cp}(x)))) \leq \inf_{k \in \mathbb{N}}(2^{2k} \cdot \mathbf{d}(\sigma_1(x), \sigma_2(x)) + \lambda^k)$.

8 Conclusion

We developed a SOS specification format that allows us to specify simultaneously uniformly continuous operators of arbitrary (and possibly different) moduli of continuity. Our format and results pave the way for a robust and modular approach to specify and verify probabilistic systems using probabilistic process algebras and probabilistic programming languages [9, 14].

We will continue this line of research by developing SOS specification formats for uniformly continuous operators wrt. weak metric semantics [10] and metric variants of branching bisimulation equivalence [2]. Our case studies (partially published in [12]) indicated that concepts such as encapsulation and abstraction are fundamental to perform the metric compositional analysis of systems described by probabilistic process algebras in a scalable manner. A second research direction we plan to investigate is the distance between operators (instead of terms) to describe the behavioral distance whenever one operator needs to be replaced or approximated by another. Intuitively, if an operator becomes unavailable, the distance between operators will suggest an optimal replacement operator to build an alternative system which is closest to the original system.

References

1 Luca Aceto, Wan Fokkink, and Chris Verhoef. Structural operational semantics. In *Handbook of Process Algebra*, pages 197–292. Elsevier, 2001.

2 Suzana Andova and Tim AC Willemse. Branching bisimulation for probabilistic systems: characteristics and decidability. *TCS*, 356(3):325–355, 2006.

3 Falk Bartels. GSOS for probabilistic transition systems. In *Proc. CMCS'02*, volume 65 of *ENTCS*, pages 29–53. Elsevier, 2002.

4 Falk Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. PhD thesis, VU University Amsterdam, 2004.

5 Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *J. ACM*, 42:232–268, 1995.

6 Pedro R. D'Argenio, Daniel Gebler, and Matias David Lee. Axiomatizing Bisimulation Equivalences and Metrics from Probabilistic SOS Rules. In *Proc. FoSSaCS'14*, volume 8412 of *LNCS*, pages 289–303. Springer, 2014.

7 Pedro R. D'Argenio and Matias David Lee. Probabilistic Transition System Specification: Congruence and Full Abstraction of Bisimulation. In *Proc. FoSSaCS'12*, volume 7213 of *LNCS*, pages 452–466. Springer, 2012.

8 Yuxin Deng, Tom Chothia, Catuscia Palamidessi, and Jun Pang. Metrics for Action-labelled Quantitative Transition Systems. In *Proc. QAPL'05*, volume 153 of *EPTCS*, pages 79–96, 2005.

9 Josée Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for Labelled Markov Processes. *TCS*, 318(3):323–354, 2004.

10 Josée Desharnais, Radha Jagadeesan, Vineet Gupta, and Prakash Panangaden. The Metric Analogue of Weak Bisimulation for Probabilistic Processes. In *Proc. LICS'02*, pages 413–422. IEEE, 2002.

11 Wan Fokkink, Rob J. van Glabbeek, and Paulien de Wind. Divide and congruence: From decomposition of modal formulas to preservation of branching and η-bisimilarity. *I&C*, 214:59–85, 2012.

12 Daniel Gebler, Kim G. Larsen, and Simone Tini. Compositional metric reasoning with Probabilistic Process Calculi. In *Proc. FoSSaCS'15*, volume 9034 of *LNCS*, pages 230–245. Springer, 2015.

13 Daniel Gebler and Simone Tini. Fixed-point Characterization of Compositionality Properties of Probabilistic Processes Combinators. In *Proc. EXPRESS/SOS'14*, volume 160 of *EPTCS*, pages 63–78. OPA, 2014.

14 T.A. Henzinger. Quantitative reactive modeling and verification. *Computer Science – R&D*, 28(4):331–344, 2013.

15 Bengt Jonsson and Wang Larsen, Kim G.and Yi. Probabilistic Extensions of Process Algebras. In *Handbook of Process Algebra*, pages 685–710. Elsevier, 2001.

16 Ruggero Lanotte and Simone Tini. Probabilistic bisimulation as a congruence. *ACM TOCL*, 10:1–48, 2009.

17 Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *I&C*, 94:1–28, 1991.

18 Matias David Lee, Daniel Gebler, and Pedro R. D'Argenio. Tree Rules in Probabilistic Transition System Specifications with Negative and Quantitative Premises. In *Proc. EXPRESS/SOS'12*, volume 89 of *EPTCS*, pages 115–130. OPA, 2012.

19 Mohammad Reza Mousavi, Michel A. Reniers, and Jan Friso Groote. SOS formats and meta-theory: 20 years after. *TCS*, 373(3):238–272, 2007.

20 Roberto Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, 1995.

21 Simone Tini. Non-expansive ϵ-bisimulations for probabilistic processes. *TCS*, 411:2202–2222, 2010.

22 Franck van Breugel and James Worrell. Towards quantitative verification of probabilistic transition systems. In *Proc. ICALP'01*, volume 2076 of *LNCS*, pages 421–432. Springer, 2001.

23 Wang Yi and Kim Guldstrand Larsen. Testing probabilistic and nondeterministic processes. In *PSTV*, volume 12, pages 47–61, 1992.

Dynamic Bayesian Networks as Formal Abstractions of Structured Stochastic Processes*

Sadegh Esmaeil Zadeh Soudjani[1], Alessandro Abate[2], and Rupak Majumdar[3]

1,3 Max Planck Institute for Software Systems (MPI-SWS), Germany
{sadegh,rupak}@mpi-sws.org
2 Department of Computer Science, University of Oxford, United Kingdom
alessandro.abate@cs.ox.ac.uk

Abstract

We study the problem of finite-horizon probabilistic invariance for discrete-time Markov processes over general (uncountable) state spaces. We compute discrete-time, finite-state Markov chains as formal abstractions of general Markov processes. Our abstraction differs from existing approaches in two ways. First, we exploit the structure of the underlying Markov process to compute the abstraction separately for each dimension. Second, we employ dynamic Bayesian networks (DBN) as compact representations of the abstraction. In contrast, existing approaches represent and store the (exponentially large) Markov chain explicitly, which leads to heavy memory requirements limiting the application to models of dimension less than half, according to our experiments.

We show how to construct a DBN abstraction of a Markov process satisfying an independence assumption on the driving process noise. We compute a guaranteed bound on the error in the abstraction w.r.t. the probabilistic invariance property; the dimension-dependent abstraction makes the error bounds more precise than existing approaches. Additionally, we show how factor graphs and the sum-product algorithm for DBNs can be used to solve the finite-horizon probabilistic invariance problem. Together, DBN-based representations and algorithms can be significantly more efficient than explicit representations of Markov chains for abstracting and model checking structured Markov processes.

1998 ACM Subject Classification G.3 Markov processes, I.6.1 systems theory

Keywords and phrases Structured stochastic systems, general space Markov processes, formal verification, dynamic Bayesian networks, Markov chain abstraction

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.169

1 Introduction

Markov processes over general uncountable state spaces appear in many areas of engineering such as power networks, transportation, biological systems, robotics, and manufacturing systems. The importance of this class of stochastic processes in applications has motivated a significant research effort into their foundations and their verification.

We study the problem of algorithmically verifying finite-horizon probabilistic invariance for Markov processes, which is the problem of computing the probability that a stochastic process remains within a given set for a given finite time horizon. For finite-state stochastic

* This work was partially supported by the European Commission IAPP project AMBI 324432, and by the John Fell OUP Research Fund.

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 169–183

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

processes, there is a mature theory of model checking discrete-time Markov chains [5], and a number of probabilistic model checking tools [14, 18] that compute explicit solutions to the verification problem. On the other hand, stochastic processes taking values over uncountable state spaces may not have explicit solutions and their numerical verification problems are undecidable even for simple dynamics [1]. A number of studies have therefore explored *abstraction* techniques that reduce the given stochastic process (over a general state space) to a finite-state process, while preserving properties in a quantitative sense [1, 7]. The abstracted model allows the application of standard model checking techniques over finite-state models. The work in [1] has further shown that an explicit error can be attached to the abstraction. This error is computed purely based on continuity properties of the concrete Markov process. Properties proved on the finite-state abstraction can be used to reason about properties of the original system. The overall approach has been extended to linear temporal specifications [24] and software tools have been developed to automate the abstraction procedure [10].

In previous works, the structure of the underlying Markov process (namely, the interdependence among its variables) has not been actively reflected in the abstraction algorithms, and the finite-state Markov chain has been always represented explicitly, which is quite expensive in terms of memory requirements. In many applications, the dynamics of the Markov process, which are characterized by a conditional kernel, often exhibit specific structural properties. More specifically, the dynamics of any state variable depends on only a small number of other state variables and the process noise driving each state variable is assumed to be independent. Examples of such structured systems are models of power grids and sensor-actuator networks as large-scale interconnected networks [23] and mass-spring-damper systems [3, 4].

We present an abstraction and model checking algorithm for discrete-time stochastic dynamical systems over general (uncountable) state spaces. Our abstraction constructs a finite-state Markov abstraction of the process, but differs from previous work in that it is based on a dimension-dependent partitioning of the state space. Additionally, we perform a precise dimension-dependent analysis of the error introduced by the abstraction, and our error bounds can be exponentially smaller than the general bounds obtained in [1]. Furthermore, we represent the abstraction as a dynamic Bayesian network (DBN) [15] instead of explicitly representing the probabilistic transition matrix. The Bayesian network representation uses independence assumptions in the model to provide potentially polynomial sized representations (in the number of dimensions) for the Markov chain abstraction for which the explicit transition matrix is exponential in the dimension. We show how factor graphs and the sum-product algorithm, developed for belief propagation in Bayesian networks, can be used to model check probabilistic invariance properties without constructing the transition matrix. Overall, our approach leads to significant reduction in computational and memory resources for model checking structured Markov processes and provides tighter error bounds.

The material is organized in six sections. Section 2 defines discrete-time Markov processes and the probabilistic invariance problem. Section 3 presents a new algorithm for abstracting a process to a DBN, together with the quantification of the abstraction error. We discuss efficient model checking of the constructed DBN in Section 4, and apply the overall abstraction algorithm to a case study in Section 5. Section 6 outlines some further directions of investigation. Proofs of statements can be found in [9].

2 Markov Processes and Probabilistic Invariance

2.1 Discrete-Time Markov Processes

We write $\mathbb{N}$ for the non-negative integers $\mathbb{N} = \{0, 1, 2, \ldots\}$ and $\mathbb{N}_n = \{1, 2, \ldots, n\}$. We use bold typeset for vectors and normal typeset for one-dimensional quantities.

We consider a discrete-time Markov process $\mathscr{M}_{\mathfrak{s}}$ defined over a general state space, and characterized by the tuple $(\mathcal{S}, \mathcal{B}, T_{\mathfrak{s}})$: $\mathcal{S}$ is the continuous state space, which we assume to be endowed with a metric and to be separable[1]; $\mathcal{B}$ is the Borel σ-algebra associated to $\mathcal{S}$, which is the smallest σ-algebra containing all open subsets of $\mathcal{S}$; and $T_{\mathfrak{s}} : \mathcal{S} \times \mathcal{B} \to [0, 1]$ is a stochastic kernel, so that $T_{\mathfrak{s}}(\cdot, B)$ is a non-negative measurable function for any set $B \in \mathcal{B}$, and $T_{\mathfrak{s}}(\boldsymbol{s}, \cdot)$ is a probability measure on $(\mathcal{S}, \mathcal{B})$ for any $\boldsymbol{s} \in \mathcal{S}$. Trajectories (also called traces or paths) of $\mathscr{M}_{\mathfrak{s}}$ are sequences $(\boldsymbol{s}(0), \boldsymbol{s}(1), \boldsymbol{s}(2), \ldots)$ which belong to the set $\Omega = \mathcal{S}^{\mathbb{N}}$. The product σ-algebra on Ω is denoted by $\mathcal{F}$. Given the initial state $\boldsymbol{s}(0) = \boldsymbol{s}_0 \in \mathcal{S}$ of $\mathscr{M}_{\mathfrak{s}}$, the stochastic Kernel $T_{\mathfrak{s}}$ induces a unique probability measure $\mathcal{P}$ on $(\Omega, \mathcal{F})$ that satisfies the Markov property: namely for any measurable set $B \in \mathcal{B}$ and any $t \in \mathbb{N}$

$$\mathcal{P}\left(\boldsymbol{s}(t+1) \in B | \boldsymbol{s}(0), \boldsymbol{s}(1), \ldots, \boldsymbol{s}(t)\right) = \mathcal{P}\left(\boldsymbol{s}(t+1) \in B | \boldsymbol{s}(t)\right) = T_{\mathfrak{s}}(\boldsymbol{s}(t), B).$$

We assume that the stochastic kernel $T_{\mathfrak{s}}$ admits a density function $t_{\mathfrak{s}} : \mathcal{S} \times \mathcal{S} \to \mathbb{R}_{\geq 0}$, such that $T_{\mathfrak{s}}(\boldsymbol{s}, B) = \int_B t_{\mathfrak{s}}(\bar{\boldsymbol{s}} | \boldsymbol{s}) d\bar{\boldsymbol{s}}$.

A familiar class of discrete-time Markov processes is that of stochastic dynamical systems. If $\{\boldsymbol{\zeta}(t),\ t \in \mathbb{N}\}$ is a sequence of independent and identically distributed (iid) random variables taking values in $\mathbb{R}^n$, and $\boldsymbol{f} : \mathcal{S} \times \mathbb{R}^n \to \mathcal{S}$ is a measurable map, then the recursive equation

$$\boldsymbol{s}(t+1) = \boldsymbol{f}(\boldsymbol{s}(t), \boldsymbol{\zeta}(t)), \quad \forall t \in \mathbb{N}, \quad \boldsymbol{s}(0) = \boldsymbol{s}_0 \in \mathcal{S}, \tag{1}$$

induces a Markov process that is characterized by the kernel

$$T_{\mathfrak{s}}(\boldsymbol{s}, B) = T_{\zeta}\left(\boldsymbol{\zeta} \in \mathbb{R}^n \ : \ \boldsymbol{f}(\boldsymbol{s}, \boldsymbol{\zeta}) \in B\right),$$

where T_{ζ} is the distribution of the r.v. $\boldsymbol{\zeta}(0)$ (in fact, of any $\boldsymbol{\zeta}(t)$ since these are iid random variables). In other words, the map $\boldsymbol{f}$ together with the distribution of the r.v. $\{\boldsymbol{\zeta}(t)\}$ uniquely define the stochastic kernel of the process. The converse is also true as shown in [13, Proposition 7.6]: any discrete-time Markov process $\mathscr{M}_{\mathfrak{s}}$ admits a dynamical representation as in (1), for an appropriate selection of function $\boldsymbol{f}$ and distribution of the r.v. $\{\boldsymbol{\zeta}(t)\}$.

Let us expand the dynamical equation (1) explicitly over its states $\boldsymbol{s} = [s_1, \ldots, s_n]^T$, map components $\boldsymbol{f} = [f_1, \ldots, f_n]^T$, and uncertainly terms $\boldsymbol{\zeta} = [\zeta_1, \ldots, \zeta_n]^T$, as follows:

$$\begin{aligned} s_1(t+1) &= f_1(s_1(t), s_2(t), \ldots, s_n(t), \zeta_1(t)), \\ s_2(t+1) &= f_2(s_1(t), s_2(t), \ldots, s_n(t), \zeta_2(t)), \\ &\vdots \\ s_n(t+1) &= f_n(s_1(t), s_2(t), \ldots, s_n(t), \zeta_n(t)). \end{aligned} \tag{2}$$

In this article we are interested in exploiting the knowledge of the structure of the dynamics in (2) for formal verification via abstractions [1, 7, 8]. We focus our attention to continuous (unbounded and uncountable) Euclidean spaces $\mathcal{S} = \mathbb{R}^n$, and further assume that for any $t \in \mathbb{N}$, $\zeta_k(t)$ are independent for all $k \in \mathbb{N}_n$. This latter assumption is widely used in the

[1] A metric space $\mathcal{S}$ is called separable if it has a countable dense subset.

theory of dynamical systems, and allows for the following multiplicative structure on the conditional density function of the process:

$$t_{\mathfrak{s}}(\bar{\boldsymbol{s}}|\boldsymbol{s}) = t_1(\bar{s}_1|\boldsymbol{s})t_2(\bar{s}_2|\boldsymbol{s})\dots t_n(\bar{s}_n|\boldsymbol{s}), \tag{3}$$

where the function $t_k : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}_{\geq 0}$ solely depends on the map f_k and the distribution of ζ_k. The reader is referred to Section 5 for the detailed computation of the functions t_k from the dynamical equations in (2).

▶ Remark. The results of this article are presented under the structural assumption that $\zeta_k(\cdot)$ are independent over $k \in \mathbb{N}_n$. These results can be generalized to a broader class of processes by allowing inter-dependencies between the entries of the process noise, which requires partitioning the set of entries of $\boldsymbol{\zeta}(\cdot)$ so that any two entries from different partition sets are independent, whereas entries within a partition set may still be dependent. This assumption induces a multiplicative structure on $t_{\mathfrak{s}}(\bar{\boldsymbol{s}}|\boldsymbol{s})$ with respect to the partition, which is similar to (3). The finer the partition, the more efficient is our abstraction process.

▶ **Example 1.** Figure 1 shows a system of n masses connected by springs and dampers. For $i \in \mathbb{N}_n$, block i has mass m_i, the i^{th} spring has stiffness k_i, and the i^{th} damper has damping coefficient b_i. The first mass is connected to a fixed wall by the left-most spring/damper connection. All other masses are connected to the previous mass with a spring and a damper. A force ζ_i is applied to each mass, modeling the effect of a disturbance or of process noise. The dynamics of the overall system is comprised of the position and velocity of the blocks. It can be shown that the dynamics in discrete time take the form $\boldsymbol{s}(t+1) = \Phi\boldsymbol{s}(t) + \boldsymbol{\zeta}(t)$, where $\boldsymbol{s}(t) \in \mathbb{R}^{2n}$ with $s_{2i-1}(t), s_{2i}(t)$ indicating the velocity and position of mass i. The state transition matrix $\Phi = [\Phi_{ij}]_{i,j} \in \mathbb{R}^{2n\times 2n}$ is a band matrix with lower and upper bandwidth 3 and 2, respectively ($\Phi_{ij} = 0$ for $j < i-3$ and for $j > i+2$). ◀

▶ **Example 2.** A second example of structured dynamical systems is a discrete-time large-scale interconnected system. Consider an interconnected system of $N_{\mathfrak{d}}$ heterogeneous linear time-invariant (LTI) subsystems described by the following stochastic difference equations:

$$\boldsymbol{s}_i(t+1) = \Phi_i\boldsymbol{s}_i(t) + \sum_{j\in N_i} G_{ij}\boldsymbol{s}_j(t) + B_i\boldsymbol{u}_i(t) + \boldsymbol{\zeta}_i(t),$$

where $i \in \mathbb{N}_{N_{\mathfrak{d}}}$ denotes the i^{th} subsystem and $\boldsymbol{s}_i \in \mathbb{R}^{n\times 1}, \boldsymbol{u}_i \in \mathbb{R}^{p\times 1}, \boldsymbol{\zeta}_i \in \mathbb{R}^{m\times 1}$ are the state, the input, and the process noise of subsystem i. The term $\sum_{j\in N_i} G_{ij}\boldsymbol{s}_j(t)$ represents the physical interconnection between the subsystems where N_i, $|N_i| \ll N_{\mathfrak{d}}$, is the set of subsystems to which system i is physically connected. The described interconnected system can be found in many application areas including smart power grids, traffic systems, and sensor-actuator networks [11]. ◀

2.2 Probabilistic Invariance

We focus on verifying probabilistic invariance, which plays a central role in verifying properties of a system expressed as PCTL formulae or as linear temporal specifications [5, 22, 24].

▶ **Definition 3** (Probabilistic Invariance)**.** Consider a bounded Borel set $A \in \mathcal{B}$, representing a set of safe states. The finite-horizon *probabilistic invariance problem* asks to compute the probability that a trajectory of $\mathscr{M}_{\mathfrak{s}}$ associated with an initial condition $\boldsymbol{s}_0$ remains within the set A during the finite time horizon N:

$$p_N(\boldsymbol{s}_0, A) = \mathcal{P}\{\boldsymbol{s}(t) \in A \text{ for all } t = 0, 1, 2, \dots, N | \boldsymbol{s}(0) = \boldsymbol{s}_0\}.$$

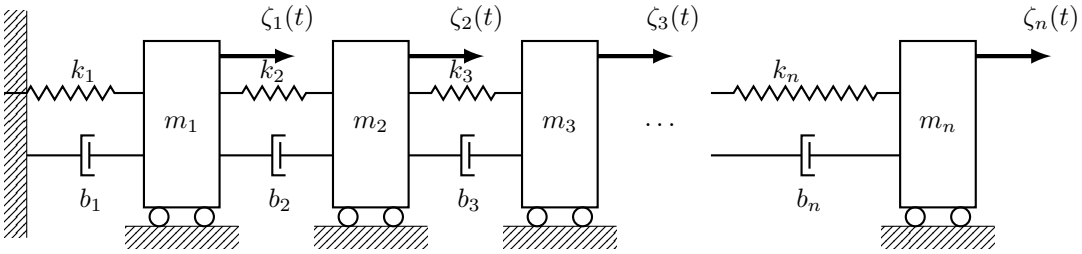


Figure 1 n-body mass-spring-damper system.

This quantity allows us to extend the result to a general probability distribution $\pi : \mathcal{B} \to [0, 1]$ for the initial state $\boldsymbol{s}(0)$ of the system as

$$\mathcal{P}\{\boldsymbol{s}(t) \in A \text{ for all } t = 0, 1, 2, \ldots, N\} = \int_{\mathcal{S}} p_N(\boldsymbol{s}_0, A)\pi(d\boldsymbol{s}_0). \tag{4}$$

Solution of the probabilistic invariance problem can be characterized via the value functions $V_k : \mathcal{S} \to [0, 1]$, $k = 0, 1, 2, \ldots, N$, defined by the following Bellman backward recursion [1]:

$$V_k(\boldsymbol{s}) = \mathbf{1}_A(\boldsymbol{s}) \int_A V_{k+1}(\bar{\boldsymbol{s}}) t_{\boldsymbol{s}}(\bar{\boldsymbol{s}}|\boldsymbol{s}) d\bar{\boldsymbol{s}} \quad \text{for} \quad k = 0, 1, 2, \ldots, N-1. \tag{5}$$

This recursion is initialized with $V_N(\boldsymbol{s}) = \mathbf{1}_A(\boldsymbol{s})$, where $\mathbf{1}_A(\boldsymbol{s})$ is the indicator function which is 1 if $\boldsymbol{s} \in A$ and 0 otherwise, and results in the solution $p_N(\boldsymbol{s}_0, A) = V_0(\boldsymbol{s}_0)$.

Equation (5) characterizes the finite-horizon probabilistic invariance quantity as the solution of a dynamic programming problem. However, since its explicit solution is in general not available, the actual computation of the quantity $p_N(\boldsymbol{s}_0, A)$ requires N numerical integrations at each state in the set A. This is usually performed with techniques based on state-space discretization [6].

3 Formal Abstractions as Dynamic Bayesian Networks

3.1 Dynamic Bayesian Networks

A Bayesian network (BN) is a tuple $\mathfrak{B} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$. The pair $(\mathcal{V}, \mathcal{E})$ is a directed Acyclic Graph (DAG) representing the structure of the network. The nodes in $\mathcal{V}$ are (discrete or continuous) random variables and the arcs in $\mathcal{E}$ represent the dependence relationships among the random variables. The set $\mathcal{T}$ contains conditional probability distributions (CPD) in forms of tables or density functions for discrete and continuous random variables, respectively. In a BN, knowledge is represented in two ways: qualitatively, as dependences between variables by means of the DAG; and quantitatively, as conditional probability distributions attached to the dependence relationships. Each random variable $X_i \in \mathcal{V}$ is associated with a conditional probability distribution $\mathbb{P}(X_i|Pa(X_i))$, where $Pa(Y)$ represents the parent set of the variable $Y \in \mathcal{V}$: $Pa(Y) = \{X \in \mathcal{V} | (X, Y) \in \mathcal{E}\}$. A BN is called *two-layered* if the set of nodes $\mathcal{V}$ can be partitioned to two sets $\mathcal{V}_1, \mathcal{V}_2$ with the same cardinality such that only the nodes in the second layer $\mathcal{V}_2$ have an associated CPD.

A dynamic Bayesian network [15, 20] is a way to extend Bayesian networks to model probability distributions over collections of random variables $X(0), X(1), X(2), \ldots$ indexed by time t. A DBN[2] is defined to be a pair $(\mathfrak{B}_0, \mathfrak{B}_{\to})$, where $\mathfrak{B}_0$ is a BN which defines the

[2] The DBNs considered in this paper are stationary (the structure of the network does not change with

distribution of $X(0)$, and $\mathfrak{B}_{\rightarrow}$ is a two-layered BN that defines the transition probability distribution for $(X(t+1)|X(t))$.

3.2 DBNs as Representations of Markov Processes

We now show that any discrete-time Markov process $\mathscr{M}_{\mathfrak{s}}$ over $\mathbb{R}^n$ can be represented as a DBN $(\mathfrak{B}_0, \mathfrak{B}_{\rightarrow})$ over n continuous random variables. The advantage of the reformulation is that it makes the dependencies between random variables explicit.

The BN $\mathfrak{B}_0$ is trivial for a given initial state of the Markov process $\boldsymbol{s}(0) = \boldsymbol{s}_0$. The DAG of $\mathfrak{B}_0$ has the set of nodes $\{X_1, X_2, \ldots, X_n\}$ without any arc. The Dirac delta distribution located in the initial state of the process is assigned to each node of $\mathfrak{B}_0$.[3] The DAG for the two-layered BN $\mathfrak{B}_{\rightarrow} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ comprises a set of nodes $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$, with $\mathcal{V}_1 = \{X_1, X_2, \ldots, X_n\}$ and $\mathcal{V}_2 = \{\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_n\}$. Each arc in $\mathcal{E}$ connects a node in $\mathcal{V}_1$ to another node in $\mathcal{V}_2$; $(X_i, \bar{X}_j) \in \mathcal{E}$ if and only if $t_j(\bar{s}_j|\boldsymbol{s})$ is not a constant function of s_i. The set $\mathcal{T}$ assigns a CPD to each node $\bar{X}_j$ according to the density function $t_j(\bar{s}_j|\boldsymbol{s})$.

▶ **Example 4.** Consider the following stochastic linear dynamical system:

$$\boldsymbol{s}(t+1) = \Phi \boldsymbol{s}(t) + \boldsymbol{\zeta}(t) \quad t \in \mathbb{N}, \quad \boldsymbol{s}(0) = \boldsymbol{s}_0 = [s_{01}, s_{02}, \ldots, s_{0n}]^T, \tag{6}$$

where $\Phi = [a_{ij}]_{i,j}$ is the system matrix and $\boldsymbol{\zeta}(t) \sim \mathcal{N}(0, \Sigma)$ are independent Gaussian r.v. for any $t \in \mathbb{N}$. The covariance matrix Σ is assumed to be full rank. Consequently, a linear transformation can be employed to change the coordinates and obtain a stochastic linear system with a diagonal covariance matrix. Then without loss of generality we assume $\Sigma = diag([\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2])$, which clearly satisfies the independence assumption on the process noise raised in Section 2.1. Model (6) for a lower bidiagonal matrix Φ can be expanded as follows:

$$\begin{aligned}
s_1(t+1) &= a_{11}s_1(t) + \zeta_1(t) \\
s_2(t+1) &= a_{21}s_1(t) + a_{22}s_2(t) + \zeta_2(t) \\
s_3(t+1) &= a_{32}s_2(t) + a_{33}s_3(t) + \zeta_3(t) \\
&\vdots \\
s_n(t+1) &= a_{n(n-1)}s_{n-1}(t) + a_{nn}s_n(t) + \zeta_n(t),
\end{aligned}$$

where $\zeta_i(\cdot)$, $i \in \mathbb{N}_n$ are independent Gaussian r.v. $\mathcal{N}(0, \sigma_i^2)$. The conditional density function of the system takes the following form:

$$t_{\mathfrak{s}}(\bar{\boldsymbol{s}}|\boldsymbol{s}) = t_1(\bar{s}_1|s_1)t_2(\bar{s}_2|s_1, s_2)t_3(\bar{s}_3|s_2, s_3)\ldots t_n(\bar{s}_n|s_{n-1}, s_n).$$

The DAG of the two-layered BN $\mathfrak{B}_{\rightarrow}$ associated with this system is sketched in Figure 2 for $n = 4$. The BN $\mathfrak{B}_0$ has an empty graph on the set of nodes $\{X_1, \ldots, X_n\}$ with the associated Dirac delta density functions located at s_{0i}, $\delta_d(s_i(0) - s_{0i})$. ◀

---

the time index t). They have no input variables and are fully observable: the output of the DBN model equals to its state.

[3] For a general initial probability distribution $\pi : \mathcal{B} \rightarrow [0, 1]$, a set of arcs must be added to reflect its possible product structure. This construction is not important at the current stage because of the backward recursion formulation of the probabilistic safety (please refer to (4) in Section 2.2).

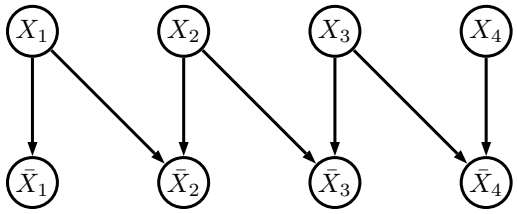


Figure 2 Two-layered BN $\mathfrak{B}_{\rightarrow}$ associated with the stochastic linear dynamical system in (6) for $n = 4$.

3.3 Finite Abstraction of Markov Processes as Discrete DBNs

Let $A \in \mathcal{B}$ be a bounded Borel set of safe states. We abstract the structured Markov process $\mathscr{M}_{\mathfrak{s}}$ interpreted in the previous section as a DBN with continuous variables to a DBN with discrete random variables. Our abstraction is relative to the set A. Algorithm 1 provides the steps of the abstraction procedure. It consists of discretizing each dimension into a finite number of bins.

In Algorithm 1, the projection operators $\Pi_i : \mathbb{R}^n \to \mathbb{R}$, $i \in \mathbb{N}_n$, are defined as $\Pi_i(\boldsymbol{s}) = s_i$ for any $\boldsymbol{s} = [s_1, \ldots, s_n]^T \in \mathbb{R}^n$. These operators are used to project the safe set A over different dimensions, $D_i \doteq \Pi_i(A)$. In step 2 of the Algorithm, set D_i is partitioned as $\{D_{ij}\}_{j=1}^{n_i}$ (for any $i \in \mathbb{N}_n$, D_{ij}'s are arbitrary but non-empty, non-intersecting, and $D_i = \cup_{j=1}^{n_i} D_{ij}$). The corresponding representative points $z_{ij} \in D_{ij}$ are also chosen arbitrarily. Step 5 of the algorithm constructs the support of the random variables in $\mathfrak{B}_{\rightarrow}$, $\mathcal{V} = \{X_i, \bar{X}_i, i \in \mathbb{N}_n\}$, and step 6 computes the discrete CPDs $T_i(\bar{X}_i | Pa(\bar{X}_i))$, reflecting the dependencies among the variables. For any $i \in \mathbb{N}_n$, $\Xi_i : Z_i \to 2^{D_i}$ represents a set-valued map that associates to any point $z_{ij} \in Z_i$ the corresponding partition set $D_{ij} \subset D_i$ (this is known as the "refinement map"). Furthermore, the abstraction map $\xi_i : D_i \to Z_i$ associates to any point $s_i \in D_i$ the corresponding discrete state in Z_i. Additionally, notice that the absorbing states $\phi = \{\phi_1, \ldots, \phi_n\}$ are added to the definition of BN $\mathfrak{B}_{\rightarrow}$ so that the conditional probabilities $T_i(\bar{X}_i | Pa(\bar{X}_i))$ marginalize to one. The function $v(\cdot)$ used in step 6 acts on (possibly a set of) random variables and provides their instantiation. In other words, the term $v(Pa(\bar{X}_i))$ that is present in the conditioned argument of t_i leads to evaluate function $t_i(\bar{s}_i|\cdot)$ at the instantiated values of $Pa(\bar{X}_i)$.

The construction of the DBN with discrete r.v. in Algorithm 1 is closely related to the Markov chain abstraction method in [1, 8]. The main difference lies in partitioning in each dimension separately instead of doing it for the whole state space. Absorbing states are also assigned to each dimension separately instead of having only one for the unsafe set. Moreover, Algorithm 1 stores the transition probabilities efficiently as a BN.

3.4 Probabilistic Invariance for the Abstract DBN

We extend the use of $\mathbb{P}$ by denoting the probability measure on the set of events defined over a DBN with discrete r.v. $\boldsymbol{z} = (X_1, X_2, \ldots, X_n)$. Given a discrete set $Z_{\mathfrak{a}} \subset \prod_i \Omega_i$, the probabilistic invariance problem asks to evaluate the probability $p_N(\boldsymbol{z}_0, Z_{\mathfrak{a}})$ that a finite execution associated with the initial condition $\boldsymbol{z}(0) = \boldsymbol{z}_0$ remains within the set $Z_{\mathfrak{a}}$ during the finite time horizon $t = 0, 1, 2, \ldots, N$. Formally,

$$p_N(\boldsymbol{z}_0, Z_{\mathfrak{a}}) = \mathbb{P}(\boldsymbol{z}(t) \in Z_{\mathfrak{a}}, \text{ for all } t = 0, 1, 2, \ldots, N | \boldsymbol{z}(0) = \boldsymbol{z}_0).$$

Algorithm 1 Abstraction of model $\mathscr{M}_{\mathfrak{s}}$ as a DBN with $\mathfrak{B}_{\rightarrow} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ over discrete r.v.

Require: input model $\mathscr{M}_{\mathfrak{s}} = (\mathcal{S}, \mathcal{B}, T_{\mathfrak{s}})$, safe set A

1: Project safe set A in each dimension $D_i \doteq \Pi_i(A)$, $i \in \mathbb{N}_n$
2: Select finite n_i-dimensional partition of D_i as $D_i = \cup_{j=1}^{n_i} D_{ij}$, $i \in \mathbb{N}_n$
3: For each D_{ij}, select single representative point $z_{ij} \in D_{ij}$, $z_{ij} = \xi_i(D_{ij})$
4: Construct the DAG $(\mathcal{V}, \mathcal{E})$, with $\mathcal{V} = \{X_i, \bar{X}_i, i \in \mathbb{N}_n\}$ and $\mathcal{E}$ as per Section 3.2
5: Define $Z_i = \{z_{i1}, \ldots, z_{in_i}\}$, $i \in \mathbb{N}_n$, and take $\Omega_i = Z_i \cup \{\phi_i\}$ as the finite state space of two r.v. X_i and $\bar{X}_i$, ϕ_i being dummy variables as per Section 3.3
6: Compute elements of the set $\mathcal{T}$, namely CPD T_i related to the node $\bar{X}_i$, $i \in \mathbb{N}_i$, as

$$T_i(\bar{X}_i = z | v(Pa(\bar{X}_i))) = \begin{cases} \int_{\Xi_i(z)} t_i(\bar{s}_i | v(Pa(\bar{X}_i)))d\bar{s}_i, & z \in Z_i,\ v(Pa(\bar{X}_i)) \cap \phi = \emptyset \\ 1 - \sum\limits_{z \in Z_i} \int_{\Xi_i(z)} t_i(\bar{s}_i | v(Pa(\bar{X}_i)))d\bar{s}_i, & z = \phi_i,\ v(Pa(\bar{X}_i)) \cap \phi = \emptyset \\ 1, & z = \phi_i,\ v(Pa(\bar{X}_i)) \cap \phi \neq \emptyset \\ 0, & z \in Z_i,\ v(Pa(\bar{X}_i)) \cap \phi \neq \emptyset \end{cases}$$

Ensure: output DBN with $\mathfrak{B}_{\rightarrow} = (\mathcal{V}, \mathcal{E}, \mathcal{T})$ over discrete r.v.

This probability can be computed by a discrete analogue of the Bellman backward recursion (see [2] for details).

▶ **Theorem 5.** *Consider value functions $V_k^d : \prod_i \Omega_i \to [0,1]$, $k = 0, 1, 2, \ldots, N$, computed by the backward recursion*

$$V_k^d(\boldsymbol{z}) = \mathbf{1}_{Z_{\mathfrak{a}}}(\boldsymbol{z}) \sum_{\bar{\boldsymbol{z}} \in \prod_i \Omega_i} V_{k+1}^d(\bar{\boldsymbol{z}}) \mathbb{P}(\bar{\boldsymbol{z}} | \boldsymbol{z}) \quad k = 0, 1, 2, \ldots, N-1, \tag{7}$$

and initialized with $V_N^d(\boldsymbol{z}) = \mathbf{1}_{Z_{\mathfrak{a}}}(\boldsymbol{z})$. Then the solution of the invariance problem is characterized as $p_N(\boldsymbol{z}_0, Z_{\mathfrak{a}}) = V_0^d(\boldsymbol{z}_0)$.

The discrete transition probabilities $\mathbb{P}(\bar{\boldsymbol{z}} | \boldsymbol{z})$ in Equation (7) are computed by taking the product of the CPD in $\mathcal{T}$. More specifically, for any $\boldsymbol{z}, \bar{\boldsymbol{z}} \in \prod_i \Omega_i$ of the form $\boldsymbol{z} = (z_1, z_2, \ldots, z_n), \bar{\boldsymbol{z}} = (\bar{z}_1, \bar{z}_2, \ldots, \bar{z}_n)$ we have

$$\mathbb{P}(\bar{\boldsymbol{z}} | \boldsymbol{z}) = \prod_i T_i(\bar{X}_i = \bar{z}_i | Pa(\bar{X}_i) = \boldsymbol{z}).$$

Our algorithm for probabilistic invariance computes $p_N(\boldsymbol{z}_0, Z_{\mathfrak{a}})$ to approximate $p_N(\boldsymbol{s}_0, A)$, for suitable choices of $\boldsymbol{z}_0$ and $Z_{\mathfrak{a}}$ depending on $\boldsymbol{s}_0$ and A. The natural choice for the initial state is $\boldsymbol{z}_0 = (z_1(0), \ldots, z_n(0))$ with $z_i(0) = \xi_i(\Pi_i(\boldsymbol{s}_0))$. For A, the n-fold Cartesian product of the collection of the partition sets $\{D_{ij}\}$, $i \in \mathbb{N}_n$ generates a cover of A as

$$\begin{aligned} A &\subset \bigcup \{D_{1j}\}_{j=1}^{n_1} \times \{D_{2j}\}_{j=1}^{n_2} \times \ldots \times \{D_{nj}\}_{j=1}^{n_n} \\ &= \bigcup_{\boldsymbol{j}} \{D_{\boldsymbol{j}} | \boldsymbol{j} = (j_1, j_2, \ldots, j_n), D_{\boldsymbol{j}} \doteq D_{1j_1} \times D_{2j_2} \times \ldots \times D_{nj_n}\}. \end{aligned}$$

We define the safe set $Z_{\mathfrak{a}}$ of the DBN as

$$Z_{\mathfrak{a}} = \bigcup_{\boldsymbol{j}} \{(z_{1j_1}, z_{2j_2}, \ldots, z_{nj_n}), \text{ such that } A \cap D_{\boldsymbol{j}} \neq \emptyset \text{ for } \boldsymbol{j} = (j_1, j_2, \ldots, j_n)\}, \tag{8}$$

which is a discrete representation of the continuous set $\bar{A} \subset \mathbb{R}^n$

$$\bar{A} = \bigcup_{\boldsymbol{j}} \{D_{\boldsymbol{j}}, \text{ such that } \boldsymbol{j} = (j_1, j_2, \ldots, j_n), A \cap D_{\boldsymbol{j}} \neq \emptyset\}. \tag{9}$$

For instance $\bar{A}$ can be a finite union of hypercubes in $\mathbb{R}^n$ if the partition sets D_{ij} are intervals. It is clear that the set $\bar{A}$ is in general different form A.

There are thus two sources of error: first due to replacing A with $\bar{A}$, and second, due to the abstraction of the dynamics between the discrete outcome obtained by Theorem 5 and the continuous solution that results from (5). In the next section we provide a quantitative bound on the two sources of error.

3.5 Quantification of the Error due to Abstraction

Let us explicitly write the Bellman recursion (5) of the safety problem over the set $\bar{A}$:

$$W_N(\boldsymbol{s}) = \mathbf{1}_{\bar{A}}(\boldsymbol{s}), \quad W_k(\boldsymbol{s}) = \int_{\bar{A}} W_{k+1}(\bar{\boldsymbol{s}}) t_{\mathfrak{s}}(\bar{\boldsymbol{s}}|\boldsymbol{s}) d\bar{\boldsymbol{s}}, \quad k = 0, 1, 2, \ldots, N-1, \tag{10}$$

which results in $p_N(\boldsymbol{s}_0, \bar{A}) = W_0(\boldsymbol{s}_0)$. Theorem 6 characterizes the error due to replacing the safe set A by $\bar{A}$.

▶ **Theorem 6.** *Solution of the probabilistic invariance problem with the time horizon N and two safe sets $A, \bar{A}$ satisfies the inequality*

$$|p_N(\boldsymbol{s}_0, A) - p_N(\boldsymbol{s}_0, \bar{A})| \leq MN\mathcal{L}(A \Delta \bar{A}), \quad \forall \boldsymbol{s}_0 \in A \cap \bar{A},$$

where $M \doteq \sup\left\{t_{\mathfrak{s}}(\bar{\boldsymbol{s}}|\boldsymbol{s}) \middle| \boldsymbol{s}, \bar{\boldsymbol{s}} \in A \Delta \bar{A}\right\}$. $\mathcal{L}(B)$ denotes the Lebesgue measure of any set $B \in \mathcal{B}$ and $A \Delta \bar{A} \doteq (A \backslash \bar{A}) \cup (\bar{A} \backslash A)$ is the symmetric difference of the two sets $A, \bar{A}$.

The second contribution to the error is related to the discretization of Algorithm 1 which is quantified by posing regularity conditions on the dynamics of the process. The following Lipschitz continuity assumption restricts the generality of the density functions t_k characterizing the dynamics of model $\mathscr{M}_{\mathfrak{s}}$.

▶ **Assumption 1.** *Assume the density functions $t_k(\bar{s}_i|\cdot)$ are Lipschitz continuous with the finite positive d_{ij}*

$$|t_j(\bar{s}_j|\boldsymbol{s}) - t_j(\bar{s}_j|\boldsymbol{s'})| \leq d_{ij}|s_i - s_i'|,$$

with $\boldsymbol{s} = [s_1, \ldots, s_{i-1}, s_i, s_{i+1}, \ldots, s_n]$ and $\boldsymbol{s'} = [s_1, \ldots, s_{i-1}, s_i', s_{i+1}, \ldots, s_n]$, for all $s_k, s_k', \bar{s}_k \in D_k$, $k \in \mathbb{N}_n$, and for all $i, j \in \mathbb{N}_n$.

Note that Assumption 1 holds with $d_{ij} = 0$ if and only if $(X_i, \bar{X}_j) \notin \mathcal{E}$ in the DAG of the BN $\mathfrak{B}_{\rightarrow}$. Assumption 1 enables us to assign non-zero weights to the arcs of the graph and turn it into a weighted DAG. The non-zero weight $w_{ij} = d_{ij}\mathcal{L}(D_j)$ is assigned to the arc $(X_i, \bar{X}_j) \in \mathcal{E}$, for all $i, j \in \mathbb{N}_n$. We define the out-weight of the node X_i by $\mathcal{O}_i = \sum_{j=1}^n w_{ij}$ and the in-weight of the node $\bar{X}_j$ by $\mathcal{I}_j = \sum_{i=1}^n w_{ij}$.

▶ Remark. The above assumption implies Lipschitz continuity of the conditional density functions $t_j(\bar{s}_j|\boldsymbol{s})$. Since trivially $|s_i - s_i'| \leq \|\boldsymbol{s} - \boldsymbol{s'}\|$ for all $i \in \mathbb{N}_n$, we obtain

$$|t_j(\bar{s}_j|\boldsymbol{s}) - t_j(\bar{s}_j|\boldsymbol{s'})| \leq \mathcal{H}_j \|\boldsymbol{s} - \boldsymbol{s'}\| \quad \forall \boldsymbol{s}, \boldsymbol{s'} \in \bar{A}, \bar{s}_j \in D_j,$$

where $\mathcal{H}_j = \sum_{i=1}^n d_{ij}$. The density function $t_{\mathfrak{s}}(\bar{\boldsymbol{s}}|\boldsymbol{s})$ is also Lipschitz continuous if the density functions $t_j(\bar{s}_j|\boldsymbol{s})$ are bounded, but the boundedness assumption is not necessary for our result to hold.

Assumption 1 enables us to establish Lipschitz continuity of the value functions W_k in (10). This continuity property is essential in proving an upper bound on the discretization error of Algorithm 1, which is presented in Corollary 8.

▶ **Lemma 7.** *Consider the value functions $W_k(\cdot)$, $k = 0, 1, 2, \ldots, N$, employed in Bellman recursion (10) of the safety problem over the set $\bar{A}$. Under Assumption 1, these value functions are Lipschitz continuous*

$$|W_k(\boldsymbol{s}) - W_k(\boldsymbol{s}')| \leq \kappa \|\boldsymbol{s} - \boldsymbol{s}'\|, \quad \forall \boldsymbol{s}, \boldsymbol{s}' \in \bar{A},$$

for all $k = 0, 1, 2, \ldots, N$ with the constant $\kappa = \sum_{j=1}^{n} \mathcal{I}_j$, where $\mathcal{I}_j$ is the in-weight of the node $\bar{X}_j$ in the DAG of the BN $\mathfrak{B}_{\rightarrow}$.

▶ **Corollary 8.** *The following inequality holds under Assumption 1:*

$$|p_N(\boldsymbol{s}_0, A) - p_N(\boldsymbol{z}_0, Z_{\mathfrak{a}})| \leq MN\mathcal{L}(A \Delta \bar{A}) + N\kappa\delta \quad \forall \boldsymbol{s}_0 \in A,$$

where $p_N(\boldsymbol{z}_0, Z_{\mathfrak{a}})$ is the invariance probability for the DBN obtained by Algorithm 1. The initial state of the DBN is $\boldsymbol{z}_0 = (z_1(0), \ldots, z_n(0))$ with $z_i(0) = \xi_i(\Pi_i(\boldsymbol{s}_0))$. The set $Z_{\mathfrak{a}}$ and the constant M are defined in (8) and Theorem 6, respectively. The diameter of the partition of Algorithm 1 is defined and used as $\delta = \sup\{\|\boldsymbol{s} - \boldsymbol{s}'\|, \forall \boldsymbol{s}, \boldsymbol{s}' \in D_{\boldsymbol{j}}, \forall \boldsymbol{j} \;\; D_{\boldsymbol{j}} \subset \bar{A}\}$.

The second error term in Corollary 8 is a linear function of the partition diameter δ, which depends on all partition sets along different dimensions. We are interested in proving a dimension-dependent error bound in order to parallelize the whole abstraction procedure along different dimensions. The next theorem gives this dimension-dependent error bound.

▶ **Theorem 9.** *The following inequality holds under Assumption 1:*

$$|p_N(\boldsymbol{s}_0, A) - p_N(\boldsymbol{z}_0, Z_{\mathfrak{a}})| \leq MN\mathcal{L}(A \Delta \bar{A}) + N \sum_{i=1}^{n} \mathcal{O}_i \delta_i \quad \forall \boldsymbol{s}_0 \in A, \tag{11}$$

with the constants defined in Corollary 8. $\mathcal{O}_j$ is the out-weight of the node X_i in the DAG of the BN $\mathfrak{B}_{\rightarrow}$. The quantity δ_i is the maximum diameter of the partition sets along the i^{th} dimension $\delta_i = \sup\{|s_i - s_i'|, \forall s_i, s_i' \in D_{ij}, \forall j \in \mathbb{N}_{n_i}\}$.

For a given error threshold ϵ, we can select the set $\bar{A}$ and consequently the diameters δ_i such that $MN\mathcal{L}(A \Delta \bar{A}) + N \sum_{i=1}^{n} \mathcal{O}_i \delta_i \leq \epsilon$. Therefore, generation of the abstract DBN, namely selection of the partition sets $\{D_{ij},\, j \in \mathbb{N}_i\}$ (according to the diameter δ_i) and computation of the CPD, can be implemented in parallel. For a given ϵ and set $\bar{A}$, the cardinality of the state space $\Omega_i, i \in \mathbb{N}_n$, of the discrete random variable X_i and thus the size of the CPD T_i, grow linearly as a function of the horizon of the specification N.

4 Efficient Model Checking of the Finite-State DBN

Existing numerical methods for model checking DBNs with discrete r.v. transform the DBN into an explicit matrix representation [12, 19, 21], which defeats the purpose of a compact representation. Instead, we show that the multiplicative structure of the transition probability matrix can be incorporated in the computation which makes the construction of $\mathbb{P}(\bar{\boldsymbol{z}}|\boldsymbol{z})$ dispensable. For this purpose we employ *factor graphs* and the *sum-product algorithm* [17] originally developed for marginalizing functions and applied to belief propagation in Bayesian networks. Suppose that a *global* function is given as a product of *local* functions,

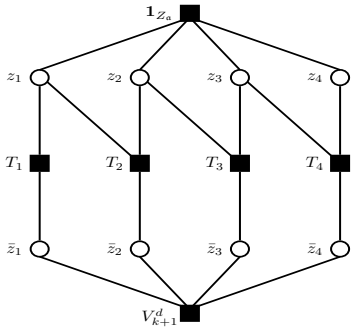


Figure 3 Factor graph of the linear stochastic system (6) for $n = 4$.

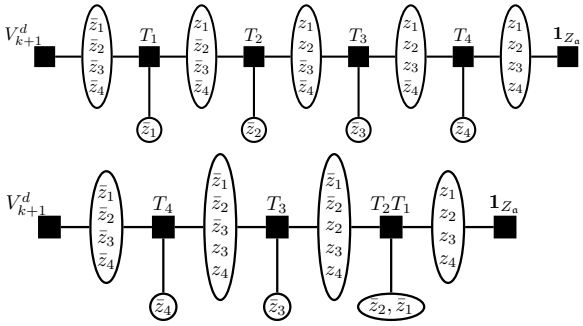


Figure 4 Spanning tree of the linear stochastic system in (6) for $n = 4$ and two orderings $(\bar{z}_4, \bar{z}_3, \bar{z}_2, \bar{z}_1)$ (top plot) and $(\bar{z}_1, \bar{z}_2, \bar{z}_3, \bar{z}_4)$ (bottom plot).

and that each local function depends on a subset of the variables of the global map. In its most general form, the sum-product algorithm acts on factor graphs in order to marginalize the global function, i.e., taking summation respect to a subset of variables, exploiting its product structure [17]. In our problem, we restrict the summation domain of the Bellman recursion (7) to $\prod_i Z_i$ because the value functions are simply equal to zero in the complement of this set. The summand in (7) has the multiplicative structure

$$g(\boldsymbol{z}, \bar{\boldsymbol{z}}) \doteq \mathbf{1}_{Z_a}(\boldsymbol{z}) V^d_{k+1}(\bar{\boldsymbol{z}}) \prod_i T_i(\bar{X}_i = \bar{z}_i | Pa(\bar{X}_i) = \boldsymbol{z}), \quad V^d_k(\boldsymbol{z}) = \sum_{\bar{\boldsymbol{z}} \in \prod_i Z_i} g(\boldsymbol{z}, \bar{\boldsymbol{z}}). \tag{12}$$

The function $g(\boldsymbol{z}, \bar{\boldsymbol{z}})$ depends on variables $\{z_i, \bar{z}_i, i \in \mathbb{N}_n\}$. The factor graph of $g(\boldsymbol{z}, \bar{\boldsymbol{z}})$ has $2n$ *variable nodes*, one for each variable and $(n+2)$ *function nodes* for local functions $\mathbf{1}_{Z_a}, V^d_{k+1}, T_i$. An arc connects a variable node to a function node if and only if the variable is an argument of the local function. The factor graph of Example 4 for $n = 4$ is presented in Figure 3 – factor graphs of general functions $g(\boldsymbol{z}, \bar{\boldsymbol{z}})$ in (12) are similar to that in Figure 3, the only part needing to be modified being the set of arcs connecting variable nodes $\{z_i, i \in \mathbb{N}_n\}$ and function nodes $\{T_i, i \in \mathbb{N}_n\}$. This part of the graph can be obtained from the DAG of $\mathfrak{B}_{\rightarrow}$ of the DBN.

The factor graph of a function $g(\boldsymbol{z}, \bar{\boldsymbol{z}})$ contains loops for $n \geq 2$ and must be transformed to a *spanning tree* using clustering and stretching transformations [17]. For this purpose the order of clustering function nodes $\{T_i, i \in \mathbb{N}_n\}$ and that of stretching variable nodes $\{z_i, i \in \mathbb{N}_n\}$ needs to be chosen. Figure 4 presents the spanning trees of the stochastic system in (6) for two such orderings. The variable nodes at the bottom of each spanning tree specify the order of the summation, whereas the function nodes considered from the left to the right indicate the order of multiplication of the local functions. The rest of the variable nodes show the arguments of the intermediate functions, which reflects the required memory for storing such functions. The computational complexity of the solution carried out on the spanning tree clearly depends on this ordering.

Algorithm 2 presents a greedy procedure that operates on the factor graph and provides an ordering of the variables and of the functions, in order to reduce the overall memory usage. This algorithm iteratively combines the function nodes and selects the next variable node, over which the summation is carried out. The output of this algorithm implemented on the factor graph of Example 4 is the orderings $\kappa_{\mathfrak{f}} = (\bar{z}_4, \bar{z}_3, \bar{z}_2, \bar{z}_1)$ and $e_{\mathfrak{f}} = (T_4, T_3, T_2, T_1)$, started from the outermost sum, which is related to the spanning tree on top of Figure 4.

Algorithm 2 Greedy algorithm for obtaining the order of stretching variables and clustering functions in the factor graph

Require: Factor graph of the summand in Bellman recursion

1: Initialize the sets $\mathcal{U}_1 = \{z_i,\, i \in \mathbb{N}_n\}$, $\mathcal{U}_2 = \{\bar{z}_i,\, i \in \mathbb{N}_n\}$, $\mathcal{U}_3 = \{T_i,\, i \in \mathbb{N}_n\}$, $e_{\mathfrak{f}} = \kappa_{\mathfrak{f}} = \emptyset$
2: **while** $\mathcal{U}_1 \neq \emptyset$ **do**
3: For any node $u \in \mathcal{U}_3$ compute $Pa_{\mathfrak{f}}(u)$ (resp. $Ch_{\mathfrak{f}}(u)$) as the elements of $\mathcal{U}_1$ (resp. $\mathcal{U}_2$) connected to u by an arc in the factor graph
4: Define the equivalence relation R on $\mathcal{U}_3$ as $uR\bar{u}$ iff $Pa_{\mathfrak{f}}(u) = Pa_{\mathfrak{f}}(\bar{u})$
5: Replace the set $\mathcal{U}_3$ with the set of equivalence classes induced by R.
6: Combine all the variable nodes of $Ch_{\mathfrak{f}}(u)$ connected to one class
7: Select $u \in \mathcal{U}_3$ with the minimum cardinality of $Pa_{\mathfrak{f}}(u)$ and put $e_{\mathfrak{f}} = (u, e_{\mathfrak{f}}), \kappa_{\mathfrak{f}} = (Ch_{\mathfrak{f}}(u), \kappa_{\mathfrak{f}})$
8: Update the sets $\mathcal{U}_1 = \mathcal{U}_1 \backslash Pa_{\mathfrak{f}}(u)$, $\mathcal{U}_2 = \mathcal{U}_2 \cup Pa_{\mathfrak{f}}(u) \backslash Ch_{\mathfrak{f}}(u)$, $\mathcal{U}_3 = \mathcal{U}_3 \backslash \{u\}$, and eliminate all the arcs connected to u
9: **end while**

Ensure: The order of variables $\kappa_{\mathfrak{f}}$ and functions $e_{\mathfrak{f}}$

5 Comparison with the State of the Art

In this section we compare our approach with the state-of-the-art abstraction procedure presented in [1] (referred to as AKLP in the following), which does not exploit the structure of the dynamics. The AKLP algorithm approximates the concrete model with a finite-state Markov chain by uniformly gridding the safe set. As in our work, the error bound of the AKLP procedure depends on the global Lipschitz constant of the density function of the model, however it does not exploit its structure as proposed in this work. We compare the two procedures on (1) error bounds and (2) computational resources.

Consider the stochastic linear dynamical model in (6), where $\Phi = [a_{ij}]_{i,j}$ is an arbitrary matrix. The Lipschitz constants d_{ij} in Assumption 1 can be computed as $d_{ij} = |a_{ji}|/\sigma_j^2\sqrt{2\pi e}$, where e is Euler's constant. From Theorem 9, we get the following error bound:

$$e_{\mathrm{DBN}} \doteq MN\mathcal{L}(A\Delta\bar{A}) + \frac{N}{\sqrt{2\pi e}} \sum_{i,j=1}^{n} \frac{|a_{ji}|}{\sigma_j^2} \mathcal{L}(D_j)\delta_i.$$

On the other hand, the error bound for AKLP is

$$e_{\mathrm{AKLP}} = MN\mathcal{L}(A\Delta\bar{A}) + \frac{Ne^{-1/2}}{(\sqrt{2\pi})^n \sigma_1\sigma_2\ldots\sigma_n} \|\Sigma^{-1/2}\Phi\|_2 \delta\mathcal{L}(A).$$

In order to meaningfully compare the two error bounds, select set $A = [-\alpha, \alpha]^n$ and $\sigma_i = \sigma, i \in \mathbb{N}_n$, and consider hypercubes as partition sets. The two error terms then become

$$e_{\mathrm{DBN}} = \varsigma n\eta \left(\frac{\|\Phi\|_1}{n\sqrt{n}}\right), \quad e_{\mathrm{AKLP}} = \varsigma\eta^n \|\Phi\|_2, \quad \eta = \frac{2\alpha}{\sigma\sqrt{2\pi}}, \quad \varsigma = \frac{N\delta}{\sigma\sqrt{e}},$$

where $\|\Phi\|_1$ and $\|\Phi\|_2$ are the entry-wise one-norm and the induced two-norm of matrix Φ, respectively. The error e_{AKLP} depends exponentially on the dimension n as η^n, whereas we have reduced this term to a linear one ($n\eta$) in our proposed new approach resulting in error e_{DBN}. Note that $\eta \leq 1$ means that the standard deviation of the process noise is larger than

Table 1 Comparison of the AKLP and the DBN-based algorithms, over the stochastic linear dynamical model (6). The number of partition sets (or bins) per dimension, the number of marginals, and the total required number of (addition and multiplication) operations for the verification step, are compared for models of different dimensions (number of continuous variables n).

dimension n		1	2	3	4	5	6	7	8
# bins/dim	AKLP	1.2×10^3	1.1×10^4	6.0×10^4	2.9×10^5	1.3×10^6	5.8×10^6	2.5×10^7	1.1×10^8
	DBN	1.2×10^3	3.6×10^3	6.0×10^3	8.5×10^3	1.1×10^4	1.3×10^4	1.6×10^4	1.8×10^4
# marginals	AKLP	1.5×10^6	1.5×10^{16}	4.8×10^{28}	4.8×10^{43}	1.5×10^{61}	1.5×10^{81}	4.3×10^{103}	3.5×10^{128}
	DBN	1.5×10^6	4.8×10^{10}	4.4×10^{11}	1.8×10^{12}	5.2×10^{12}	1.2×10^{13}	2.3×10^{13}	4.2×10^{13}
# operations	AKLP	2.9×10^7	3.1×10^{17}	1.0×10^{30}	1.1×10^{45}	3.7×10^{62}	3.7×10^{82}	1.1×10^{105}	9.5×10^{129}
	DBN	2.9×10^7	1.9×10^{12}	8.0×10^{16}	3.5×10^{21}	1.7×10^{26}	8.9×10^{30}	5.2×10^{35}	3.4×10^{40}

the selected safe set: in this case the value functions (which characterize the probabilistic invariance problem) uniformly converge to zero with rate η^n; clearly the case of $\eta > 1$ is more interesting. On the other hand for any matrix Φ we have $\frac{\|\Phi\|_1}{n\sqrt{n}} \leq \|\Phi\|_2$. This second term indicates how sparsity is reflected in the error computation. Denote by r the degree of connectivity of the DAG of $\mathfrak{B}_{\rightarrow}$ for this linear system, which is the maximum number of non-zero elements in rows of matrix Φ. We adapt the following inequalities from [16] for the norms of matrix Φ:

$$\|\Phi\|_2 \leq \sqrt{nr} \max_{i,j} |a_{ij}|, \qquad \frac{\|\Phi\|_1}{n\sqrt{n}} \leq \frac{r}{\sqrt{n}} \max_{i,j} |a_{ij}|,$$

which shows that for a fixed dimension n, sparse dynamics, compared to fully connected dynamics, results in better error bounds in the new approach.

In order to compare computational resources, consider the numerical values $N = 10$, $\alpha = 1$, $\sigma = 0.2$, and the error threshold $\epsilon = 0.2$ for the lower bidiagonal matrix Φ with all the non-zero entries set to one. Table 1 compares the number of required partition sets (or bins) per dimension, the number of marginals, and the required number of (addition and multiplication) operations for the verification step, for models of different dimensions (number of continuous variables n). The numerical values in Table 1 confirm that for a given upper bound on the error ϵ, the number of bins per dimension and the required marginals grow exponentially in dimension for AKLP and polynomially for our DBN-based approach. For instance, to ensure the error is at most ϵ for the model of dimension $n = 4$, the cardinality of the partition of each dimension for the uniform gridding and for the structured approach is 2.9×10^5 and 8.5×10^3, respectively. Then, AKLP requires storing 4.8×10^{43} entries (which is infeasible!), whereas the DBN approach requires 1.8×10^{12} entries ($\sim$ 8GB). The number of operations required for computation of the safety probability are 1.1×10^{45} and 3.5×10^{21}, respectively. This shows a substantial reduction in memory usage and computational time effort: with given memory and computational resources, the DBN-based approach in compare with AKLP promises to handle systems with dimension that is at least twice as large.

6 Conclusions and Future Directions

While we have focused on probabilistic invariance, our abstraction approach can be extended to more general properties expressed within the bounded-horizon fragment of PCTL [22] or to bounded-horizon linear temporal properties [24, 25], since the model checking problem for these logics reduce to computations of value functions similar to the Bellman recursion scheme. Our focus in this paper has been the foundations of DBN-based abstraction for general Markov processes: factored representations, error bounds, and algorithms. We are currently

implementing these algorithms in the FAUST^2 tool [10], and scaling the algorithms using dimension-dependent adaptive gridding [8] as well as implementations of the sum-product algorithm on top of data structures such as algebraic decision diagrams (as in probabilistic model checkers [18]).

References

1 A. Abate, J.-P. Katoen, J. Lygeros, and M. Prandini. Approximate model checking of stochastic hybrid systems. *European Journal of Control*, 6:624–641, 2010.

2 A. Abate, M. Prandini, J. Lygeros, and S. Sastry. Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. *Automatica*, 44(11):2724–2734, 2008.

3 A. Abate, S. Vincent, R. Dobbe, A. Silletti, N. Master, J. Axelrod, and C.J. Tomlin. A mechanical modeling framework for the study of epithelial morphogenesis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(6):1607–1620, Nov 2012.

4 J. Angeles. *Dynamic Response of Linear Mechanical Systems – Modeling, Analysis and Simulation*. Springer US, 2012.

5 C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

6 D.P. Bertsekas. Convergence of discretization procedures in dynamic programming. *IEEE Transactions on Automatic Control*, 20(3):415–419, 1975.

7 S. Esmaeil Zadeh Soudjani and A. Abate. Adaptive gridding for abstraction and verification of stochastic hybrid systems. In *QEST*, pages 59–69, 2011.

8 S. Esmaeil Zadeh Soudjani and A. Abate. Adaptive and sequential gridding procedures for the abstraction and verification of stochastic processes. *SIAM Journal on Applied Dynamical Systems*, 12(2):921–956, 2013.

9 S. Esmaeil Zadeh Soudjani, A. Abate, and R. Majumdar. Dynamic bayesian networks as formal abstractions of structured stochastic processes. *26th Conference on Concurrency Theory*, 2015. arXiv:1507.00509.

10 S. Esmaeil Zadeh Soudjani, C. Gevaerts, and A. Abate. FAUST^2: Formal abstractions of uncountable-state stochastic processes. In *TACAS*, volume 9035 of *LNCS*, pages 272–286. Springer, 2015.

11 A. Gusrialdi and S. Hirche. Communication topology design for large-scale interconnected systems with time delay. In *American Control Conference*, pages 4508–4513, June 2011.

12 S.K. Jha, E.M. Clarke, C.J. Langmead, A. Legay, A. Platzer, and P. Zuliani. A Bayesian approach to model checking biological systems. In *Computational Methods in Systems Biology*, volume 5688 of *LNCS*, pages 218–234. Springer, 2009.

13 O. Kallenberg. *Foundations of Modern Probability*. Probability and its Applications. Springer Verlag, New York, 2002.

14 J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *QEST*, pages 243–244. IEEE, 2005.

15 D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques – Adaptive Computation and Machine Learning*. The MIT Press, 2009.

16 L.Y. Kolotilina. Bounds for the singular values of a matrix involving its sparsity pattern. *Journal of Mathematical Sciences*, 137(3):4794–4800, 2006.

17 F.R. Kschischang, B.J. Frey, and H.-A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

18 M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

19 C.J. Langmead. Generalized queries and Bayesian statistical model checking in dynamic Bayesian networks: Application to personalized medicine. In *Proc. 8th International Conference on Computational Systems Bioinformatics*, pages 201–212, 2009.

20 K.P. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, UC Berkeley, Computer Science Division, 2002.

21 S.K. Palaniappan and P.S. Thiagarajan. Dynamic Bayesian networks: A factored model of probabilistic dynamics. In *ATVA*, volume 7561 of *LNCS*, pages 17–25. Springer, 2012.

22 F. Ramponi, D. Chatterjee, S. Summers, and J. Lygeros. On the connections between PCTL and dynamic programming. In *HSCC*, pages 253–262, 2010.

23 V. Sundarapandian. Distributed control schemes for large-scale interconnected discrete-time linear systems. *Mathematical and Computer Modelling*, 41(2–3):313–319, 2005.

24 I. Tkachev and A. Abate. Formula-free finite abstractions for linear temporal verification of stochastic hybrid systems. In *HSCC*, pages 283–292, 2013.

25 I. Tkachev, A. Mereacre, J.-P. Katoen, and A. Abate. Quantitative automata-based controller synthesis for non-autonomous stochastic hybrid systems. In *HSCC*, pages 293–302, 2013.

On Frequency LTL in Probabilistic Systems

Vojtěch Forejt[1] and Jan Krčál[2]

1 **Department of Computer Science, University of Oxford, UK**
2 **Saarland University – Computer Science, Saarbrücken, Germany**

Abstract

We study frequency linear-time temporal logic (fLTL) which extends the linear-time temporal logic (LTL) with a path operator $\mathbf{G}^p$ expressing that on a path, certain formula holds with at least a given frequency p, thus relaxing the semantics of the usual $\mathbf{G}$ operator of LTL. Such logic is particularly useful in probabilistic systems, where some undesirable events such as random failures may occur and are acceptable if they are rare enough. Frequency-related extensions of LTL have been previously studied by several authors, where mostly the logic is equipped with an extended "until" and "globally" operator, leading to undecidability of most interesting problems.

For the variant we study, we are able to establish fundamental decidability results. We show that for Markov chains, the problem of computing the probability with which a given fLTL formula holds has the same complexity as the analogous problem for LTL. We also show that for Markov decision processes the problem becomes more delicate, but when restricting the frequency bound p to be 1 and negations not to be outside any $\mathbf{G}^p$ operator, we can compute the maximum probability of satisfying the fLTL formula. This can be again performed with the same time complexity as for the ordinary LTL formulas.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Markov chains, Markov decision processes, LTL, controller synthesis

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.184

1 Introduction

Probabilistic verification is a vibrant area of research that aims to formally check properties of stochastic systems. Among the most prominent formalisms, with applications in e.g. modelling of network security protocols [19] or randomised algorithms [17], are Markov chains and Markov decision processes (MDPs). Markov chains are apt for modelling systems that contain purely stochastic behaviour, for example random failures, while MDPs can also express nondeterminism, most commonly present as decisions of a controller or dually as adversarial events in the system.

More technically, MDP is a process that moves in discrete steps within a finite state space (labelled by sets of atomic propositions). Its evolution starts in a given initial state s_0. In each step a *controller* chooses an action a_i from a finite set $A(s_i)$ of actions available in the current state s_i. The next state s_{i+1} is then chosen randomly according to a fixed probability distribution $\Delta(s_i, a_i)$. The controller may base its choice on the previous evolution $s_0 a_0 \dots a_{i-1} s_i$ and may also choose the action randomly. A Markov chain is an MDP where the set $A(s)$ is a singleton for each state s.

For the systems modelled as Markov chains or MDPs, the desired properties such as "whenever a signal arrives to the system, the system eventually switches off" can be often captured by a suitable linear-time logic. The most prominent one in the verification community is Linear Temporal Logic (LTL). Although LTL is suitable in many scenarios, it does not allow to capture some important linear-time properties, for example that a given

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 184–197

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

event takes place *sufficiently often*. The need for such properties becomes even more apparent in stochastic systems, in which probabilities often model random failures. Instead of requiring that no failure ever happens, it is natural to require that failures are infrequent, while still having the power of the LTL to specify these failures using a complex LTL formula.

A natural solution to the above problem is to extend LTL with operators that allow us to talk about *frequencies* of events. Adding such operators can easily lead to undecidability as they often allow one to encode values of potentially infinite counters [6, 7]. In both the above papers this is caused by a variant of a "frequency until" operator that talks about the ratio of the number of given events happening along a finite path. The undecidability results from [6, 7] carry over to the stochastic setting easily, and so, to avoid undecidability, care needs to be taken.

In this paper, we take an approach similar to [21] and in addition to usual operators of LTL such as $\mathbf{X}$, $\mathbf{U}$, $\mathbf{G}$ or $\mathbf{F}$ we only allow *frequency globally* formulae $\mathbf{G}^p\varphi$ that require the formula φ to hold on p-fraction of suffixes of an infinite path, or more formally, $\mathbf{G}^p\varphi$ is true on an infinite path $s_0a_0s_1a_1\ldots$ of an MDP if and only if

$$\liminf_{n\to\infty} \frac{1}{n} \cdot \Big|\{i \mid i < n \text{ and } s_i a_i s_{i+1} a_{i+1} \ldots \text{ satisfies } \varphi\}\Big| \geq p$$

This logic, which we call *frequency LTL (fLTL)*, is still a significant extension to LTL, and because all operators can be nested, it allows to express much larger class of properties (a careful reader will notice that *nesting* of frequency operators is not the main challenge when dealing with fLTL as it can be easily removed for the price of exponential blow-up of the size of the formula).

The problem studied in this paper asks, given a Markov chain and an fLTL formula, to compute the probability with which the formula is satisfied in the Markov chain when starting in the initial state. Analogously, for MDPs we study the *controller synthesis* problem which asks to compute the maximal probability of satisfying the formula, over all controllers.

For an example of possible application, suppose a network service accepts queries by immediately sending back responses, and in addition it needs to be switched off for maintenance during which the queries are not accepted. In most states, a new query comes in the next step with probability 0.5. In the waiting state, the system chooses either to wait further (action w), or to start a maintenance (action m) which takes one step to finish. The service is modelled as an MDP from Figure 1, leaving some parts of the behaviour unspecified. The aim is to synthesise a control strategy that meets with a given probability the requirements on the system. Example requirements can be given by a formula $\mathbf{G}\,\mathbf{F}\,\mathsf{m} \wedge \mathbf{G}\,\mathbf{F}\,(\mathsf{q} \to \mathbf{X}\,\mathsf{r})$ which will require that the service sometimes accepts the request, and sometimes goes for maintenance. However, there is no quantitative restriction on how often the maintenance can take place, and such restriction is inexpressible in LTL. However, in fLTL we can use the formula $\mathbf{G}\,\mathbf{F}\,\mathsf{m} \wedge \mathbf{G}^{0.95}(\mathsf{q} \to \mathbf{X}\,\mathsf{r})$ to restrict that the service is running sufficiently often, or a strong restriction $\mathbf{G}\,\mathbf{F}\,\mathsf{m} \wedge \mathbf{G}^{1}(\mathsf{q} \to \mathbf{X}\,\mathsf{r})$ saying that it is running with frequency 1. The formula may also contain several frequency operators. In order to push the frequency of correctly handled queries towards a bound p, the controller needs to choose to perform the maintenance less and less frequently during operation.

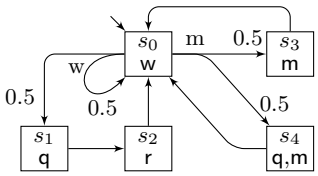


Figure 1 An example MDP.

Related work. Controller synthesis for ordinary LTL is a well-studied problem solvable in time polynomial in the size of the model and doubly exponential in the size of the

formula [2]. Usually, the LTL formula is transformed to an equivalent Rabin automaton, and the probability of reaching certain subgraphs is computed in a product of the MDP (or Markov Chain) with the automaton.

A similar approach is taken by [21]. They study a logic similar to our fLTL, where LTL is extended with a mean-payoff reward constraints in which the reward structures are determined by validity of given subformulas. The authors show that any formula can be converted to a variant of non-deterministic Büchi automata, called multi-threshold mean-payoff Büchi automata, with decidable emptiness problem, thus yielding decidability for model-checking and satisfiability problems of labelled transition systems. Results of [21] cannot be applied to probabilistic systems: here one needs to work with *deterministic automata*, but as pointed out in [21, Section 4, Footnote 4] the approach of [21] heavily relies on non-determinism, since reward values depend on complete future, and so deterministic "multi-threshold mean-payoff Rabin automata" are strictly less expressive than the logic. Another variant of frequency LTL was studied in [6, 7], in which also a modified until operator is introduced. The work [6] maintains boolean semantics of the logic, while in [7] the value of a formula is a number between 0 and 1. Both works obtain undecidability results for their logics, and [6] also yields decidability for restricted nesting. Another logic that speaks about frequencies on a finite interval was introduced in [20] but provides analysis algorithm only for a bounded fragment.

Significant attention has been given to the study of quantitative objectives. The work [5] adds mean-payoff objectives to temporal logics, but only as atomic propositions and not allowing more complex properties to be quantified. The work [3] extends LTL with another form of quantitative operators, allowing accumulated weight constraint expressed using automata, again not allowing quantification over complex formulas. [4] introduces lexicographically ordered mean-payoff objectives in non-stochastic parity games and [9] gives a polynomial time algorithm for almost-sure winning in MDPs with mean-payoff and parity objectives. These objectives do not allow to attach mean-payoff (i.e. frequencies) to properties more complex than atomic propositions. The solution to the problem requires infinite-memory strategy which at high level has a form similar to the form of strategies we construct for MDPs. Similar strategies also occur in [11, 10, 8] although each of these works deals with a fundamentally different problem.

In branching-time logics, CSL is sometimes equipped with a "steady-state" operator whose semantics is similar to our $\mathbf{G}^p$ (see e.g. [1]), and an analogous approach has been taken for the logic PCTL [16, 13]. In such logics every temporal subformula is evaluated over states, and thus the model-checking of a frequency operator can be directly reduced to achieving a single mean-payoff reward. This is contrasted with our setting in which the whole formula is evaluated over a single path, giving rise to much more complex behaviour.

Our contributions. To our best knowledge, this paper gives the first decidability results for probabilistic verification against linear-time temporal logics extended by frequency operators with *complex nested subformulas* of the logic.

We first give an algorithm for computing the probability of satisfying an fLTL formula in a Markov Chain. The algorithm works by breaking the fLTL formula into linearly many ordinary LTL formulas, and then off-the-shelf verification algorithms can be applied. We obtain that the complexity of fLTL model-checking is the same as the complexity of LTL model checking. Although the algorithm itself is very simple, some care needs to be taken when proving its correctness: as we explain later, the "obvious" proof approach would fail since some common assumptions on independence of events are not satisfied.

We then proceed with Markov decision processes, where we show that the controller synthesis problem is significantly more complex. Unlike the ordinary LTL, for fLTL the

controller-synthesis problem may require strategies to use *infinite memory*, even for very simple formulas. On the positive side, we give an algorithm for synthesis of strategies for formulas in which the negations are pushed to atomic propositions, and all the frequency operators have lower bound 1. Although this might appear to be a simple problem, it is not easily reducible to the problem for LTL, and the proof of the correctness of the algorithm is in fact very involved. This is partly because even if a strategy satisfies the formula, it can exhibit a very "insensible" behaviour, as long as this behaviour has zero frequency in the limit. In the proof, we need to identify these cases and eliminate them. Ultimately, our construction again yields the same complexity as the problem for ordinary LTL. We believe the contribution of the fragment is both practical, as it gives a "weaker" alternative of the $\mathbf{G}$ operator usable in controller synthesis, and theoretical, giving new insights into many of the challenges one will face in solving the controller-synthesis problem for the whole fLTL.

2 Preliminaries

We now proceed with introducing basic notions we use throughout this paper.

A *probability distribution* over a finite or countable set X is a function $d : X \to [0, 1]$ such that $\sum_{x \in X} d(x) = 1$, and $\mathcal{D}(X)$ denotes the set of all probability distributions over X.

Markov decision processes and Markov chains. A *Markov decision process* (MDP) is a tuple $\mathcal{M} = (S, A, \Delta)$ where S is a finite set of states, A is a finite set of actions, and $\Delta : S \times A \to \mathcal{D}(S)$ is a partial probabilistic transition function. A *Markov chain* (MC) is an MDP in which for every $s \in S$ there is exactly one a with $\Delta(s, a)$ being defined. We omit actions completely when we speak about Markov chains and no confusion can arise.

An infinite *path*, also called *run*, in $\mathcal{M}$ is a sequence $\omega = s_0 a_0 s_1 a_1 \cdots$ of states and actions such that $\Delta(s_i, a_i)(s_{i+1}) > 0$ for all i, and we denote by $\omega(i)$ the suffix $s_i a_i s_{i+1} a_{i+1} \cdots$. A finite path h, also called *history*, is a prefix of an infinite path ending in a state. Given a finite path $h = s_0 a_0 s_1 a_1 \cdots s_i$ and a finite or infinite path $h' = s_i a_i s_{i+1} a_{i+1} \cdots$ we use $h \cdot h'$ to denote the concatenated path $s_0 a_0 s_1 a_1 \cdots$. The set of paths starting with a prefix h is denoted by $\mathit{Cyl}(h)$, or simply by h if it leads to no confusion. We overload the notation also for sets of histories, we simply use H instead of $\bigcup_{h \in H} \mathit{Cyl}(h)$.

A *strategy* is a function σ that to every finite path h assigns a probability distribution over actions such that if an action a is assigned a non-zero probability, then $\Delta(s, a)$ is defined where s denotes the last state in h. A strategy σ is *deterministic* if it assigns Dirac distribution to any history, and *randomised* otherwise. Further, it is *memoryless* if its choice only depends on the last state of the history, and *finite-memory* if there is a finite automaton such that σ only makes its choice based on the state the automaton ends in after reading the history.

An MDP $\mathcal{M}$, a strategy σ and an initial state s_{in} give rise to a probability space $\mathbb{P}^{s_{in}}_{\sigma}$ defined in a standard way [15]. For a history h and a measurable set of runs U starting from the last state of h, we denote by $\mathbb{P}^{h}_{\sigma}(U)$ the probability $\mathbb{P}^{s_{in}}_{\sigma}(\{h \cdot \omega \mid \omega \in U\} \mid h)$. Similarly, for a random variable X we denote by $\mathbb{E}^{s_{in}}_{\sigma}(X)$ the expectation of X in this probability space and by $\mathbb{E}^{h}_{\sigma}(X)$ the expectation $\mathbb{E}^{s_{in}}_{\sigma}(X_h \mid h)$. Here, X_h is defined by $X_h(h \cdot \omega) = X(\omega)$ for runs of the form $h \cdot \omega$, and by $X_h(\omega') = 0$ for all other runs. We say that a property holds *almost surely* (or for almost all runs, or almost every run) if the probability of the runs satisfying the property is 1.

Frequency LTL. The syntax of *frequency LTL (fLTL)* is defined by the equation:

$$\varphi ::= \alpha \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\,\varphi \mid \varphi\,\mathbf{U}\,\varphi \mid \mathbf{G}^{p}\varphi$$

where α ranges over a set AP of atomic propositions. The logic LTL is obtained by omitting the rule for $\mathbf{G}^{p}\varphi$. For Markov chains we study the whole fLTL whereas for MDP, we restrict to a fragment that we call *1-fLTL*. In this fragment, negations only occur immediately preceding atomic propositions, and $\mathbf{G}^{p}$ operators occur only with $p = 1$.

For an infinite sequence $\gamma = x_1 x_2 \ldots$ of numbers, we set $\mathsf{freq}(\gamma) := \liminf_{i\to\infty} \frac{1}{i}\sum_{j=1}^{i} x_i$. Given a valuation $\nu : S \to 2^{AP}$, the semantics of fLTL is defined over a path $\omega = s_0 a_0 s_1 \ldots$ of an MDP as follows.

$\omega \models \alpha$	iff $\alpha \in \nu(s_0)$	$\omega \models \mathbf{X}\,\varphi$	iff $\omega(1) \models \varphi$
$\omega \models \neg\varphi$	iff $\omega \not\models \varphi$	$\omega \models \varphi_1\,\mathbf{U}\,\varphi_2$	iff $\exists k : \omega(k) \models \varphi_2 \wedge \forall \ell{<}k : \omega(\ell) \models \varphi_1$
$\omega \models \varphi_1 \vee \varphi_2$	iff $\omega{\models}\varphi_1$ or $\omega{\models}\varphi_2$	$\omega \models \mathbf{G}^{p}\varphi$	iff $\mathsf{freq}(\mathbf{1}_{\varphi,0}\mathbf{1}_{\varphi,1}\ldots) \geq p$

where $\mathbf{1}_{\varphi,i}$ is 1 for ω iff $\omega(i) \models \varphi$, and 0 otherwise. We define $\mathsf{true}, \mathsf{false}, \wedge$, and $\rightarrow$ by their usual definitions and introduce standard operators $\mathbf{F}$ and $\mathbf{G}$ by putting $\mathbf{F}\,\varphi \equiv \mathsf{true}\,\mathbf{U}\,\varphi$ and $\mathbf{G}\,\varphi \equiv \neg\mathbf{F}\,\neg\varphi$. Finally, we use $\mathbb{P}_\sigma(\varphi)$ as a shorthand for $\mathbb{P}_\sigma(\{\omega \mid \omega \models \varphi\})$.

▶ **Definition 1** (Controller synthesis)**.** The controller synthesis problem asks to decide, given an MDP $\mathcal{M}$, a valuation ν, an initial state s_{in}, an fLTL formula φ and a probability bound x, whether $\mathbb{P}^{s_{in}}_{\sigma}(\varphi) \geq x$ for some strategy σ.

As an alternative to the above problem, we can ask to compute the maximal possible x for which the answer is true. In the case of Markov chains, we speak about Satisfaction problem since there is no strategy to synthesise.

Rabin automata. A *(deterministic) Rabin automaton* is a tuple $R = (Q, q_{in}, \Sigma, \delta, \mathcal{F})$ where Q is a finite set of states, Σ is an input alphabet, $\delta : Q \times \Sigma \to Q$ is a transition function, and $\mathcal{F} \subseteq Q \times Q$ is an accepting condition. A computation of R on an infinite word $\varrho = a_0 a_1 \ldots$ over the alphabet Σ is the infinite sequence $R[\varrho] = q_0 q_1 \ldots$ with $q_0 = q_{in}$ and $\delta(q_i, a_i) = q_{i+1}$. A computation is accepting (or "R accepts ϱ") if there is $(E, F) \in \mathcal{F}$ such that all states of E occur only finitely many times in the computation, and some state of F occurs in it infinitely many times. For a run $\omega = s_0 a_0 s_1 a_1 \ldots$ and a valuation ν, we use $\nu(\omega)$ for the sequence $\nu(s_0)\nu(s_1)\ldots$ of sets of atomic propositions.

As a well known result [2], for every MDP $\mathcal{M}$, valuation ν and an LTL formula φ there is a Rabin automaton R over the alphabet 2^{AP} such that R is constructible in doubly exponential time and $\omega \models \varphi$ iff R accepts $\nu(\omega)$. We say that R is equivalent to φ. It is not clear whether this result and the definition of Rabin automata can be extended to work with fLTL in a way that would be useful for our goals. The reason for this is, as pointed out in [21, Section 4, Footnote 4], that the frequencies in fLTL depend on the future of a run, and so require non-determinism, which is undesirable in stochastic verification.

3 Satisfaction problem for Markov Chains

In this section we show how to solve the satisfaction problem for MCs and fLTL. Let us fix a MC $\mathcal{M} = (S, \Delta)$, an initial state s_{in} and fLTL formula ψ. We will use the notion of *bottom strongly connected component* (bscc) of $\mathcal{M}$, which is a set of states S' such that for all $s \in S'$ the set of states reachable from s is exactly S'. If s is in a bscc, by $\mathsf{bscc}(s)$ we denote the bscc containing s.

We first describe the algorithm computing the probability of satisfying ψ from s_{in}, and then prove its correctness.

The algorithm. The algorithm proceeds in the following steps. First, for each state contained in some bscc B, we compute the steady-state frequency x_s of s within B. It is the number $\mathbb{E}^s(\mathsf{freq}(\mathbf{1}_{s,0}\mathbf{1}_{s,1}\ldots))$ where $\mathbf{1}_{s,i}(\omega)$ equals 1 if the ith state of ω is s, and 0, otherwise. Afterwards, we repeat the following steps and keep modifying ψ for as long as it contains any $\mathbf{G}^p$ operators:

1. Let φ be a LTL formula and p a number such that ψ contains $\mathbf{G}^p\varphi$.
2. Compute $\mathbb{P}^s(\varphi)$ for every state s contained in some bscc.
3. Create a fresh atomic proposition $\alpha_{\varphi,p}$ which is true in a state s iff s is contained in a bscc and $\sum_{t\in\mathsf{bscc}(s)} x_t \cdot \mathbb{P}^t(\varphi) \geq p$.
4. Modify ψ by replacing any occurrence of $\mathbf{G}^p\varphi$ with $\mathbf{F}\,\alpha_{\varphi,p}$.

Once ψ contains no $\mathbf{G}^p$ operators, it is an LTL formula and we can use off-the-shelf techniques to compute $\mathbb{P}^{s_{in}}(\psi)$, which is our desired value.

Correctness. The correctness of the algorithm relies on the fact that $\alpha_{\varphi,p}$ labels states in a bscc B if almost every run reaching B satisfies the corresponding frequency constraint:

▶ **Proposition 2.** *For every LTL formula φ, every number p, every bscc B and almost every run ω that enters B we have $\omega \models \mathbf{G}^p\varphi$ if and only if $\sum_{t\in B} x_t \cdot \mathbb{P}^t(\varphi) \geq p$.*

The proposition might seem "obviously true", but the proof is not trivial. The main obstacle is that satisfactions of φ on $\omega(i)$ and $\omega(j)$ are *not independent* events in general: for example if $\varphi \equiv \mathbf{F}\,\alpha$ and $i < j$, then $\omega(j) \models \varphi$ implies $\omega(i) \models \varphi$. Hence we cannot apply the Strong law of large numbers (SLLN) for independent random variables or Ergodic theorem for Markov chains [18, Theorems 1.10.1-2], which would otherwise be obvious candidates. Nevertheless, we can use the following variant of SLLN for correlated events.

▶ **Lemma 3.** *Let $Y_0, Y_1 \ldots$ be a sequence of random variables which only take values 0 or 1 and have expectation μ. Assume there are $0 < r, c < 1$ such that for all $i, j \in \mathbb{N}$ we have $\mathbb{E}((Y_i - \mu)(Y_j - \mu)) \leq r^{\lfloor c|i-j|\rfloor}$. Then $\lim_{n\to\infty} \sum_{i=0}^{n} Y_i/n = \mu$ almost surely.*

Using the above lemma, we now prove Proposition 2 for fixed φ, B, p. Let R denote the Rabin automaton equivalent to φ and $\mathcal{M} \times R$ be the Markov chain product of $\mathcal{M}$ and R.

First, we say that a finite path $s_0 \ldots s_k$ of $\mathcal{M}$ is *determined* if the state q_k reached by R after reading $\nu(s_0 \ldots s_{k-1})$ satisfies that (s_k, q_k) is in a bscc of $\mathcal{M} \times R$. We point out that for a determined path $s_0 \ldots s_k$, either almost every run of $Cyl(s_0 \ldots s_k)$ satisfies φ, or almost no run of $Cyl(s_0 \ldots s_k)$ satisfies φ. Also, the probability of runs determined within k steps is at least $\sum_{i=0}^{\lfloor k/M\rfloor}(1-r^M)^i r^M = 1 - (1-r^M)^{\lfloor k/M\rfloor}$ where M is the number of states of $\mathcal{M} \times R$ and r is the minimum probability that occurs in $\mathcal{M} \times R$.

Now fix a state $s \in B$. For all $t \in B$ and $i \geq 0$ we define random variables X_i^t over runs initiated in s. We let $X_i^t(\omega)$ take value 1 if t is visited at least i times in ω and the suffix of ω starting from the ith visit to t satisfies φ. Otherwise, we let $X_i^t(\omega) = 0$. Note that all X_i^t have a common expected value $\mu_t = \mathbb{P}^t(\varphi)$.

Next, let i and j be two numbers with $i \leq j$. We denote by Ω the set of all runs and by D the set of runs ω for which the suffixes starting from the ith visit to t are determined before the jth visit to t (note that D can possibly be $\emptyset$). Because on these determined runs $\mathbb{E}^s(X_j^t - \mu_t \mid D) = 0$, we get

$$\mathbb{E}^s((X_i^t - \mu_t)(X_j^t - \mu_t)) \leq 1 - \mathbb{P}^s(D) \leq (1 - r^M)^{\lfloor (i-j)/M\rfloor}$$

as shown in [14]. Thus, Lemma 3 applies to the random variables X_i^t for a fixed t. Considering all $t \in B$ together, we show in [14] that $\mathsf{freq}(\mathbf{1}_{\varphi,0}\mathbf{1}_{\varphi,1}\ldots) = \sum_{t\in\mathsf{bscc}(s)} x_t \mathbb{P}^s(\varphi)$ for almost all runs initiated in the state s we fixed above. Because almost all runs that enter B eventually visit s, and because satisfaction of $\mathbf{G}^p\varphi$ is independent of any prefix, the proof of Proposition 2 is finished, and we can establish the following.

▶ **Theorem 4.** *The satisfaction problem for Markov chains and fLTL is solvable in time polynomial in the size of the model, and doubly exponential in the size of the formula.*

4 Controller synthesis for MDPs

We now proceed with the controller synthesis problem for MDPs and 1-fLTL. The problem for this restricted fragment of 1-fLTL is still highly non-trivial. In particular, it is not equivalent to synthesis for the *LTL* formula where every $\mathbf{G}^1$ is replaced with $\mathbf{G}$. Indeed, for satisfying any LTL formula, finite memory is sufficient, while for 1-fLTL, the following theorem shows that infinite memory may be necessary.

▶ **Theorem 5.** *There is a 1-fLTL formula ψ and a Markov decision process $\mathcal{M}$ with valuation ν such that the answer to the controller synthesis problem is "yes", but there is no finite-memory strategy witnessing this.*

Proof idea. Consider the MDP from Figure 1 together with the formula $\psi = \mathbf{G}\,\mathbf{F}\,\mathsf{m} \wedge \mathbf{G}^1(\mathsf{q} \to \mathbf{X}\,\mathsf{r})$. Independent of the strategy being used, no run initiated in s_4 satisfies the subformula $\mathsf{q} \to \mathbf{X}\,\mathsf{r}$, while every run initiated in any other state satisfies this subformula. This means that we need the frequency of visiting s_4 to be 0. The only finite-memory strategies achieving this are those that from some history on never choose to go right in the controllable state. However, under such strategies the formula $\mathbf{G}\,\mathbf{F}\,\mathsf{m}$ is not almost surely satisfied. On the other hand, the infinite-memory strategy that on i-th visit to s_0 picks m if and only if i is of the form 2^j for some j satisfies ψ.

Note that although the above formula requires infinite memory due to "conflicting" conjuncts, infinite memory is needed already for simpler formulae of the form $\mathbf{G}^1(\mathsf{a}\,\mathbf{U}\,\mathsf{b})$. ◀

The above result suggests that it is not possible to easily re-use verification algorithms for ordinary LTL. Nevertheless, our results allow us to establish the following theorem.

▶ **Theorem 6.** *The controller-synthesis problem for MDPs and 1-fLTL is solvable in time polynomial in the size of the model and doubly exponential in the size of the formula.*

For the rest of this section, in which we prove Theorem 6, we fix an MDP $\mathcal{M}$, valuation ν, an initial state s_{in}, and a 1-fLTL formula ψ. The proof is given in two parts. In the first part, in Section 4.1 we show that the controller-synthesis problem is equivalent to problems of reaching a certain set Υ and then "almost surely winning" from this set. To prove this, the "almost surely winning" property will further be reduced to finding certain set of states and actions on a product MDP (Lemma 13). In the second part of the proof, given in Section 4.2, we will show that all the above sets can be computed.

4.1 Properties of satisfying strategies

Without loss of generality suppose that the formula ψ does not contain $\mathbf{G}^1$ as the outermost operator, and that it contains n subformulas of the form $\mathbf{G}^1\varphi$. Denote these subformulas $\mathbf{G}^1\varphi_1, \ldots \mathbf{G}^1\varphi_n$. For example, $\psi = \mathsf{i} \to \big(\mathbf{G}\,(\mathsf{q} \to \mathsf{a}) \wedge \mathbf{G}^1(\mathsf{p}_1\,\mathbf{U}\,\mathsf{r} \vee \mathbf{G}^1\mathsf{a})\big)$ contains $\varphi_1 = \mathsf{p}_1\,\mathbf{U}\,\mathsf{r} \vee \mathbf{G}^1\mathsf{a}$ and $\varphi_2 = \mathsf{a}$.

The first step of our construction is to convert these formulae $\psi, \varphi_1, \ldots, \varphi_n$ to equivalent Rabin automata. However, as the formulae contain $\mathbf{G}^1$ operators, they cannot be directly expressed using Rabin automata (and as pointed out by [21], there is a fundamental obstacle preventing us from extending Rabin automata to capture $\mathbf{G}^p$).

To overcome this, we replace all occurrences of $\mathbf{G}^1\varphi_i$ in such formulae by either true or false, to capture that the frequency constraint is or is not satisfied on a run. Such a replacement can be fixed only after a point in the execution is reached where it becomes clear which frequency constraints in ψ can be satisfied. For a formula $\xi \in \{\psi, \varphi_1, \ldots \varphi_n\}$, any subset $I \subseteq \{1, \ldots, n\}$ of satisfied constraints defines a LTL formula ξ^I obtained from ξ by replacing all subformulas $\mathbf{G}^1\varphi_i$ (not contained in any other $\mathbf{G}^1$) with true if $i \in I$ and with false if $i \notin I$. The Rabin automaton for ξ^I is then denoted by $R_{\xi,I}$. For the formula ψ above, we have, e.g., $\psi^{\{1\}} = \psi^{\{1,2\}} = \mathsf{i} \to \big(\mathbf{G}\,(\mathsf{q} \to \mathsf{a}) \wedge \mathsf{true}\big)$, and $\varphi_1^{\emptyset} = \mathsf{p_1}\,\mathbf{U}\,\mathsf{r} \vee \mathsf{false}$.

We use Q for a disjoint union of the state spaces of these distinct Rabin automata, and Q_ψ for a disjoint union of the state spaces of the automata $R_{\psi,I}$, called *main* automata, for all I. Finally, for $q \in Q$ belonging to a Rabin automaton R we denote by R^q the automaton obtained from R by changing its initial state to q.

Let us fix a state s of $\mathcal{M}$ and a state q of $R_{\psi,I}$ for some $I \subseteq \{1, \ldots, n\}$. We say that a run $s_0 a_0 s_1 a_1 \ldots$ *reaches* (s, q) if for some k we have $s = s_k$ and q is the state reached by the main automaton $R_{\psi,I}$ after reading $\nu(s_0 a_0 s_1 \ldots s_{k-1})$. Once (s, q) is reached, we say that a strategy σ' is *almost-surely winning* from (s, q) if $\mathbb{P}^s_{\sigma'}$ assigns probability 1 to the set of runs ω such that $\nu(\omega)$ is accepted by $R^q_{\psi,I}$, and $\omega \models G^1\varphi_i$ whenever[1] we have $i \in I$.

▶ **Proposition 7.** *There is a strategy σ such that $\mathbb{P}_\sigma(\psi) = x$ if and only if there is a set $\Upsilon \subseteq S \times Q_\psi$ for which the following two conditions are satisfied:*

1. *There is a strategy σ' such that $\mathbb{P}_{\sigma'}(\{\omega \mid \omega \text{ reaches a pair from } \Upsilon\}) = x$.*
2. *For any $(s, q) \in \Upsilon$ there is $\sigma_{s,q}$ almost-surely winning from (s, q).*

Intuitively, the proposition relies on the fact that if $\mathbf{G}^1\varphi_i$ holds on a run, then it holds on all its suffixes, and says that any strategy σ can be altered so that almost surely there will be a prefix after which we know which of the $\mathbf{G}^1\varphi_i$ will be satisfied.

▶ **Example 8.** Let us first illustrate the set Υ on a formula $\mathbf{X}\,q \wedge \mathbf{G}\,\mathbf{F}\,\mathsf{m} \wedge \mathbf{G}^1(\mathsf{q} \to \mathbf{X}\,\mathsf{r})$ that can be satisfied on the MDP from Figure 1 with probability 0.5. Figure 2 shows Rabin automata for the formulae $\psi^{\{1\}} = \mathbf{X}\,\mathsf{q} \wedge \mathbf{G}\,\mathbf{F}\,\mathsf{m} \wedge \mathsf{true}$ (left) and $\varphi_1^{\{1\}} = \mathsf{q} \to \mathbf{X}\,\mathsf{r}$. In this simple example, the "decision" whether the formula will be satisfied (and which $\mathbf{G}^1$ subformulas will be satisfied) comes after the first step. Thus, we can set $\Upsilon = \{(s_1, q_1)\}$.

We now prove Proposition 7. The direction $\Leftarrow$ is straightforward. It suffices to define σ so that it behaves as σ' initially until it reaches some $(s, q) \in \Upsilon$ for the first time; then it behaves as $\sigma_{s,q}$.

Significantly more difficult is the direction $\Rightarrow$ of Proposition 7 that we address now. We fix a strategy σ with $\mathbb{P}_\sigma(\psi) = x$. The proof is split into three steps. We first show how to identify the set Υ, and then we show that items 1 and 2 of Proposition 7 are satisfied. The last part of the proof requires most of the effort. In the proof, we

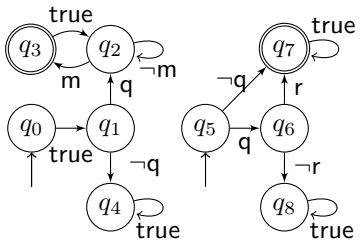


Figure 2 Example Rabin aut.

[1] Note that the product construction that we later introduce does not give us "iff" here. This is also why we require the negations to only occur in front of atomic propositions

will need to eliminate some unlikely events, and for this we will require that their probability is small to start with. For this purpose, we fix a *very small* positive number λ; to avoid cluttering of notation, we do not give a precise value of λ, but instead point out that it needs to be chosen such that any numbers that depend on it in the following text have the required properties (i.e. are sufficiently small or big; note that such choice is indeed possible). We should stress that λ is influencing *neither* the size of representation of our strategy *nor* the complexity of our algorithm.

Identifying the set Υ

In the first step, we identify an appropriate set Υ. Intuitively, we put into Υ positions of the runs satisfying ψ where *the way* ψ *is satisfied* is (nearly) decided, i.e. where it is (nearly) clear which frequency constraints will be satisfied by σ in the future. To this end, we mark every run ω satisfying ψ with a set $I_\omega \subseteq \{1, \dots, n\}$ such that $i \in I_\omega$ iff the formula $\mathbf{G}^1\varphi_i$ holds on the run. We then define a set of finite paths Γ to contain all paths h for which there is $I_h \subseteq \{1, \dots, n\}$ such that exactly the frequency constraints from I_h as well as ψ^{I_h} are satisfied on (nearly) all runs starting with h. Precisely, such that $\mathbb{P}_\sigma(\{\omega' \mid \omega' \models \psi^{I_h} \wedge I_{\omega'} = I_h\} \mid h) \geq 1 - \lambda$. Finally, for every $h \in \Gamma$ we add to Υ the pair (s, q) where $h = h's$ and q is the state in which R_{ψ,I_h} ends after reading $\nu(h')$.

Reaching Υ

It suffices to show that the strategy σ itself satisfies $\mathbb{P}_\sigma(\Gamma) = x$. We will use the following variant of Lévy's Zero-One Law, a surprisingly powerful formalization of the intuitive fact that "things need to get (nearly) decided, eventually".

▶ **Lemma 9** (Lévy's Zero-One Law [12]). *Let σ be a strategy and X a measurable set of runs. Then for* almost *every run ω we have $\lim_{n\to\infty} \mathbb{P}_\sigma(X \mid h_n) = \mathbf{1}_X(\omega)$ where each h_n denotes the prefix of ω with n states and the function $\mathbf{1}_X$ assigns 1 to $\omega \in X$ and 0 to $\omega \notin X$.*

For every $I \subseteq \{1, \dots, n\}$ we define $X_I = \{\omega' \mid \omega' \models \psi^I \wedge I_{\omega'} = I\}$ to be the set of runs that are marked by I and satisfy the formula ψ^I. Then by Lemma 9, for almost every run ω that satisfies ψ and has $I_\omega = I$, there must be a prefix h of the run for which $\mathbb{P}_\sigma(X_I \mid h) \geq 1 - \lambda$ because $\omega \in X_I$. Any such prefix was added to Γ, with $I_h = I$.

Almost-surely winning from Υ

For the third step of the proof of direction $\Rightarrow$ of Proposition 7 we fix $(s^\star, q^\star) \in \Upsilon$ and we construct a strategy $\sigma_{s^\star,q^\star}$ that is almost-surely winning from $(s^\star, q^\star)$. Furthermore, let $I^\star \subseteq \{1, \dots, n\}$ denote the set such that $q^\star$ is a state from $R_{\psi,I^\star}$. As we have shown in Theorem 5, strategies might require infinite memory, and this needs to be taken into consideration when constructing $\sigma_{s^\star,q^\star}$. The strategy cycles through two "phases", called *accumulating* and *reaching* that we illustrate on our example.

▶ **Example 10.** Returning to Example 8, we fix $(s^\star, q^\star) = (s_1, q_1)$ and $I^\star = \{1\}$, with the corresponding history from Γ being $s_0 w s_1$. The strategy σ_{s_1,q_1} we would like to obtain

- first "accumulates" arbitrarily many steps from which all $\varphi_1^{\{1\}}$ can be almost surely satisfied. I.e., it accumulates arbitrarily many newly started instances of the Rabin automaton $R_{\varphi_1,\{1\}}$ (all being in state q_5) by repeating action w in s_0.
- Then it "reaches" with all the Rabin automata $R_{\psi,\{1\}}$ and $R_{\varphi_1,\{1\}}$ accumulated in the previous phase their accepting states q_3 and q_7 respectively. For $R_{\varphi_1,\{1\}}$ this happens

without any intervention of the strategy, but for $R_{\psi,\{1\}}$ the strategy needs to take the action m. Then after returning to s_0 it comes back to a state where the next accumulating phase starts. Thus, we need to make sure we make the accumulating phases progressively longer so that in the long run they take place with frequency 1.

The proof that such a simple behaviour suffices is highly non-trivial. To illustrate this, let us extend the MDP from Figure 1 with an action $\mathsf{decline}$ with $\Delta(s_1, \mathsf{decline}) = s_0$. The strategy σ from the proof of Theorem 5 satisfies $\mathbb{P}_\sigma(\psi) = 1$ for $\psi = \mathbf{G}\,\mathbf{F}\,\mathsf{m} \wedge \mathbf{G}^1(\mathsf{q} \to \mathbf{X}\,\mathsf{r})$. However, we can modify σ and obtain a "weird" strategy σ' that takes the action $\mathsf{decline}$ in the i-th visit to s_1 with probability $1/2^i$. Such a strategy (a) still satisfies $\mathbb{P}_{\sigma'}(\psi) = 1/2$ but (b) it does not guarantee almost sure satisfaction of $\varphi_1^{\{1\}}$ in s_1. Thus, it does not accumulate in the sense explained above. We will show that any such weird strategy can be slightly altered to fit into our scheme. ◀

To show that alternation between such accumulating and reaching suffices (and to make a step towards the algorithm to construct such $\sigma_{s^\star,q^\star}$), we introduce a tailor-made product construction $\mathcal{M}_\otimes$. The product keeps track of a *collection* of arbitrarily many Rabin automata accumulated up to now. We need to make sure that almost all runs of all automata in the collection are accepting, and we will do this by ensuring that: (i) almost every computation of all Rabin automata eventually commits to an accepting condition (E, F), and (ii) from the point the automaton "commits" to the accepting condition, no more elements of E are visited and (iii) some element of F is visited infinitely often. To ensure this, we store additional information along any state $q \in Q$ of each automaton:

- $(q, \star)$ is a new instance that has to commit to an accepting condition soon;
- $(q, (E, F)_\circ)$ is an instance that has to visit a state of F soon;
- $(q, (E, F)_\bullet)$ is an instance that recently fulfilled the accepting condition by visiting F;
- $(q, \bot)$ is an instance that violated the accepting condition it had committed to.

Let $\mathcal{C}$ denote the set of these pairs for all $q \in Q$ and all accepting conditions (E, F) of the Rabin automaton where the state q belongs. Note that $\mathcal{C}$ is finite; because we need to encode unbounded number of instances of Rabin automata along the run, each element of a collection $C \subseteq \mathcal{C}$ might stand for multiple instances that are in exactly the same configuration. We say that $C \subseteq \mathcal{C}$ is *fulfilled* if it contains only elements of the form $(q, (E, F)_\bullet)$. The aim is to fulfil the collection infinitely often, the precise meaning of "recently" and "soon" above is then "since the last fulfilled state" and "before the next fulfilled state".

Using the product $\mathcal{M}_\otimes$, we show that if there is a satisfying strategy in $\mathcal{M}$, there is a strategy in $\mathcal{M}_\otimes$ with a simple structure that visits a fulfilled state infinitely often (in Lemma 13). Due to the simple structure, such a strategy can be found algorithmically. Finally, we show that such a strategy in the product induces a satisfying strategy in $\mathcal{M}$ (in Lemma 12) yielding correctness of the algorithm.

The product. Let $\mathcal{M}_\otimes$ be an MDP with states $S_\otimes = S \times 2^{\mathcal{C}}$, actions $A_\otimes = A \times 2^{\mathcal{C}}$, and transition function $\Delta_\otimes$ defined as follows. We first define possible choices of a strategy in $\mathcal{M}_\otimes$. Given a state (s, C_s), we say that an action (a, C_a) is *legal* in (s, C_s) if a is a valid choice in s in the original MDP, i.e. $\Delta(s, a)$ is defined; and C_a satisfies the following:

- for all tuples $(q, \star) \in C_s$ we have $(q, \star) \in C_a$ or $(q, (E, F)_\circ) \in C_a$ for some accepting condition (E, F), (q can "commit" to (E, F), or keep waiting)
- for all $(q, x) \in C_s$ with $x \neq \star$ we have $(q, x) \in C_a$, (all q are kept along with the commitments)
- all $(q, x) \in C_a$, not added by one of the two above items, are of the form $(q_{in}, \star)$ where q_{in} is the initial state of a Rabin automaton $R_{\varphi_i, I^\star}$ for $i \in I^\star$, (initial states can be added)

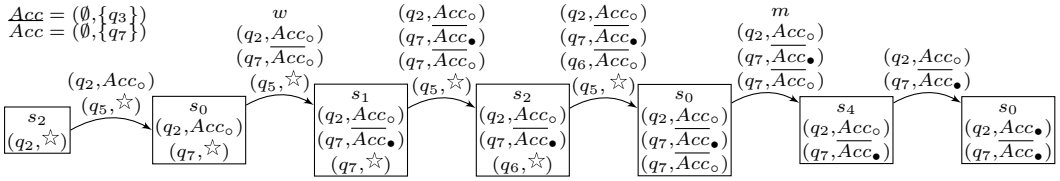


Figure 3 Illustration for Example 11. The names of actions from $\mathcal{M}$ are omitted when only a single action is available.

The randomness in $\mathcal{M}_\otimes$ comes only from $\mathcal{M}$. We set $\Delta_\otimes((s, C_s), (a, C_a))(t, C_t) = \Delta(s, a)(t)$ for any state (s, C_s), any action (a, C_a) legal in (s, C_s), and any state (t, C_t) such that C_a "deterministically evolves" by reading s into C_t. Precisely, we require that C_t is the minimal set such that for any $(q, x) \in C_a$ there is $(q', x') \in C_t$ with $q \overset{\nu(s)}{\to} q'$ and $x \overset{q'}{\rightsquigarrow} x'$ where the latter relation is defined by $\star \overset{q'}{\rightsquigarrow} \star$ and $\bot \overset{q'}{\rightsquigarrow} \bot$ and for any $\cdot \in \{\bullet, \circ\}$ by

- $(E, F). \overset{q'}{\rightsquigarrow} (E, F).$ if $q' \notin E \cup F$ and C is not fulfilled, (no special state visited)
- $(E, F). \overset{q'}{\rightsquigarrow} (E, F)_\circ$ if $q' \notin E \cup F$ and C is fulfilled, (resetting back to $\circ$)
- $(E, F). \overset{q'}{\rightsquigarrow} (E, F)_\bullet$ if $q' \in F$, (the accepting condition becomes fulfilled)
- $(E, F). \overset{q'}{\rightsquigarrow} \bot$ if $q' \in E$; (the accepting condition is violated)

Finally, a state is called *fulfilled* if its second component is fulfilled.

▶ **Example 11.** Figure 3 shows one path in the product $\mathcal{M}_\otimes$ for the MDP and the Rabin automata from Example 8. The path shown illustrates how the initial states can be added non-deterministically (in the first three steps), and then reaches a fulfilled state.

A very useful property of the product is that any strategy that ensures visiting fulfilled states infinitely often yields a strategy in the original MDP such that the automata the strategy added almost surely accept. This is formalised in the following lemma.

▶ **Lemma 12.** *For a deterministic strategy π in $\mathcal{M}_\otimes$ there is a strategy π' in $\mathcal{M}$ that for any $h = (s_0, C_0) \cdots (a_n, D_n)(s_{n+1}, C_{n+1})$ with $\mathbb{P}^h_\pi(\{\text{fulfilled state visited infinitely often}\}) = 1$:*

- $\mathbb{P}^{s_0}_{\pi'}(s_0 \dots a_n s_{n+1}) = \mathbb{P}^{(s_0, C_0)}_\pi(h)$, *and*
- *for any $(q, \star) \in D_n$ where R is the automaton of q, $\mathbb{P}^{s_0 a_0 \cdots s_n}_{\pi'}(\{\omega \mid R^q \text{ accepts } \omega\}) = 1$.*

To be able to use above lemma, we need to establish that it is *sufficient* to look for a strategy that visits fulfilled states infinitely often. In other words that existence of the satisfying strategy σ implies existence of a strategy that visits fulfilled states infinitely often. Here we use the following lemma saying that σ and $(s^\star, q^\star)$ give rise to two strategies in the product $\mathcal{M}_\otimes$ that can be used to add initial states into the collections, and to reach fulfilled states. We will show below how these strategies can be used to finish the proof of Proposition 7.

▶ **Lemma 13.** *Assume $s^\star, q^\star, I^\star$ are chosen as described on page 192. Then there are sets $M \subseteq S_\otimes$, $N \subseteq A_\otimes$ where N contains only "accumulating" actions, i.e. actions (a, C) with $\{(q_{in}, \star) \mid q_{in} \text{ is the initial state of } R_{\varphi_i, I^\star} \text{ for } i \in I^\star\} \subseteq C$; and there are finite-memory deterministic strategies π and ζ such that:*

1. *When starting in $(s, C) \in M$, π only uses actions from N and never leaves M*
2. *When starting in $(s, C) \in M \cup \{(s^\star, \{(q^\star, \star)\})\}$, ζ almost surely reaches a fulfilled state (possibly leaving M) and then reaches M.*

Proof idea. The proof is involved and gives a crucial insight into the main obstacles of the proof of Theorem 6. Due to the space constraints we only sketch it here.

We first prove that for any fixed ℓ, almost every ω that satisfies all $\mathbf{G}^1\varphi_i^{I^\star}$ has infinitely many *good* prefixes. Intuitively, a finite path h is *good* if, when starting from h, all the automata $R_{\varphi_i,I^\star}$ for $i \in I^\star$ started within ℓ first steps accept with probability at least $1-\lambda$.

In the second step, we show how to avoid actions that cause that any $R_{\varphi_i,I^\star}$ does not accept. To do so, we inductively start labelling the prefixes of runs of the MDP with elements of $\mathcal{C}$. Having fixed a label for a prefix, the label for its extension is obtained by "deterministic evolving" as in the definition of the product MDP, and by (non-deterministically) adding $(q_{in}, \star)$. The latter part is performed by switching between a "pseudo-accumulating" and "pseudo-reaching" phase. Initially, we start in a pseudo-reaching phase, only with singletons corresponding to the current state of $R_{\psi,I^\star}$, and do not add any $(q_{in}, \star)$. When a good prefix is reached (which happens almost surely), we switch to a pseudo-accumulating phase for the next ℓ steps and we keep adding "initial states" $(q_{in}, \star)$ of $R_{\varphi_i,I^\star}$ for each $i \in I^\star$. After ℓ steps, we switch back to a pseudo-reaching phase and do not add any new elements to the label until we pass through a state whose label is fulfilled and get to a good prefix again, in which point another pseudo-accumulating phase starts.

Along the way, we might obtain tuples of the form $(q, \bot)$ in the label, or we might not ever visit a fulfilled state. Indeed, if we repeated our steps to infinity, such an "error" might take place almost surely. However, before an error happens with too high probability, the labels start repeating because $\mathcal{C}$ is finite. We show that supposing ℓ was large enough and our tolerance λ was small enough, there must be a strategy that *almost-surely* traverses such a cycle without any error. We can extract from the pseudo-accumulating and pseudo-reaching phases of such a strategy the sets M (and N), given by the tuples of the MDP states (actions) and their labels. ◀

We are now ready to finish the proof of Proposition 7. We show that Lemma 13 allows us to construct a strategy $\sigma_\otimes$ for $\mathcal{M}_\otimes$ that almost surely (i) visits fulfilled states, and (ii) with frequency 1 it takes actions from N. By Lemma 12 this strategy yields an almost-surely winning strategy $\sigma_{s^\star,q^\star}$ in $\mathcal{M}$.

The strategy $\sigma_\otimes$ is constructed as follows. Inductively, for path h in $\mathcal{M}_\otimes$, we say that its first accumulating phase starts in the first step, ith accumulating phase takes i steps, and the $(i+1)$th accumulating phase starts when the set M is reached through a fulfilled state after the ith accumulating phase ended. Within every accumulating phase started in a history h, $\sigma_\otimes$ is defined to play as π initiated after h. Similarly, outside every accumulating phase ended in a history h, $\sigma_\otimes$ is defined to play as ζ.

4.2 The algorithm

To conclude the proof of Theorem 6, we need to give a procedure for computing the optimal probability of satisfying ψ. It works in the following steps (for details, see [14]):

1. Initialize $\Upsilon := \emptyset$, and construct $R_{\xi,I}$ for all $\xi \in \{\psi, \varphi_1, \ldots, \varphi_n\}$ and $I \subseteq \{1, \ldots, n\}$.
2. For every I find the largest sets (M_I, N_I) satisfying the conditions 1–2 of Lemma 13, and add to Υ all pairs (s, q) such that M_I can be almost-surely reached from $(s, \{(q, \star)\})$.
3. Compute an optimal strategy σ' for "reaching" Υ and return the probability. Intuitively,
   - we build the "naive" product of $\mathcal{M}$ with all the main automata $R_{\psi,I}$ for $I \subseteq \{1, \ldots, n\}$;
   - reaching Υ is reduced to ordinary reachability of all states of the form $(s, q_1, \ldots, q_m)$ such that $(s, q_i) \in \Upsilon$ for some i.
   - By standard algorithms for reachability in MDP, we find an optimal strategy σ'' in the naive product that easily induces the strategy σ' in $\mathcal{M}$.

By connecting Proposition 7, Lemmas 12 and 13, and the construction of $\sigma_{\otimes}$ above, there is a strategy σ in $\mathcal{M}$ yielding probability $\geq p$ iff the set Υ computed by the algorithm can also be reached with probability $\geq p$.

We briefly discuss the complexity of the algorithm. Each of the Rabin automata in step 1 above can be computed in time $2^{2^{\mathsf{poly}(|\varphi|)}}$, and since there is exponentially many such automata (in $|\varphi|$), step 1. takes time $2^{2^{poly(|\varphi|)}}$. Step 2 can be performed in time $\mathsf{poly}(S) \cdot 2^{2^{\mathsf{poly}(|\varphi|)}}$. In step 3 we are computing reachability probability in the naive product MDP which is of size $\mathsf{poly}(S) \cdot 2^{2^{\mathsf{poly}(|\varphi|)}}$, and so also this step can be done in time $\mathsf{poly}(S) \cdot 2^{2^{\mathsf{poly}(|\varphi|)}}$.

5 Conclusions

We have given algorithms for controller synthesis of the logic LTL extended with an operator expressing that frequencies of some events exceed a given bound. For Markov chains we gave an algorithm working with the complete logic, and for MDPs we require the formula to be from a certain fragment. The obvious next step is extending the MDP results to the whole fLTL. This will require new insights. Our product construction relies on the (non-trivial) observation that given $\mathbf{G}^1\varphi$, the formula φ is almost surely satisfied from any history of an accumulating phase. This is no longer true when the frequency bound is lower than 1. In such cases different histories may require different probability of satisfying φ. However, both authors strongly believe that even for these cases the problem is decidable. Another promising direction for future work is implementing the algorithms into a probabilistic model checker and evaluating their time requirements experimentally.

Acknowledgements. The authors would like to thank anonymous reviewers for their insightful comments on an earlier version of this paper. This work is partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center AVACS (SFB/TR 14), by the Czech Science Foundation under grant agreement P202/12/G061, by the EU 7th Framework Programme under grant agreement no. 295261 (MEALS) and 318490 (SENSATION), by the CDZ project 1023 (CAP), by the CAS/SAFEA International Partnership Program for Creative Research Teams, and EPSRC grant EP/M023656/1. Vojtěch Forejt is also affiliated with Faculty of Informatics, Masaryk University, Brno, Czech Republic.

References

1 Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model checking continuous-time Markov chains by transient analysis. In *CAV*, volume 1855 of *LNCS*. Springer, 2000.

2 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

3 Christel Baier, Joachim Klein, Sascha Klüppelholz, and Sascha Wunderlich. Weight monitoring with linear temporal logic: Complexity and decidability. In *CSL-LICS*, page 11. ACM, 2014.

4 Roderick Bloem, Krishnendu Chatterjee, Thomas A Henzinger, and Barbara Jobstmann. Better quality in synthesis through quantitative objectives. In *CAV*, pages 140–156. Springer, 2009.

5 Udi Boker, Krishnendu Chatterjee, Thomas A Henzinger, and Orna Kupferman. Temporal specifications with accumulative values. In *LICS 2011*, pages 43–52. IEEE, 2011.

6 Benedikt Bollig, Normann Decker, and Martin Leucker. Frequency linear-time temporal logic. In *TASE'12*, pages 85–92, Beijing, China, July 2012. IEEE Computer Society Press.

7 Patricia Bouyer, Nicolas Markey, and Raj Mohan Matteplackel. Averaging in LTL. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014*, volume 8704 of *LNCS*, pages 266–280. Springer, 2014.

8 Tomáš Brázdil, Vojtěch Forejt, and Antonín Kučera. Controller synthesis and verification for markov decision processes with qualitative branching time objectives. In *ICALP 2008*, volume 5126 of *LNCS*, pages 148–159. Springer, 2008.

9 Krishnendu Chatterjee and Laurent Doyen. Energy and mean-payoff parity Markov decision processes. In *MFCS 2011*, pages 206–218. Springer, 2011.

10 Krishnendu Chatterjee and Laurent Doyen. Games and markov decision processes with mean-payoff parity and energy parity objectives. In *MEMICS*, pages 37–46. Springer, 2012.

11 Krishnendu Chatterjee, Thomas A Henzinger, and Marcin Jurdzinski. Mean-payoff parity games. In *LICS 2005*, pages 178–187. IEEE, 2005.

12 Kai-lai Chung. *A Course in Probability Theory*. Academic Press, 3 edition, 2001.

13 Luca De Alfaro. How to specify and verify the long-run average behaviour of probabilistic systems. In *LICS 1998*, pages 454–465. IEEE, 1998.

14 Vojtěch Forejt and Jan Krčál. On frequency LTL in probabilistic systems. *CoRR*, abs/1501.05561, 2015.

15 J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. Springer, 2nd edition, 1976.

16 Antonín Kučera and Oldřich Stražovský. On the controller synthesis for finite-state Markov decision processes. In *FSTTCS 2005*, pages 541–552. Springer, 2005.

17 M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic verification of herman's self-stabilisation algorithm. *Formal Aspects of Computing*, 24(4):661–670, 2012.

18 J. R. Norris. *Markov chains*. Cambridge University Press, 1998.

19 V. Shmatikov. Probabilistic model checking of an anonymity system. *Journal of Computer Security*, 12(3/4):355–377, 2004.

20 Takashi Tomita, Shigeki Hagihara, and Naoki Yonezaki. A probabilistic temporal logic with frequency operators and its model checking. In *INFINITY*, volume 73 of *EPTCS*, pages 79–93, 2011.

21 Takashi Tomita, Shin Hiura, Shigeki Hagihara, and Naoki Yonezaki. A temporal logic with mean-payoff constraints. In *Formal Methods and Soft. Eng.*, volume 7635 of *LNCS*. Springer, 2012.

Modal Logics for Nominal Transition Systems

Joachim Parrow, Johannes Borgström, Lars-Henrik Eriksson, Ramūnas Gutkovas, and Tjark Weber

Uppsala University
Sweden

Abstract

We define a uniform semantic substrate for a wide variety of process calculi where states and action labels can be from arbitrary nominal sets. A Hennessy-Milner logic for these systems is introduced, and proved adequate for bisimulation equivalence. A main novelty is the use of finitely supported infinite conjunctions. We show how to treat different bisimulation variants such as early, late and open in a systematic way, and make substantial comparisons with related work. The main definitions and theorems have been formalized in Nominal Isabelle.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.2 Modes of Computation, F.3.1 Specifying and Verifying and Reasoning about Programs, F.3.2 Semantics of Programming Languages, F.4.1 Mathematical Logic

Keywords and phrases Process algebra, nominal sets, bisimulation, modal logic

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.198

1 Introduction

Transition systems. Transition systems are ubiquitous in models of computing, and specifications to say what may and must happen during executions are often formulated in a modal logic. There is a plethora of different versions of both transition systems and logics, including a variety of higher-level constructs such as updatable data structures, new name generation, alias generation, dynamic topologies for parallel components etc. In this paper we formulate a general framework where such aspects can be treated uniformly, and define accompanying modal logics which are adequate for bisimulation. This is related to, but independent of, our earlier work on psi-calculi [4], which proposes a particular syntax for defining behaviours. The present paper does not depend on any such language, and provides general results for a large class of transition systems.

In any transition system there is a set of *states* $P, Q, \ldots$ representing the configurations a system can reach, and a relation telling how a computation can move between them. Many formalisms, for example all process algebras, define languages for expressing states, but in the present paper we shall make no assumptions about any such syntax.

In systems describing communicating parallel processes the transitions are labelled with *actions* α, β, representing the externally observable effect of the transition. A transition $P \xrightarrow{\alpha} P'$ thus says that in state P the execution can progress to P' while conducting the action α, which is visible to the rest of the world. For example, in CCS these actions are atomic and partitioned into output and input communications. In value-passing calculi the actions can be more complicated, consisting of a channel designation and a value from some data structure to be sent along that channel.

Scope openings. With the advent of the pi-calculus [19] an important aspect of transitions was introduced: that of name generation and scope opening. The main idea is that names (i.e.,

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 198–211

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

atomic identifiers) can be scoped to represent local resources. They can also be transmitted in actions, to give a parallel entity access to this resource. In the monadic pi-calculus such an action is written $\overline{a}(\nu b)$, to mean that the local name b is exported along the channel a. These names can be subjected to alpha-conversion: if $P \xrightarrow{\overline{a}(\nu b)} P'$ and c is a fresh name then also $P \xrightarrow{\overline{a}(\nu c)} P'\{c/b\}$, where $P'\{c/b\}$ is P' with all bs replaced by cs. Making this idea fully formal is not entirely trivial and many papers gloss over it. In the polyadic pi-calculus several names can be exported in one action, and in psi-calculi arbitrary data structures may contain local names. In this paper we make no assumptions about how actions are expressed, and just assume that for any action α there is a finite set of names $\mathrm{bn}(\alpha)$, the *binding names*, representing exported names. In our formalization we use nominal sets, an attractive theory to reason about objects depending on names on a high level and in a fully rigorous way.

State predicates. The final general components of our transition systems are the *state predicates* ranged over by φ, representing what can be concluded in a given state. For example state predicates can be equality tests of expressions, or connectivity between communication channels. We write $P \vdash \varphi$ to mean that in state P the state predicate φ holds.

A structure with states, transitions, and state predicates as discussed above we call a *nominal transition system*.

Hennessy-Milner Logic. Modal logic has been used since the 1970s to describe how facts evolve through computation. We use the popular and general branching time logic known as Hennessy-Milner Logic [15] (HML). Here the idea is that an action modality $\langle\alpha\rangle$ expresses a possibility to perform an action α. If A is a formula then $\langle\alpha\rangle A$ says that it is possible to perform α and reach a state where A holds. With conjunction and negation this gives a powerful logic shown to be *adequate* for bisimulation equivalence: two processes satisfy the same formulas exactly if they are bisimilar. In the general case, conjunction must take an infinite number of operands when the transition systems have states with an infinite number of outgoing transitions. The fully formal treatment of this requires care in ensuring that such infinite conjunctions do not exhaust all names, leaving none available for alpha-conversion. All previous works that have considered this issue properly have used uniformly bounded conjunction, i.e., the set of all names in all conjuncts is finite.

Contributions. Our definition of nominal transition systems is very general since we leave open what the states, transitions and predicates are. The only requirement is that transitions satisfy alpha-conversion. A technically important point is that we do not assume the usual *name preservation principle*, that if $P \xrightarrow{\alpha} P'$ then the names occurring in P' must be a subset of those occurring in P and α. This means that the results are applicable to a wide range of calculi. For example, the pi-calculus represents a trivial instance where there are no state predicates. CCS represent an even more trivial instance where bn always returns the empty set. In the fusion calculus and the applied pi-calculus the state contains an environmental part which tells what expressions are equal to what. In the general framework of psi-calculi the states are processes with assertions describing their environments.

We define a modal logic with the $\langle\alpha\rangle$ operator that binds the names in $\mathrm{bn}(\alpha)$, and contains operators for state predicates. In this way we get a logic for an arbitrary nominal transition system such that logical equivalence coincides with bisimilarity. We also show how variants of the logic correspond to late, open and hyperbisimilarity in a uniform way. The main technical difficulty is to ensure that formulas and their alpha-equivalence classes throughout are finitely supported, i.e., only depend on a finite set of names, even in the

presence of infinite conjunction. Instead of uniformly bounded conjunction we use the notion of finite support from nominal sets. This results in greater generality and expressiveness. For example, we can now define quantifiers and the next step modalities as derived operators.

Formalization. Our main definitions and theorems have been formalized in Nominal Isabelle [27]. This has required significant new ideas to represent data types with infinitary constructors like infinite conjunction and their alpha-equivalence classes. As a result we corrected several details in our formulations and proofs, and now have very high confidence in their correctness. The formalization effort has been substantial, but certainly less than half of the total effort, and we consider it a very worthwhile investment.

Exposition. In the following section we provide the necessary background on nominal sets. In Section 3 we present our main definitions and results on nominal transition systems and modal logics. In Section 4 we derive useful operators such as quantifiers and fixpoints, and indicate some practical uses. Section 5 shows how to treat variants of bisimilarity such as late and open in a uniform way, and in Section 6 we compare with related work and demonstrate how our framework can be applied to recover earlier results uniformly. Finally Section 7 concludes with some remarks on the formalization in Nominal Isabelle. All full proofs are contained in the appendix of the technical report [22].

2 Background on nominal sets

Nominal sets [25] is a general theory of objects which depend on names, and in particular formulates the notion of alpha-equivalence when names can be bound. The reader need not know nominal set theory to follow this paper, but some key definitions will make it easier to appreciate our work and we recapitulate them here.

We assume an infinitely countable multi-sorted set of atomic identifiers or *names* $\mathcal{N}$ ranged over by $a, b, \ldots$. A *permutation* is a bijection on names that leaves all but finitely many names invariant. The singleton permutation which swaps names a and b and has no other effect is written $(a\,b)$, and the identity permutation that swaps nothing is written id. Permutations are ranged over by π, π'. The effect of applying a permutation π to an object X is written $\pi \cdot X$. Formally, the permutation action $\cdot$ can be any operation that satisfies $\mathrm{id} \cdot X = X$ and $\pi \cdot (\pi' \cdot X) = (\pi \circ \pi') \cdot X$, but a reader may comfortably think of $\pi \cdot X$ as the object obtained by permuting all names in X according to π.

A set of names N *supports* an object X if for all π that leave all members of N invariant it holds $\pi \cdot X = X$. In other words, if N supports X then names outside N do not matter to X. If a finite set supports X then there is a unique minimal set supporting X, called the *support* of X, written $\mathrm{supp}(X)$, intuitively consisting of exactly the names that matter to X. As an example the set of names textually occurring in a datatype element is the support of that element, and the set of free names is the support of the alpha-equivalence class of the element. Note that in general, the support of a set is not the same as the union of the support of its members. An example is the set of all names; each element has itself as support, but the whole set has empty support since $\pi \cdot \mathcal{N} = \mathcal{N}$ for any π.

We write $a\#X$, pronounced "a is fresh for X", for $a \notin \mathrm{supp}(X)$. The intuition is that if $a\#X$ then X does not depend on a in the sense that a can be replaced with any fresh name without affecting X. If A is a set of names we write $A\#X$ for $\forall a \in A\,.\, a\#X$.

A *nominal set* S is a set with a permutation action such that $X \in S \Rightarrow \pi \cdot X \in S$, and where each member $X \in S$ has finite support. A main point is that then each member

has infinitely many fresh names available for alpha-conversion. Similarly, a set of names N supports a function f on a nominal set if for all π that leave N invariant it holds $\pi \cdot f(X) = f(\pi \cdot X)$, and similarly for relations and functions of higher arity. Thus we extend the notion of support to finitely supported functions and relations as the minimal finite support, and can derive general theorems such as $\mathrm{supp}(f(X)) \subseteq \mathrm{supp}(f) \cup \mathrm{supp}(X)$.

An object that has empty support we call *equivariant*. For example, a unary function f is equivariant if $\pi \cdot f(X) = f(\pi \cdot X)$ for all π, X. The intuition is that an equivariant object does not treat any name special.

3 Nominal transition systems and Hennessy-Milner logic

▶ **Definition 1.** A *nominal transition system* is characterized by the following

- STATES: A nominal set of *states* ranged over by P, Q.
- PRED: A nominal set of *state predicates* ranged over by φ.
- An equivariant binary relation $\vdash$ on STATES and PRED. We write $P \vdash \varphi$ to mean that in state P the state predicate φ holds.
- ACT: A nominal set of *actions* ranged over by α.
- An equivariant function bn from ACT to finite sets of names, which for each α returns a subset of $\mathrm{supp}(\alpha)$, called the *binding names*.
- An equivariant transition relation $\rightarrow$ on states and residuals. A residual is a pair of action and state. For $\rightarrow (P, (\alpha, P'))$ we write $P \xrightarrow{\alpha} P'$. The transition relation must satisfy alpha-conversion of residuals: If $a \in \mathrm{bn}(\alpha)$, $b \# \alpha, P'$ and $P \xrightarrow{\alpha} P'$ then also $P \xrightarrow{(a\,b)\cdot\alpha} (a\,b) \cdot P'$.

▶ **Definition 2.** A *bisimulation* R is a symmetric binary relation on states in a nominal transition system satisfying the following two criteria: $R(P, Q)$ implies

1. *Static implication*: $P \vdash \varphi$ implies $Q \vdash \varphi$.
2. *Simulation*: For all α, P' such that $\mathrm{bn}(\alpha) \# Q$ there exist Q' such that if $P \xrightarrow{\alpha} P'$ then $Q \xrightarrow{\alpha} Q'$ and $R(P', Q')$

We write $P \dot{\sim} Q$ to mean that there exists a bisimulation R such that $R(P, Q)$.

Static implication means that bisimilar states must satisfy the same state predicates; this is reasonable since these can be tested by an observer. The simulation requirement is familiar from the pi-calculus.

▶ **Proposition 1.** *$\dot{\sim}$ is an equivariant equivalence relation.*

The minimal HML for nominal transition systems is the following.

▶ **Definition 3.** The nominal set of formulas $\mathcal{A}$ ranged over by A is defined by induction as follows:

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

Support and name permutation are defined as usual (permutation distributes over all formula constructors). In $\bigwedge_{i \in I} A_i$ it is assumed that the indexing set I has bounded cardinality, by which we mean that $|I| \leq \kappa$ for some fixed infinite cardinal κ at least as large as the cardinality of STATES, ACT and PRED. It is also required that the set of conjuncts $\{A_i \mid i \in I\}$ has finite support; this is then the support of the conjunction. Alpha-equivalent formulas are identified; the only binding construct is in $\langle \alpha \rangle A$ where $\mathrm{bn}(\alpha)$ binds into A.

Compared to previous work there are two main novelties in Definition 3. The first is that we use conjunction of a possibly infinite and finitely supported set of conjuncts. In comparison, the earliest HML for CCS, Hennessy and Milner (1985) [15], uses finite conjunction, meaning that the logic is adequate only for finite branching transition systems. In his subsequent book (1989) [18] Milner admits arbitrary infinite conjunction, disregarding the danger of running into paradoxes. Abramsky (1991) [3] employs a kind of uniformly bounded conjunction, with a finite set of names that supports all conjuncts, an idea that is also used in the first HML for the pi-calculus (1993) [20]. All subsequent developments follow one of these three approaches. Our main point is that both finite and uniformly bounded conjunction are expressively weak, in that the logic is not adequate for the full range of nominal transition systems, and in that quantifiers over infinite structures are not definable. In contrast, our use of finitely supported sets of conjuncts is adequate for all nominal transition systems (cf. Theorems 6 and 9 below) and admits quantifiers as derived operators (cf. Section 4 below). As a simple example, universal quantification over names $\forall x \in \mathcal{N}.A(x)$ is usually defined to mean that $A(n)$ must hold for all $n \in \mathcal{N}$. We can define this as the (infinite) conjunction of all these $A(n)$. This set of conjuncts is not uniformly bounded if $n \in \mathrm{supp}(A(n))$. But it is supported by $\mathrm{supp}(A)$ since, for any permutation π not affecting $\mathrm{supp}(A)$ we have $\pi \cdot A(n) = A(\pi(n))$ which is also a conjunct; thus the set of conjuncts is unaffected by π.

The second novelty is the use of a nominal set of actions α with binders, and the formal definition of alpha-equivalence. We define it by structural recursion over formulas. Two conjunctions $\bigwedge_{i\in I} A_i$ and $\bigwedge_{i\in I} B_i$ are alpha-equivalent if for every conjunct A_i there is an alpha-equivalent conjunct B_j, and vice versa. The other cases are standard; two formulas $\langle\alpha\rangle A$ and $\langle\beta\rangle B$ are alpha-equivalent if there exists a permutation π, renaming the binding names of α to those of β, such that $\pi \cdot A$ and B are alpha-equivalent, and $\pi \cdot \alpha = \beta$. Moreover, π must leave names that are free in A invariant. The free names in a formula are also defined by structural recursion. Most cases are standard again; a name is free in $\langle\alpha\rangle A$ if it is in $\mathrm{supp}(\alpha)$ or free in A, and not contained in $\mathrm{bn}(\alpha)$. However, the free names in a conjunction are given by the support of its alpha-equivalence class (rather than by the union of free names in all conjuncts). This is analogous to the situation for nominal sets in general, whose support is not necessarily the same as the union of the support of its members. Fortunately, our formalization proves that we need not keep the details of this construction in mind, but can simply identify alpha-equivalent formulas. The notions of free names and support then coincide.

The validity of a formula A for a state P is written $P \models A$ and is defined by recursion over A as follows.

▶ **Definition 4.**

$P \models \bigwedge_{i\in I} A_i$	if for all $i \in I$ it holds that $P \models A_i$
$P \models \neg A$	if not $P \models A$
$P \models \varphi$	if $P \vdash \varphi$
$P \models \langle\alpha\rangle A$	if there exists P' such that $P \xrightarrow{\alpha} P'$ and $P' \models A$

In the last clause we assume that $\langle\alpha\rangle A$ is a representative of its alpha-equivalence class such that $\mathrm{bn}(\alpha)\#P$. It is easy to show that $\models$ is an equivariant relation.

▶ **Definition 5.** Two states P and Q are *logically equivalent*, written $P \doteq Q$, if for all A it holds that $P \models A$ iff $Q \models A$

▶ **Theorem 6.** $P \dot{\sim} Q \implies P \doteq Q$

The proof is by induction over formulas. The converse result uses the idea of *distinguishing formulas*.

▶ **Definition 7.** A *distinguishing formula* for P and Q is a formula A such that $P \models A$ and not $Q \models A$.

The following lemma says that we can find such a formula where, a bit surprisingly, the support does not depend on Q.

▶ **Lemma 8.** *If $P \not\doteq Q$ then there exists a distinguishing formula B for P and Q such that* $\mathrm{supp}(B) \subseteq \mathrm{supp}(P)$.

The proof is by direct construction. If $P \not\doteq Q$ then there exists a distinguishing formula A for P and Q. Let Π_P be the group of finite permutations that leave names in $\mathrm{supp}(P)$ invariant, i.e., $\Pi_P = \{\pi \mid \forall n \in \mathrm{supp}(P).\ \pi(n) = n\}$. Then $\{\pi \cdot A \mid \pi \in \Pi_P\}$ is supported by $\mathrm{supp}(P)$. Since $\models$ is equivariant we have that for all $\pi \in \Pi_P$ it holds $P = \pi \cdot P \models \pi \cdot A$. Let $B = \bigwedge_{\pi \in \Pi_P} \pi \cdot A$, thus $P \models B$ but $Q \not\models B$ since the identity is in Π_P and $Q \not\models A$. Note that B here uses a conjunction which is not uniformly bounded.

▶ **Theorem 9.** $P \doteq Q \implies P \dot{\sim} Q$

The main idea of the proof is to establish that $\doteq$ is a bisimulation. The simulation requirement is by contradiction: Assume that $\doteq$ does not satisfy the simulation requirement. Then there exist P, Q, P', α such that $P \doteq Q$ and $P \xrightarrow{\alpha} P'$ and, letting $\mathcal{Q} = \{Q' \mid Q \xrightarrow{\alpha} Q'\}$, for all $Q' \in \mathcal{Q}$ it holds $P' \not\doteq Q'$. By Lemma 8 we can find a distinguishing formula $B_{Q'}$ for P' and Q' with $\mathrm{supp}(B_{Q'}) \subseteq \mathrm{supp}(P')$. Therefore the formula $B = \bigwedge_{Q' \in \mathcal{Q}} B_{Q'}$ is well-formed with support included in $\mathrm{supp}(P')$. We thus get that $P \models \langle\alpha\rangle B$ but not $Q \models \langle\alpha\rangle B$, contradicting $P \doteq Q$.

This proof of the simulation property is different from other such proofs in the literature. For finite branching transition systems, $\mathcal{Q}$ is finite so finite conjunction is enough to define B. For transition systems with the name preservation property, i.e., that if $P \xrightarrow{\alpha} P'$ then $\mathrm{supp}(P') \subseteq \mathrm{supp}(P) \cup \mathrm{supp}(\alpha)$, uniformly bounded conjunction suffices with common support $\mathrm{supp}(P) \cup \mathrm{supp}(Q) \cup \mathrm{supp}(\alpha)$. Without the name preservation property, we here use a not uniformly bounded conjunction in Lemma 8.

4 Derived formulas

Dual connectives. We define logical disjunction $\bigvee_{i \in I} A_i$ in the usual way as $\neg \bigwedge_{i \in I} \neg A_i$, when the indexing set I has bounded cardinality and $\{A_i \mid i \in I\}$ has finite support. A special case is $I = \{1, 2\}$: we then write $A_1 \wedge A_2$ instead of $\bigwedge_{i \in I} A_i$, and dually for $A_1 \vee A_2$. We write $\top$ for the empty conjunction $\bigwedge_{i \in \emptyset}$, and $\bot$ for $\neg\top$. The must modality $[\alpha]A$ is defined as $\neg\langle\alpha\rangle\neg A$, and requires A to hold after every possible α-labelled transition from the current state. For example, $[\alpha](A \wedge B)$ is equivalent to $[\alpha]A \wedge [\alpha]B$, and dually $\langle\alpha\rangle(A \vee B)$ is equivalent to $\langle\alpha\rangle A \vee \langle\alpha\rangle B$.

Quantifiers. Let S be any finitely supported set of bounded cardinality and use v to range over members of S. Write $A\{^v/_x\}$ for the substitution of v for x in A, and assume this substitution function is equivariant. Then we define $\forall x \in S\,.\,A$ as $\bigwedge_{v \in S} A\{^v/_x\}$. There is not necessarily a common finite support for the formulas $A\{^v/_x\}$, for example if S is some term algebra over names, but the set $\{A\{^v/_x\} \mid v \in S\}$ has finite support bounded by $\{x\} \cup \mathrm{supp}(S) \cup \mathrm{supp}(A)$. In our examples in Section 6, substitution is defined inductively

on the structure of formulas, based on primitive substitution functions for actions and state predicates, avoiding capture and preserving the binding names of actions.

Existential quantification $\exists x \in S \,.\, A$ is defined as the dual $\neg\forall x \in S \,.\, \neg A$. When X is a metavariable used to range over a nominal set $\mathcal{X}$, we simply write X for "$X \in \mathcal{X}$". As an example, $\forall a \,.\, A$ means that the formula $A\{{}^{n}/_{a}\}$ holds for all names $n \in \mathcal{N}$.

New name quantifier. The new name quantifier $\text{И}x.A$ intuitively states that $P \models A\{{}^{n}/_{x}\}$ holds where n is a fresh name for P. For example, suppose we have actions of the form $a\,b$ for input, and $\overline{a}\,b$ for output where a and b are free names, then the formula $\text{И}x.[a\,x]\langle\overline{b}\,x\rangle\top$ expresses that whenever a process inputs a fresh name x on channel a, it has to be able to output that name on channel b. If the name received is not fresh (i.e., already present in P) then P is not required to do anything. Therefore this formula is weaker than $\forall x \,.\, [a\,x]\langle\overline{b}\,x\rangle\top$.

To define this formally we use name permutation rather than substitution. Since A and P have finite support, if $P \models (x\,n) \cdot A$ holds for some n fresh for P, by equivariance it also holds for almost all n, i.e., all but finitely many n. Conversely, if it holds for almost all n, it must hold for some $n\#\operatorname{supp}(P)$. Therefore $\text{И}x$ is often pronounced "for almost all x". In other words, $P \models \text{И}x.A$ holds if $\{x \mid P \models A(x)\}$ is a cofinite set of names [25, Definition 3.8]. Letting $\textsc{cof} = \{S \subseteq \mathcal{N} \mid \mathcal{N} \setminus S \text{ is finite}\}$ we thus encode $\text{И}x.A$ as $\bigvee_{S\in\textsc{cof}} \bigwedge_{n\in S}(x\,n)\cdot A$. This formula states there is a cofinite set of names such that for all of them A holds. The support of $\bigwedge_{n\in S}(x\,n)\cdot A$ is bounded by $(\mathcal{N} \setminus S) \cup \operatorname{supp}(A)$ where $S \in \textsc{cof}$, and the support of the encoding $\bigvee_{S\in\textsc{cof}} \bigwedge_{n\in S}(x\,n)\cdot A$ is bounded by $\operatorname{supp}(A)$.

Next step. We generalise the action modality to sets of actions in the following way. If T is a finitely supported set of actions such that $\operatorname{bn}(\alpha)\#A$ for all $\alpha \in T$, we write $\langle T\rangle A$ for $\bigvee_{\alpha\in T}\langle\alpha\rangle A$. The support of the set $\{\langle\alpha\rangle A \mid \alpha \in T\}$ is bounded by $\operatorname{supp}(T) \cup \operatorname{supp}(A)$ and thus finite. Dually, we write $[T]A$ for $\neg\langle T\rangle\neg A$, denoting that A holds after all transitions with actions in T.

To encode the next-step modality, let $\textsc{act}_A = \{\alpha \mid \operatorname{bn}(\alpha)\#A\}$. Note that $\operatorname{supp}(\textsc{act}_A) \subseteq \operatorname{supp}(A)$ is finite. We write $\langle\,\rangle A$ for $\langle\textsc{act}_A\rangle A$, meaning that we can make some (non-capturing) transition to a state where A holds. As an example, $\langle\,\rangle\top$ means that the current state is not deadlocked. The dual modality $[\,]A = \neg\langle\,\rangle\neg A$ means that A holds after every transition from the current state. Larsen [17] uses the same approach to define next-step operators in HML, though his version is less expressive since he uses a finite action set to define the next-step modality.

Fixpoints. Fixpoint operators are a way to introduce recursion into a logic. For example, they can be used to concisely express safety and liveness properties of a transition system, where by safety we mean that some invariant holds for all reachable states, and by liveness that some property will eventually hold. Kozen (1983) [16] introduced the least ($\mu X.A$) and the greatest ($\nu X.A$) fixpoints in modal logic. Intuitively, the least fixpoint states a property that holds for states of a finite path, while the greatest holds for states of an infinite path.

▶ **Theorem 10.** *The least and greatest fixpoint operators are expressible in our HML.*

For the full proofs and definitions, see the appendix of [22]. The idea is to start with an extended language with the forms $\mu X.A$ and X, where X ranges over a countable set of variables and all occurrences of X in A are in the scope of an even number of negations. Write $A(B)$ for the capture-avoiding substitution of B for X in A, and let $A^0(B) = B$ and $A^{i+1}(B) = A(A^i(B))$. Then the encoding of a least fixpoint $\mu X.A$ is $\bigvee_{i\in\mathbb{N}} A^i(\bot)$, given that

fixpoints have been recursively expanded in A. The disjunction has finite support $\mathrm{supp}(A)$, since substitution is equivariant. When interpreting formulas as elements of the power-set lattice of STATES, this encoding yields a fixpoint of $A(\cdot)$: the sequence of formulas $A^i(\bot)$ yields an approximation from below. We define the greatest fixpoint operator $\nu X.A$ in terms of the least as $\neg\mu X.\neg A(\neg X)$.

Using the greatest fixpoint operator we can state global invariants: $\nu X.[\alpha]X \wedge A$ expresses that A holds along all paths labelled with α. Temporal operators such as eventually can also be encoded using the least fixpoint operator: the formula $\mu X.\langle\alpha\rangle X \vee A$ states that eventually A holds along some path labelled with α. We can freely mix the fixpoint operators to obtain formulas like $\nu X.[\alpha]X \wedge (\mu Y.\langle\beta\rangle Y \vee A)$ which means that for each state along any path labelled with α, a state where A holds is reachable along a path labelled with β. Formulas with mixed fixpoint combinators are very expressive, and with the next operator they can encode the branching-time logic CTL* [11].

5 Logics for variants of bisimilarity

The bisimilarity of Section 3 is of the early kind: any substitutive effect of an input (typically replacing a variable with the value received) must have manifested already in the action corresponding to the input, since we apply no substitution to the target state. Alternative treatments of substitutions include late-, open- and hyperbisimilarity, where the input action instead contains the variable to be replaced, and there are different ways to make sure that bisimulations are preserved by relevant substitutions.

In our definition of nominal transition systems there are no particular input variables in the states or in the actions, and thus no a priori concept of "substitution". We therefore choose to formulate the alternatives using so called effect functions. An *effect* is simply a finitely supported function from states to states. For example, in the monadic pi-calculus the effects would be the functions replacing one name by another. In a value-passing calculus the effects would be substitutions of values for variables. In the psi-calculi framework the effects would be sequences of parallel substitutions. Our definitions and results are applicable to any of these; our only requirement is that the effects form a nominal set which we designate by $\mathcal{F}$. Variants of bisimilarity then correspond to requiring continuation after various effects. For example, if the action contains an input variable x then the effects appropriate for late bisimilarity would be substitutions for x.

We will formulate these variants as F/L-*bisimilarity*, where F (for *first*) represents the set of effects that must be observed before following a transition, and L (for *later*) is a function that represents how this set F changes depending on the action of a transition, i.e., $L(\alpha, F)$ is the set of effects that must follow the action α if the previous effect set was F. In the following let $\mathcal{P}_{\mathrm{fs}}(\mathcal{F})$ ranged over by F be the finitely supported subsets of $\mathcal{F}$, and L range over equivariant functions from actions and $\mathcal{P}_{\mathrm{fs}}(\mathcal{F})$ to $\mathcal{P}_{\mathrm{fs}}(\mathcal{F})$.

▶ **Definition 11.** An L-*bisimulation* where $L : \mathrm{ACT} \times \mathcal{P}_{\mathrm{fs}}(\mathcal{F}) \to \mathcal{P}_{\mathrm{fs}}(\mathcal{F})$ is a $\mathcal{P}_{\mathrm{fs}}(\mathcal{F})$-indexed family of symmetric binary relations on states satisfying the following:

If $R_F(P, Q)$ then:

1. *Static implication*: for all $f \in F$ it holds that $f(P) \vdash \varphi$ implies $f(Q) \vdash \varphi$.
2. *Simulation*: For all $f \in F$ and α, P' such that $\mathrm{bn}(\alpha)\#f(Q)$ there exist Q' such that

$$\text{if } f(P) \xrightarrow{\alpha} P' \text{ then } f(Q) \xrightarrow{\alpha} Q' \text{ and } R_{L(\alpha,F)}(P', Q')$$

We write $P \overset{F/L}{\sim} Q$, called F/L-bisimilarity, to mean that there exists an L-bisimulation R such that $R_F(P, Q)$.

Most strong bisimulation varieties can be formulated as F/L-bismilarity. Write $\mathrm{id}_{\mathrm{STATES}}$ for the identity function on states, ID for the singleton set $\{\mathrm{id}_{\mathrm{STATES}}\}$ and $\mathrm{all}_{\mathrm{ID}}$ for the constant function $\lambda(\alpha, F).\mathrm{ID}$.

- *Early bisimilarity*, precisely as defined in Definition 2, is $\mathrm{ID} \,/\, \mathrm{all}_{\mathrm{ID}}$-bisimilarity.
- *Early equivalence*, i.e., early bisimilarity for all possible effects, is $\mathcal{F} \,/\, \mathrm{all}_{\mathrm{ID}}$-bisimilarity.
- *Late bisimilarity* is $\mathrm{ID} \,/\, L$-bisimilarity, where $L(\alpha, F)$ yields the effects that represent substitutions for variables in input actions α (and ID for other actions).
- *Late equivalence* is similarly $\mathcal{F} \,/\, L$-bisimilarity.
- *Open bisimilarity* is $\mathcal{F} \,/\, L$-bisimilarity where $L(\alpha, F)$ is the set F minus all effects that change bound output names in α.
- *Hyperbisimilarity* is $\mathcal{F} \,/\, \lambda(\alpha, F).\mathcal{F}$-bisimilarity.

All of the above are generalizations of known and well-studied definitions. The original value-passing variant of CCS [18] uses early bisimilarity. The original bisimilarity for the pi-calculus is of the late kind [19], where it also was noted that late equivalence is the corresponding congruence. Early bisimilarity and equivalence and open bisimilarity for the pi-calculus were introduced in 1993 [20, 26], and hyperbisimilarity for the fusion calculus in 1998 [23].

In view of this we only need to provide a modal logic adequate for F/L-bisimilarity; it can then immediately be specialized to all of the above variants. For this we introduce a new kind of logical operator as follows.

▶ **Definition 12.** For each $f \in \mathcal{F}$ the logical unary *effect consequence* operator $\langle f \rangle$ has the definition

$$P \models \langle f \rangle A \quad \text{if} \quad f(P) \models A$$

Thus the formula $\langle f \rangle A$ means that A holds if the effect f is applied to the state. Note that by definition this distributes over conjunction and negation, e.g. $P \models \neg\langle f \rangle A$ iff $P \models \langle f \rangle \neg A$ iff not $f(P) \models A$ etc. The effect consequence operator is similar in spirit to the action modalities: both $\langle f \rangle A$ and $\langle \alpha \rangle A$ assert that something (an effect or action) must be possible and that A holds afterwards. Indeed, effects can be viewed as a special case of transitions (as formalised in Definition 16 below) which is why we give the operators a common syntactic appearance.

Now define the formulas that can directly use effects from F and after actions use effects according to L, ranged over by $A^{F/L}$, in the following way:

▶ **Definition 13.** Given L as in Definition 11, for all $F \in \mathcal{P}_{\mathrm{fs}}(\mathcal{F})$ define $\mathcal{A}^{F/L}$ as the set of formulas given by the mutually recursive definitions:

$$A^{F/L} \quad ::= \quad \bigwedge_{i \in I} A_i^{F/L} \quad | \quad \neg A^{F/L} \quad | \quad \langle f \rangle \varphi \quad | \quad \langle f \rangle \langle \alpha \rangle A^{L(\alpha,F)/L}$$

where we require $f \in F$ and that the conjunction has bounded cardinality and finite support.

Let $P \overset{F/L}{=} Q$ mean that P and Q satisfy the same formulas in $\mathcal{A}^{F/L}$.

▶ **Theorem 14.** $P \overset{F/L}{\sim} Q \quad \Leftrightarrow \quad P \overset{F/L}{=} Q$

Proof: The direction $\Rightarrow$ is a generalization of Theorem 6. The other direction is a generalization of Theorem 9: we prove that $\overset{F/L}{=}$ is an F/L-bisimulation. It needs a variant of Lemma 8:

▶ **Lemma 15.** *If $A \in \mathcal{A}^{F/L}$ is a distinguishing formula for P and Q, then there exists a distinguishing formula $B \in \mathcal{A}^{F/L}$ for P and Q such that* $\mathrm{supp}(B) \subseteq \mathrm{supp}(P, F)$.

The proof is an easy generalisation of Lemma 8.

An alternative to the effect consequence operators is to transform the transition system such that standard (early) bisimulation on the transforms coincides with F/L-bisimilarity. The idea is to let the effect function be part of the transition relation, thus $f(P) = P'$ becomes $P \xrightarrow{f} P'$.

▶ **Definition 16.** Assume $\mathcal{F}$ and L as above. The *L-transform* of a nominal transition system $\mathbf{T}$ is a nominal transition system where:

- The states are of the form $\textsc{ac}(F, f(P))$ and $\textsc{ef}(F, P)$, for $f \in F \in \mathcal{P}_{\mathrm{fs}}(\mathcal{F})$ and states P of $\mathbf{T}$. The intuition is that states of kind AC can perform ordinary actions, and states of kind EF can commit effects.
- The state predicates are those of $\mathbf{T}$.
- $\textsc{ac}(F, P) \vdash \varphi$ if in $\mathbf{T}$ it holds $P \vdash \varphi$, and $\textsc{ef}(F, P) \vdash \varphi$ never holds.
- The actions are the actions of $\mathbf{T}$ and the effects in $\mathcal{F}$.
- bn is as in $\mathbf{T}$, and additionally $\mathrm{bn}(f) = \emptyset$ for $f \in \mathcal{F}$.
- The transitions are of two kinds. If in $\mathbf{T}$ it holds $P \xrightarrow{\alpha} P'$, then there is a transition $\textsc{ac}(F, P) \xrightarrow{\alpha} \textsc{ef}(L(\alpha, F), P')$. And for each $f \in F$ it holds $\textsc{ef}(F, P) \xrightarrow{f} \textsc{ac}(F, f(P))$.

▶ **Theorem 17.** *$P \overset{F/L}{\sim} Q$ in $\mathbf{T}$ if and only if $\textsc{ef}(F, P) \dot{\sim} \textsc{ef}(F, Q)$ in the L-transform of $\mathbf{T}$.*

The proof idea is that from an F/L-bisimulation in $\mathbf{T}$ it is easy to construct an (ordinary) bisimulation in the L-transform of $\mathbf{T}$, and vice versa. A direct consequence is that $P \overset{F/L}{\sim} Q$ iff $\textsc{ef}(F, P) \doteq \textsc{ef}(F, Q)$ in the L-transform of $\mathbf{T}$. Here the actions in the logic would include effects $f \in \mathcal{F}$.

6 Related work and examples

In this first part of this section we discuss other modal logics for process calculi, with a focus on how their constructors can be captured by finitely supported conjunction in our HML. This comparison is by necessity somewhat informal; a fully formal correspondence would fail to hold in many cases due to differences in the conjunction operator of the logic (finite, uniformly bounded or unbounded vs. bounded support). In the later part of this section, we obtain novel, adequate HMLs for more recent process calculi.

HML for CCS. The first published HML is Hennessy and Milner (1985) [15]. They use finite (binary) conjunction with the assumption of image-finiteness for ordinary CCS. The same goes for the value-passing calculus and logic by Hennessy and Liu (1995) [14], where image-finiteness is due to a late semantics and the logic contains quantification over data values. A similar idea and argument is in a logic for LOTOS by Calder et al. (2002) [8], though that only considers stratified bisimilarity up to ω.

Hennessy and Liu's value-passing calculus is based on abstractions $(x)P$ and concretions (v, P) where v is drawn from a set of values. To encode the modalities of their logic in ours, we add effects $\mathrm{id}_{\mathrm{STATES}}$ and $?v$, with $?v((x)P) = P\{{}^{v}/_{x}\}$, and transitions $(v, P) \xrightarrow{!v} P$. Letting $L(a?, \_) = \{?v \mid v \in \text{values}\}$ and $L(\alpha, \_) = \{\mathrm{id}_{\mathrm{STATES}}\}$ otherwise, late bisimilarity is $\{\mathrm{id}_{\mathrm{STATES}}\}/L$-bisimilarity as defined in Section 5. We can then encode their universal

quantifier $\forall x.A$ as $\bigwedge_v \langle ?v\rangle A\{^v/_x\}$, which has support $\mathrm{supp}(A)\setminus\{x\}$, and their output modality $\langle c!x\rangle A$ as $\langle c!\rangle \bigvee_v \langle !v\rangle A\{^v/_x\}$, with support $\{c\}\cup(\mathrm{supp}(A)\setminus\{x\})$.

An infinitary HML for CCS is discussed in Milner's book (1989) [18], where also the process syntax contains infinite summation. There are no restrictions on the indexing sets and no discussion about how this can exhaust all names. The adequacy theorem is proved by stratifying bisimilarity and using transfinite induction over all ordinals, where the successor step basically is the contraposition of the argument in Theorem 9, though without any consideration of finite support. A more rigorous treatment of the same ideas is by Abramsky (1991) [3] where uniformly bounded conjunction is used throughout.

Pi-calculus. The first HML for the pi-calculus is by Milner et al. (1993) [20], where infinite conjunction is used in the early semantics and conjunctions are restricted to use a finite set of free names. The adequacy proof is of the same structure as in this paper. The logic defined in this paper, applied to the pi-calculus transition system omitting bound input actions $x(y)$, contains the logic $\mathcal{F}$ of Milner et al., or the equipotent logic $\mathcal{FM}$ if we take the set of name matchings $[a=b]$ as state predicates.

Spi Calculus. Frendrup et al. (2002) [12] provide three Hennessy-Milner logics for the spi calculus [2]. The action modalities in Frendrup's logic only use parts of the labels: on process output, the modality $\langle\overline{a}\rangle$ tests only the channel used. On process input, the modality $\langle a\xi\rangle$ describes how the observer σ computed the received message $M=\mathbf{e}(\xi\sigma)$, where ξ is an expression that may contain decryptions and projections, and $\mathrm{supp}(\xi)\setminus\mathrm{dom}(\sigma)$ is fresh for P and σ. Simplifying the labels of the transition system to τ and the aforementioned $\overline{a}$ and $a\xi$ labels, our minimal HML applied to the particular nominal transition system of the spi calculus has the same modalities as the logic $\mathcal{F}$ of Frendrup et al., although the latter uses infinite conjunction without any mechanism to prevent formulas from exhausting all names, leaving none available for alpha-conversion. Thus their notion of substitution is not formally well defined.

Their logic $\mathcal{EM}$ replaces the simple input modality by an early input modality $\langle\underline{a}(x)\rangle^E A$, which (after a minor manipulation of the input labels) can be encoded as the conjunction $\bigwedge_\xi \langle a\xi\rangle A\{^\xi/_x\}$, which has support $\mathrm{supp}(A)\setminus\{x\}$. We do not consider their logic $\mathcal{LM}$ that uses a late input modality, since its application relies on sets that do not have finite support [12, Theorem 6.12], which are not meaningful in nominal logic.

Applied Pi-calculus. A more recent work is a logic by Pedersen (2006) [24] for the applied pi-calculus [1], where the adequacy theorem uses image-finiteness of the semantics in the contradiction argument. The logic contains atomic formulae for equality in the frame of a process, corresponding to our state predicates. The main difference to our logic is an early input modality and a quantifier $\exists x$.

Their early input modality $\langle\underline{a}(x)\rangle A$ can be straightforwardly encoded as the conjunction $\bigwedge_M \langle\underline{a}M\rangle A\{^M/_x\}$, with support $\{a\}\cup(\mathrm{supp}(A)\setminus\{x\})$. For the existential quantifier, there is a requirement that the received term M can be computed from the current knowledge available to an observer of the process, which we here write $M\in\mathcal{S}(P)$. We add actions M/x with $\mathrm{bn}(M/x)=x$ and transitions $P\xrightarrow{M/x}P\mid\{^M/_x\}$ if $M\in\mathcal{S}(P)$ and $x\#P$. We can then encode $\exists x.A$ as $\bigvee_M\langle M/x\rangle A$, which has support $\mathrm{supp}(A)\setminus\{x\}$.

Fusion calculus. In an HML for the fusion calculus by Haugstad et al. (2006) [13] the fusions (i.e., equality relations on names) are action labels φ. The corresponding modal

operator $\langle\varphi\rangle A$ has the semantics that the formula A must be satisfied for all substitutive effects of φ (intuitively, substitutions that map each name to a fixed representative for its equivalence class). By making the substitutive effects of fusion actions visible in the transition system, we can encode this modal operator. Their adequacy theorem uses the contradiction argument with infinite conjunction, with no argument about finiteness of names for the distinguishing formula.

Nominal transition systems. De Nicola and Loreti (2008) [10] define a general format for nominal transition systems and an associated modal logic, that is adequate for image-finite transition systems only and uses several different modalities for name revelation and resource consumption. In contrast, we seek a small and expressive HML for general nominal transition systems. Indeed, the logic of De Nicola and Loreto can be seen as a special case of ours: their different transition systems can be merged into a single one, and we can encode their quantifiers and fixpoint operator as described in Section 4. Nominal SOS of Cimini et al. (2012) [9] is also a special case of nominal transition systems.

In each of the final two examples below, no HML has to our knowledge yet been proposed, and we immediately obtain one by instantiating the logic in the present paper.

Concurrent constraint pi calculus. The concurrent constraint pi calculus (CC-pi) by Buscemi and Montanari (2007) [6] extends the explicit fusion calculus [28] with a more general notion of constraint stores c. The reference equivalence for CC-pi is open bisimulation [7] (closely corresponding to hyperbisimulation in the fusion calculus [23]), which differs from labelled bisimulation in two ways: First, two equivalent processes must be equivalent under all store extensions. To encode this, we let the effects $\mathcal{F}$ be the set of constraint stores c different from 0, and let $c(P) = c \mid P$. Second, when simulating a labelled transition $P \xrightarrow{\alpha} P'$, the simulating process Q can use any transition $Q \xrightarrow{\beta} Q'$ with an equivalent label, as given by a state predicate $\alpha = \beta$. As an example, if $\alpha = \overline{a}\langle x\rangle$ is a free output label then $P \vdash \alpha = \beta$ iff $\beta = \overline{b}\langle y\rangle$ where $P \vdash a = b$ and $P \vdash x = y$. To encode this, we transform the labels of the transition system by replacing them with their equivalence classes, i.e., $P \xrightarrow{\alpha} P'$ becomes $P \xrightarrow{[\alpha]_P} P'$ where $\beta \in [\alpha]_P$ iff $P \vdash \beta = \alpha$. Hyperbisimilarity (Definition 11) on this transition system then corresponds to open bisimilarity, and the modal logic defined in Section 5 is adequate.

Psi-calculi. In psi-calculi by Bengtson et al. (2011) [4], the labelled transitions take the form $\Psi \triangleright P \xrightarrow{\alpha} P'$, where the assertion environment Ψ is unchanged after the step. We model this as a nominal transition system by letting the set of states be pairs (Ψ, P) of assertion environments and processes, and define the transition relation by $(\Psi, P) \xrightarrow{\alpha} (\Psi, P')$ if $\Psi \triangleright P \xrightarrow{\alpha} P'$. The notion of bisimulation used with psi-calculi also uses an assertion environment and is required to be closed under environment extension, i.e., if $\Psi \triangleright P \sim Q$, then $\Psi \otimes \Psi' \triangleright P \sim Q$ for all Ψ'. We let the effects $\mathcal{F}$ be the set of assertions, and define $\Psi((\Psi', P)) = (\Psi \otimes \Psi', P)$. Hyperbisimilarity on this transition system then subsumes the standard psi-calculi bisimilarity, and the modal logic defined in Section 5 is adequate.

7 Conclusion

We have given a general account of transition systems and Hennessy-Milner Logic using nominal sets. The advantage of our approach is that it is more expressive than previous work. We allow infinite conjunctions that are not uniformly bounded, meaning that we can

encode e.g. quantifiers and the next-step operator. We have given ample examples of how the definition captures different variants of bisimilarity and how it relates to many different versions of HML in the literature.

We have formalized the results of Section 3, including Theorems 6 and 9, using Nominal Isabelle [27].[1] Nominal Isabelle is an implementation of nominal logic in Isabelle/HOL [21], a popular interactive proof assistant for higher-order logic. It adds convenient specification mechanisms for, and automation to reason about, datatypes with binders.

However, Nominal Isabelle does not directly support infinitely branching datatypes. Therefore, the mechanization of formulas (Definition 3) was challenging. We construct formulas from first principles in higher-order logic, by defining an inductive datatype of *raw* formulas (where alpha-equivalent raw formulas are *not* identified). The datatype constructor for conjunction recurses through sets of raw formulas of bounded cardinality, a feature made possible only by a recent re-implementation of Isabelle/HOL's datatype package [5].

We then define alpha-equivalence of raw formulas. For finitely branching datatypes, alpha-equivalence is based on a notion of free variables. Here, to obtain the correct notion of free variables of a conjunction, we define alpha-equivalence and free variables via mutual recursion. This necessitates a fairly involved termination proof. (All recursive functions in Isabelle/HOL must be terminating.) To obtain formulas, we quotient raw formulas by alpha-equivalence, and finally carve out the subtype of all terms that can be constructed from finitely supported ones. We then prove important lemmas; for instance, a strong induction principle for formulas that allows the bound names in $\langle\alpha\rangle A$ to be chosen fresh for any finitely supported context.

Our development, which in total consists of about 2700 lines of Isabelle definitions and proofs, generalizes the constructions that Nominal Isabelle performs for finitely branching datatypes to a type with infinite branching. To our knowledge, this is the first mechanization of an infinitely branching nominal datatype in a proof assistant.

Acknowledgements. We thank Andrew Pitts for enlightening discussions on nominal datatypes with infinitary constructors, and Dmitriy Traytel for providing a formalization of cardinality-bounded sets.

References

1 Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of POPL'01*, pages 104–115. ACM, January 2001.

2 Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The Spi calculus. *Journal of Information and Computation*, 148(1):1–70, 1999.

3 Samson Abramsky. A domain equation for bisimulation. *Journal of Information and Computation*, 92(2):161–218, 1991.

4 Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: a framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science*, 7(1), 2011.

5 Jasmin Christian Blanchette, Johannes Hölzl, Andreas Lochbihler, Lorenz Panny, Andrei Popescu, and Dmitriy Traytel. Truly modular (co)datatypes for Isabelle/HOL. In G. Klein and R. Gamboa, editors, *Proc. of ITP 2014*, volume 8558 of *LNCS*, pages 93–110. Springer, 2014.

[1] Our Isabelle theories are available at `https://github.com/tjark/ML-for-NTS`.

6 Maria Grazia Buscemi and Ugo Montanari. CC-Pi: A constraint-based language for specifying service level agreements. In Rocco De Nicola, editor, *Proceedings of ESOP 2007*, volume 4421 of *LNCS*, pages 18–32. Springer, 2007.

7 Maria Grazia Buscemi and Ugo Montanari. Open bisimulation for the concurrent constraint pi-calculus. In Sophia Drossopoulou, editor, *Proceedings of ESOP 2008*, volume 4960 of *LNCS*, pages 254–268. Springer, 2008.

8 Muffy Calder, Savi Maharaj, and Carron Shankland. A modal logic for full LOTOS based on symbolic transition systems. *The Computer Journal*, 45(1):55–61, 2002.

9 Matteo Cimini, Mohammad Reza Mousavi, Michel A. Reniers, and Murdoch J. Gabbay. Nominal SOS. *Electron. Notes Theor. Comput. Sci.*, 286:103–116, September 2012.

10 Rocco De Nicola and Michele Loreti. Multiple-labelled transition systems for nominal calculi and their logics. *Mathematical Structures in Computer Science*, 18(1):107–143, 2008.

11 E. Allen Emerson. Model checking and the mu-calculus. In *DIMACS Series in Discrete Mathematics*, pages 185–214. American Mathematical Society, 1997.

12 Ulrik Frendrup, Hans Hüttel, and Jesper Nyholm Jensen. Modal logics for cryptographic processes. *Electr. Notes Theor. Comput. Sci.*, 68(2):124–141, 2002.

13 Arild Martin Møller Haugstad, Anders Franz Terkelsen, and Thomas Vindum. A modal logic for the fusion calculus. Unpublished, University of Aalborg, `http://vbn.aau.dk/ws/files/61067487/1149104946.pdf`, 2006.

14 Matthew Hennessy and Xinxin Liu. A modal logic for message passing processes. *Acta Informatica*, 32(4):375–393, 1995.

15 Matthew Hennessy and Robin Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985.

16 Dexter Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27(3):333–354, 1983. Special Issue Ninth International Colloquium on Automata, Languages and Programming (ICALP) Aarhus, Summer 1982.

17 Kim G. Larsen. Proof systems for Hennessy-Milner logic with recursion. In M. Dauchet and M. Nivat, editors, *Proc. of CAAP'88*, volume 299 of *LNCS*, pages 215–230. Springer, 1988.

18 Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

19 Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.

20 Robin Milner, Joachim Parrow, and David Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114(1):149–171, 1993.

21 Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

22 Joachim Parrow, Johannes Borgström, Lars-Henrik Eriksson, Ramūnas Gutkovas, and Tjark Weber. Modal logics for nominal transition systems. Technical Report 2015-021, Department of Information Technology, Uppsala University, June 2015.

23 Joachim Parrow and Björn Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proc. of LICS 1998*, pages 176–185. IEEE CS Press, 1998.

24 Michael Pedersen. Logics for the applied pi calculus. Master's thesis, Aalborg University, 2006. BRICS RS-06-19.

25 Andrew M. Pitts. *Nominal Sets*. Cambridge University Press, 2013.

26 Davide Sangiorgi. A theory of bisimulation for the π-calculus. In Eike Best, editor, *Proceedings of CONCUR'93*, volume 715 of *LNCS*, pages 127–142. Springer, 1993.

27 Christian Urban and Cezary Kaliszyk. General bindings and alpha-equivalence in Nominal Isabelle. *Logical Methods in Computer Science*, 8(2), 2012.

28 Lucian Wischik and Philippa Gardner. Explicit fusions. *Theoretical Computer Science*, 304(3):606–630, 2005.

Howe's Method for Contextual Semantics*

Sergueï Lenglet[1] **and Alan Schmitt**[2]

1 **Université de Lorraine, France**
serguei.lenglet@univ-lorraine.fr
2 **Inria, France**
alan.schmitt@inria.fr

Abstract

We show how to use Howe's method to prove that context bisimilarity is a congruence for process calculi equipped with their usual semantics. We apply the method to two extensions of HOπ, with passivation and with join patterns, illustrating different proof techniques.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Bisimulations, process calculi, Howe's Method

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.212

1 Introduction

Process equivalence relates processes whose behavior may not be distinguished, even when inserted in arbitrary contexts. Equivalent processes may thus be used interchangeably in any larger system, with no observable difference. This property is quite strong, and to prove it directly, one has to consider every possible context. Much effort has thus been applied to techniques that simplify the proofs of process equivalence. Such techniques often involve the definition of a relation between processes that is easier to establish. The relation, typically a form of *bisimilarity*, is then shown to characterize process equivalence. This characterization has two parts: bisimilarity is *sound* – bisimilar processes are equivalent – and *complete* – equivalent processes are bisimilar.

As process equivalence is generally intended to be preserved by every context, it is often a congruence. Hence a sound and complete bisimilarity also has to be a congruence. Even when considering sound (but not complete) bisimilarities, it is very convenient that they be congruences. Indeed, to prove that two processes are equivalent, one can then simply show they have the same external structure (context) with bisimilar processes inside. Proving congruence is thus a crucial step when working with process equivalence.

Howe's method [7] is a powerful approach to show that a bisimilarity is a congruence. In a nutshell, it reverses the problem: first define a relation, called "Howe's closure", that includes the bisimilarity of interest and is a congruence by definition. Second, show it is a bisimulation. As bisimilarity contains every bisimulations, Howe's closure is thus included in bisimilarity. Third, conclude that the bisimilarity and its Howe's closure coincide, thus the former is a congruence.

This approach works well in a functional setting. Until now, its application to higher-order process calculi has required significant adjustments, either yielding a sound but not complete bisimilarity [5], or requiring the definition of a new semantics [11]. We present a direct

* This work has been partially supported by the ANR project 2010-BLAN-0305 PiCoq and the PHC Polonium project No. 33271XH.

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 212–225

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

application of Howe's method for the higher-order π calculus (HOπ) with its usual semantics, and state the central *pseudo-simulation* property that enables the application of the method (Section 2). We then detail two approaches to prove this lemma for two extensions of HOπ: one with *passivation* (Section 3), the other with *join-patterns* (Section 4). The complete proofs are available in an accompanying research report [10].

2 Howe's Method in HOπ with Contextual Semantics

2.1 Syntax and Contextual Semantics

We recall the syntax and contextual semantics of (the process-passing fragment of) HOπ [14] in Figure 1, omitting the symmetric rules for Par and HO. We use a, b, c to range over channel names, $\overline{a}$, $\overline{b}$, $\overline{c}$ to range over conames, γ to range over names and conames, and X, Y to range over process variables. We define $\overline{\overline{a}}$ as a. Multisets $\{x_1 \dots x_n\}$ (where x ranges over some entities) are written $\widetilde{x}$. Finally, we write $\uplus$ for multiset union.

An input $a(X)P$ binds X in P, and a restriction $\nu a.P$ binds a in P. We write $\mathrm{fv}(P)$ for the free variables of a process P and $\mathrm{fn}(P)$ for its free names. A *closed process* has no free variable. We identify processes up to α-conversion of names and variables: processes and agents are always chosen such that their bound names and variables are pairwise distinct, and distinct from their free names and variables. We write $P\{Q/X\}$ for the capture-free substitution of X by Q in P. *Structural congruence* $\equiv$ equates processes up to reorganization of their sub-processes and their name restrictions; it is the smallest congruence verifying the rules of Figure 1. Because the ordering of restrictions does not matter, we abbreviate $\nu a_1 \dots \nu a_n.P$ as $\nu\widetilde{a}.P$; since bound names are pairwise distinct, $\widetilde{a}$ is a set.

We define a labeled transition system (LTS), where agents transition to processes, *abstractions* F of the form $(X)Q$, or *concretions* C of the form $\nu\widetilde{b}.\langle R\rangle S$. Like for processes, the ordering of restrictions does not matter for a concretion, therefore we write them using a set of names $\widetilde{b}$; in particular, we write $\langle R\rangle S$ if $\widetilde{b} = \emptyset$. Labels of the LTS are ranged over by α. Transitions are either an *internal action* $P \xrightarrow{\tau} P'$, a *message input* $P \xrightarrow{a} F$, or a *message output* $P \xrightarrow{\overline{a}} C$. The transition $P \xrightarrow{a} (X)Q$ means that P may receive a process R on a to continue as $Q\{R/X\}$. The transition $P \xrightarrow{\overline{a}} \nu\widetilde{b}.\langle R\rangle S$ means that P may send the process R on a and then continue as S, and the scope of the names $\widetilde{b}$ has to be expanded to encompass the recipient of R. A higher-order communication takes place when a concretion interacts with an abstraction (rule HO).

2.2 Behavioral Equivalences

Barbed congruence relates processes based on their observable actions, or *barbs*. The observable actions γ of a process P, written $P \downarrow_\gamma$, are unrestricted names or conames on which a communication may immediately occur ($P \xrightarrow{\gamma} A$, for some A). A context $\mathbb{C}$ is a term with a single hole $\Box$, that may be filled with a process P, written $\mathbb{C}\{P\}$; the free names or free variables of P may be captured by $\mathbb{C}$. An equivalence relation $\mathcal{R}$ is a *congruence* if $P\,\mathcal{R}\,Q$ implies $\mathbb{C}\{P\}\,\mathcal{R}\,\mathbb{C}\{Q\}$ for all contexts $\mathbb{C}$.

▶ **Definition 1.** A symmetric relation $\mathcal{R}$ on closed processes is a strong barbed bisimulation if $P\,\mathcal{R}\,Q$ implies:

- $P \downarrow_\gamma$ implies $Q \downarrow_\gamma$;
- if $P \xrightarrow{\tau} P'$, then there exists Q' such that $Q \xrightarrow{\tau} Q'$ and $P'\,\mathcal{R}\,Q'$.

Syntax: $P ::= \mathbf{0} \mid X \mid P|P \mid a(X)P \mid \overline{a}\langle P\rangle P \mid \nu a.P$

Agents: $A, B ::= P \mid F \mid C$

Abstractions $F, G ::= (X)P$ Concretions $C, D ::= \langle P\rangle Q \mid \nu a.C$

Extension of operators to abstractions and concretions

$$(X)Q|P \triangleq (X)(Q|P) \text{ if } X \notin \mathrm{fv}(P)$$
$$P|(X)Q \triangleq (X)(P|Q) \text{ if } X \notin \mathrm{fv}(P)$$
$$\nu a.(X)P \triangleq (X)\nu a.P$$
$$(\nu\widetilde{b}.\langle Q\rangle R)|P \triangleq \nu\widetilde{b}.\langle Q\rangle(R|P) \text{ if } \widetilde{b} \cap \mathrm{fn}(P) = \emptyset$$
$$P|(\nu\widetilde{b}.\langle Q\rangle R) \triangleq \nu\widetilde{b}.\langle Q\rangle(P|R) \text{ if } \widetilde{b} \cap \mathrm{fn}(P) = \emptyset$$
$$\nu a.(\nu\widetilde{b}.\langle Q\rangle R) \triangleq \nu a,\widetilde{b}.\langle Q\rangle R \text{ if } a \in \mathrm{fn}(\nu\widetilde{b}.Q)$$
$$\nu a.(\nu\widetilde{b}.\langle Q\rangle R) \triangleq \nu\widetilde{b}.\langle Q\rangle \nu a.R \text{ if } a \notin \mathrm{fn}(\nu\widetilde{b}.Q)$$

Pseudo-application and process application

$$(X)P \bullet \nu\widetilde{b}.\langle R\rangle Q \triangleq \nu\widetilde{b}.(P\{R/X\}|Q) \text{ if } \widetilde{b} \cap \mathrm{fn}(P) = \emptyset \qquad (X)P \circ Q \triangleq P\{Q/X\}$$

Structural congruence

$$(P|Q)|R \equiv P|(Q|R) \qquad P|Q \equiv Q|P \qquad P|\mathbf{0} \equiv P \qquad \nu a.\nu b.P \equiv \nu b.\nu a.P$$
$$P|\nu a.Q \equiv \nu a.(P|Q) \text{ if } a \notin \mathrm{fn}(P) \qquad \nu a.P \equiv P \text{ if } a \notin \mathrm{fn}(P)$$

LTS rules: $\alpha ::= \tau \mid a \mid \overline{a}$

$$a(X)P \xrightarrow{a} (X)P \ \textsc{In} \qquad \overline{a}\langle Q\rangle P \xrightarrow{\overline{a}} \langle Q\rangle P \ \textsc{Out} \qquad \frac{P \xrightarrow{\alpha} A}{P|Q \xrightarrow{\alpha} A|Q} \ \textsc{Par}$$

$$\frac{P \xrightarrow{\alpha} A \qquad \alpha \notin \{a, \overline{a}\}}{\nu a.P \xrightarrow{\alpha} \nu a.A} \ \textsc{Restr} \qquad \frac{P \xrightarrow{a} F \qquad Q \xrightarrow{\overline{a}} C}{P|Q \xrightarrow{\tau} F \bullet C} \ \textsc{HO}$$

Figure 1 Contextual LTS for HOπ.

Two processes P, Q are strong barbed congruent, written $P \sim_b Q$, if for all context $\mathbb{C}$, there exists a strong barbed bisimulation $\mathcal{R}$ such that $\mathbb{C}\{P\}\ \mathcal{R}\ \mathbb{C}\{Q\}$.

A relation $\mathcal{R}$ is *sound* with respect to $\sim_b$ if $\mathcal{R} \subseteq \sim_b$; $\mathcal{R}$ is *complete* with respect to $\sim_b$ if $\sim_b \subseteq \mathcal{R}$. In [14], barbed congruence is characterized by a (strong) *context bisimilarity*, defined as follows.

▶ **Definition 2.** A relation $\mathcal{R}$ on closed processes is a context simulation if $P\ \mathcal{R}\ Q$ implies:
- for all $P \xrightarrow{\tau} P'$, there exists Q' such that $Q \xrightarrow{\tau} Q'$ and $P'\ \mathcal{R}\ Q'$;
- for all $P \xrightarrow{a} F$, for all C, there exists F' such that $Q \xrightarrow{a} F'$ and $F \bullet C\ \mathcal{R}\ F' \bullet C$;
- for all $P \xrightarrow{\overline{a}} C$, for all F, there exists C' such that $Q \xrightarrow{\overline{a}} C'$ and $F \bullet C\ \mathcal{R}\ F \bullet C'$.

A relation $\mathcal{R}$ is a context bisimulation if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are context simulations. Context bisimilarity, written $\sim$, is the largest context bisimulation.

The definition is written in the *early* style, because the answer $Q \xrightarrow{a} F'$ depends on the particular C considered in the input case, and $Q \xrightarrow{\overline{a}} C'$ depends on F in the output case. In

the *late* style, this dependency is broken by moving the universal quantification on C or F after the existential one on F' or C'.

We extend the equivalences to open terms by defining the *open extension* of a relation $\mathcal{R}$.

▶ **Definition 3.** For two open processes P and Q, $P\ \mathcal{R}^\circ\ Q$ holds if $P\sigma\ \mathcal{R}\ Q\sigma$ holds for all process substitutions σ that close P and Q.

Conversely, we write $\mathcal{R}_c$ for the relation $\mathcal{R}$ restricted to closed processes.

In the following, we use (bi)simulation up to structural congruence, a (bi)simulation proof technique which allows to use $\equiv$ when relating processes.

▶ **Definition 4.** A relation $\mathcal{R}$ is a context simulation up to $\equiv$ if $P\ \mathcal{R}\ Q$ implies the clauses of Definition 2, where $\mathcal{R}$ is changed into $\equiv\mathcal{R}\equiv$.

Since $\equiv$ is a context bisimulation, the resulting proof technique is sound.

▶ **Lemma 5.** *If $\mathcal{R}$ is a context bisimulation up to $\equiv$, then $\mathcal{R} \subseteq \sim$.*

Context bisimilarity is sound and complete. The congruence proof of [14] does not apply, however, to certain process calculi, such as the ones with passivation [11]. For this reason, other congruence proof techniques, such as Howe's method [7], have been considered.

2.3 Howe's Method

We sketch the principles behind Howe's method and recall why its application to (early) context bisimilarity has been deemed problematic.

Howe's method [7, 6] is a systematic proof technique to show that a bisimilarity $\mathcal{B}$ (and its open extension $\mathcal{B}^\circ$) is a congruence. The method can be divided in three steps: first, prove some basic properties on the *Howe's closure* $\mathcal{B}^\bullet$ of the relation. By construction, $\mathcal{B}^\bullet$ contains $\mathcal{B}^\circ$ and is a congruence. Second, prove a simulation-like property for $\mathcal{B}^\bullet$. Finally, prove that $\mathcal{B}$ and $\mathcal{B}^\bullet$ coincide on closed processes. Since $\mathcal{B}^\bullet$ is a congruence, then so is $\mathcal{B}$.

Given a relation $\mathcal{R}$, Howe's closure is inductively defined as the smallest congruence which contains $\mathcal{R}^\circ$ and is closed under right composition with $\mathcal{R}^\circ$.

▶ **Definition 6.** Howe's closure $\mathcal{R}^\bullet$ of a relation $\mathcal{R}$ is defined inductively by the following rules, where op ranges over the operators of the language.

$$\frac{P\ \mathcal{R}^\circ\ Q}{P\ \mathcal{R}^\bullet\ Q} \qquad \frac{P\ \mathcal{R}^\bullet\ P' \qquad P'\ \mathcal{R}^\circ\ Q}{P\ \mathcal{R}^\bullet\ Q} \qquad \frac{\widetilde{P}\ \mathcal{R}^\bullet\ \widetilde{Q}}{\mathsf{op}(\widetilde{P})\ \mathcal{R}^\bullet\ \mathsf{op}(\widetilde{Q})}$$

Instantiating $\mathcal{R}$ as $\mathcal{B}$, $\mathcal{B}^\bullet$ is a congruence by definition. The composition with $\mathcal{B}^\circ$ enables some transitivity and additional properties. In particular, we can prove that $\mathcal{B}^\bullet$ is *substitutive*: if $P\ \mathcal{B}^\bullet\ Q$ and $R\ \mathcal{B}^\bullet\ S$, then $P\{R/X\}\ \mathcal{B}^\bullet\ Q\{S/X\}$. By definition, we have $\mathcal{B}^\circ \subseteq \mathcal{B}^\bullet$; for the reverse inclusion to hold, we prove that $\mathcal{B}^\bullet$ is a bisimulation, hence it is included in the bisimilarity. To this end, we first prove that $\mathcal{B}^\bullet$ (restricted to closed terms) is a simulation, using a pseudo-simulation lemma (second step of the method, discussed below). We then use the following result on the reflexive and transitive closure $(\mathcal{B}^\bullet)^*$ of $\mathcal{B}^\bullet$.

▶ **Lemma 7.** *Let $\mathcal{R}$ be an equivalence. Then $(\mathcal{R}^\bullet)^*$ is symmetric.*

If $\mathcal{B}^\bullet$ is a simulation, then $(\mathcal{B}^\bullet)^*$ (restricted to closed terms) is also a simulation. By Lemma 7, $(\mathcal{B}^\bullet)^*$ is in fact a bisimulation. Consequently, we have $\mathcal{B} \subseteq \mathcal{B}^\bullet \subseteq (\mathcal{B}^\bullet)^* \subseteq \mathcal{B}$ on closed terms, and we conclude that $\mathcal{B}$ is a congruence.

The main challenge is stating and proving a simulation-like property for the Howe's closure $\mathcal{B}^\bullet$ of a bisimilarity $\mathcal{B}$. The labels λ of a LTS $\xrightarrow{\lambda}$ of a higher-order language usually contain or depend on terms (e.g., in the λ-calculus, λ-abstractions are labels), so the technique generally extends $\mathcal{B}^\bullet$ to labels. The simulation-like property then follows the pattern below, similar to a higher-order bisimilarity clause as in Plain CHOCS [18].

If $P\ \mathcal{B}^\bullet\ Q$ and $P \xrightarrow{\lambda} A$, then for all $\lambda\ \mathcal{B}^\bullet\ \lambda'$, there exists B such that $Q \xrightarrow{\lambda'} B$ and $A\ \mathcal{B}^\bullet\ B$.

Stating and proving such a result for a Howe's closure built from an early context bisimilarity $\sim$, where inputs and outputs depend on respectively concretions and abstractions, is problematic. Indeed, we would like to prove that $P \sim^\bullet Q$ implies:

- for all $P \xrightarrow{a} F$, for all $C \sim^\bullet C'$, there exists F' such that $Q \xrightarrow{a} F'$ and $F \bullet C \sim^\bullet F' \bullet C'$;
- for all $P \xrightarrow{\overline{a}} C$, for all $F \sim^\bullet F'$ there exists C' such that $Q \xrightarrow{\overline{a}} C'$ and $F \bullet C \sim^\bullet F' \bullet C'$.

These clauses raise several issues. First, we have to find extensions of Howe's closure to abstractions and concretions which fit an early style. Even assuming such extensions, we cannot use this result to show $\sim^\bullet$ is a simulation. Indeed, suppose we are in the higher-order communication case: the processes are a parallel composition ($P = P_1 \mid P_2$, $Q = Q_1 \mid Q_2$, $P_1 \sim^\bullet Q_1$, and $P_2 \sim^\bullet Q_2$) and the transition is a higher-order communication ($P \xrightarrow{\tau} F \bullet C$, $P_1 \xrightarrow{a} F$, and $P_2 \xrightarrow{\overline{a}} C$). We thus need to find F' and C' such that $Q \xrightarrow{\tau} F' \bullet C'$, and $F \bullet C \sim^\bullet F' \bullet C'$. However, we cannot apply the input clause with $P_1 \sim^\bullet Q_1$: to have a F' such that $Q_1 \xrightarrow{a} F'$, we have to find first a concretion C' such that $C \sim^\bullet C'$. We cannot use the output clause with P_2 and Q_2 either: to have a C' such that $Q_2 \xrightarrow{\overline{a}} C'$, we have to find first an abstraction F' such that $F \sim^\bullet F'$. Taking $C \sim^\bullet C$ to obtain F' such that $F \bullet C \sim^\bullet F' \bullet C$, then $F' \sim^\bullet F'$ to yield C' and $F' \bullet C \sim^\bullet F' \bullet C'$ would not work either: to conclude we would need to show that $\sim^\bullet$ is transitive. Transitivity is the reason usual congruence proof techniques fail with weak bisimulations, and the very motivation to turn to Howe's method [11, Section 3.1]. As we cannot bypass this mutual dependency nor this transitivity requirement, the proof fails in the communication case.

In [5], the authors break the mutual dependency by partially dropping the early style: they write the output clause in the late style. The resulting *input-early* bisimilarity is complete in the strong case, but not in the weak case. In [11], we propose to make the output clause a little less early: instead of first requiring the abstraction to provide a matching output, we only require the process that does the reception – that reduces to the abstraction. This small change is sufficient to break the mutual dependency. Indeed, the concretion C' from Q_2 matching the $P_2 \xrightarrow{\overline{a}} C$ step depends only on P_1, which is known, and not on some unknown abstraction. We can then obtain the abstraction F' from Q_2 that matches the $P_1 \xrightarrow{a} F$ step. This abstraction depends fully on C', in the usual early style.

Unfortunately, we do not directly use abstractions and concretions in [11], we define instead a *complementary* LTS, and its bisimilarity. Such a LTS implements the change above as follows: when P sends a message to Q, this becomes a transition from P *using* Q *as a label*. As a result, in the corresponding bisimilarity, an output action depends on a process that performs the input instead of the input itself. The LTS we obtain is serialized compared to the contextual one: in a communication, we do not have two parallel derivation trees for the output and the input, as with rule HO, but a single one, where we first look for the output, and then look for the input. But creating such a complementary LTS can be difficult, especially to handle scope extrusion properly, as we observed with passivation [11]. In the next section, we show that we can in fact apply Howe's method with the regular LTS.

2.4 Congruence Proof Using Howe's Method

As explained in Section 2.3, the main challenge to apply Howe's method is stating and proving a pseudo-simulation lemma for the Howe's closure $\sim^\bullet$. With contextual semantics, the challenge is to avoid mutual dependencies between the input and output clauses. Following the main idea behind the complementary semantics, we propose to keep the usual LTS but change the definition of the pseudo-simulation property to make the output depend on a process performing an input, and not the input itself. Conversely, the input now depends on a process performing an output, and not the output itself. Formally, if $P_1 \sim^\bullet Q_1$, then

- for all $P_1 \xrightarrow{a} F_1$, for all $P_2 \sim^\bullet Q_2$ such that $P_2 \xrightarrow{\overline{a}} C_1$, there exist F_2, C_2, such that $Q_1 \xrightarrow{a} F_2$, $Q_2 \xrightarrow{\overline{a}} C_2$, and $F_1 \bullet C_1 \sim^\bullet F_2 \bullet C_2$;
- for all $P_1 \xrightarrow{\overline{a}} C_1$, for all $P_2 \sim^\bullet Q_2$ such that $P_2 \xrightarrow{a} F_1$, there exist F_2, C_2, such that $Q_1 \xrightarrow{\overline{a}} C_2$, $Q_2 \xrightarrow{a} F_2$, and $F_1 \bullet C_1 \sim^\bullet F_2 \bullet C_2$.

This definition offers two advantages. First, we do not have to define an extension of $\sim^\bullet$ to abstractions and concretions as we relate only processes. Second, the clauses for the input and the output are identical, exchanging only the roles of P_1 and P_2, and of Q_1 and Q_2. Therefore, we can capture the input and output clause as a single symmetric clause. This gives us the up-to $\equiv$ pseudo-simulation lemma we will prove for $\sim_c^\bullet$ (the restriction of $\sim^\bullet$ to closed processes).

▶ **Lemma 8** (Pseudo-Simulation Lemma). *Let $P_1 \sim_c^\bullet Q_1$ and $P_2 \sim_c^\bullet Q_2$. If $P_1 \xrightarrow{\overline{a}} C_1$ and $P_2 \xrightarrow{a} F_1$, then there exist C_2, F_2 such that $Q_1 \xrightarrow{\overline{a}} C_2$, $Q_2 \xrightarrow{a} F_2$, and $F_1 \bullet C_1 \equiv \sim_c^\bullet \equiv F_2 \bullet C_2$.*

With this formulation of the pseudo-simulation lemma, we easily dispatch the communication case. Suppose $P = P_1 \mid P_2$ and $Q = Q_1 \mid Q_2$ with $P_1 \sim_c^\bullet Q_1$ and $P_2 \sim_c^\bullet Q_2$. If $P \xrightarrow{\tau} F \bullet C$, with $P_1 \xrightarrow{a} F_1$ and $P_2 \xrightarrow{\overline{a}} C_1$, then we immediately have F_2, C_2 such that $Q \xrightarrow{\tau} F_2 \bullet C_2$ and $F_1 \bullet C_1 \equiv \sim_c^\bullet \equiv F_2 \bullet C_2$.

Lemma 8 can be proved in several ways, using either serialized inductions, or a simultaneous induction on $P_1 \sim_c^\bullet Q_1$ and $P_2 \sim_c^\bullet Q_2$. We discuss here the former, with proofs detailed in [10, Appendix A]. We then adapt this approach to a calculus with passivation (Section 3). The simultaneous induction approach is presented in Section 4 for a calculus with join patterns.

Using serialized inductions, we can start with $P_1 \sim_c^\bullet Q_1$ or with $P_2 \sim_c^\bullet Q_2$. Suppose we start with an induction on the sending processes $P_1 \sim_c^\bullet Q_1$. Most cases consist in using the induction hypothesis, followed by congruence properties of $\sim_c^\bullet$. There are two exceptions: (1) the base case $P_1 \sim Q_1$, and (2) the case $P_1 = \overline{a}\langle P_1^1 \rangle P_1^2$, $Q_1 = \overline{a}\langle Q_1^1 \rangle Q_1^2$, with $P_1^1 \sim_c^\bullet Q_1^1$ and $P_1^2 \sim_c^\bullet Q_1^2$. In these cases, we know which concretion C_2 the process Q_1 reduces to (either using $\sim$ in case (1), or by construction of P_1 and Q_1 in case (2)), but we have to find the abstraction F_2 the process Q_2 reduces to. To do so, we prove the following.

▶ **Lemma 9.** *Let $P_1^1 \sim_c^\bullet Q_1^1$ and $P_2 \sim_c^\bullet Q_2$ such that $P_2 \xrightarrow{a} F_1$. There exists F_2 such that $Q_2 \xrightarrow{a} F_2$, and $F_1 \circ P_1^1 \sim_c^\bullet F_2 \circ Q_1^1$.*

The proof of this lemma is by induction on the derivation of $P_2 \sim_c^\bullet Q_2$. Lemma 9 deals with case (2) directly (just add the continuations P_1^2 and Q_1^2 using congruence), but it also handles case (1) ($P_1 \sim Q_1$). Indeed, if R is the message of C_1, applying Lemma 9 with $P_1^1 = Q_1^1 = R$ gives $F_1 \circ R \sim_c^\bullet F_2 \circ R$, which implies $F_1 \bullet C_1 \sim_c^\bullet F_2 \bullet C_1$ by congruence of $\sim_c^\bullet$. Since $P_1 \sim Q_1$, there exists C_2 such that $Q_1 \xrightarrow{\overline{a}} C_2$, and $F_2 \bullet C_1 \sim F_2 \bullet C_2$. We therefore have $F_1 \bullet C_1 \sim_c^\bullet \sim F_2 \bullet C_2$, which implies $F_1 \bullet C_1 \sim_c^\bullet F_2 \bullet C_2$ by right transitivity with $\sim$.

Alternatively, we can prove Lemma 8 by starting with the induction on the receiving processes $P_2 \sim_c^\bullet Q_2$. To handle the two cases (3) $P_2 \sim Q_2$ and (4) $P_2 = a(X)P$, $Q_2 = a(X)Q$, $P \sim^\bullet Q$, we need the following result.

▶ **Lemma 10.** *Let $P \sim^\bullet Q$ such that $fv(P) \cup fv(Q) \subseteq \{X\}$, and $P_1 \sim_c^\bullet Q_1$ such that $P_1 \xrightarrow{\overline{a}} C_1$. There exists C_2 such that $Q_1 \xrightarrow{\overline{a}} C_2$ and $(X)P \bullet C_1 \equiv \sim_c^\bullet \equiv (X)Q \bullet C_2$.*

▶ Remark. Lemmas 8 and 10 are defined up to $\equiv$ while Lemma 9 is not. Structural congruence is needed to move name restriction: suppose we have $P_1 \sim_c^\bullet Q_1$, $\nu b.P_2 \sim_c^\bullet \nu b.Q_2$, with $P_2 \sim_c^\bullet Q_2$, $P_1 \xrightarrow{a} F_1$, and $\nu b.P_2 \xrightarrow{\overline{a}} \nu b.C_2$ (which comes from $P_2 \xrightarrow{\overline{a}} C_2$). Using the induction with P_1, Q_1, P_2, and Q_2, there exist F_2 and C_2 such that $Q_1 \xrightarrow{a} F_2$, $Q_2 \xrightarrow{\overline{a}} C_2$, and $F_1 \bullet C_1 \sim_c^\bullet F_2 \bullet C_2$. We also have $\nu b.Q_2 \xrightarrow{\overline{a}} \nu b.C_2$. Note that, by our convention on bound names, b is neither in F_1 nor in F_2.

We want to prove $F_1 \bullet (\nu b.C_1) \sim_c^\bullet F_2 \bullet (\nu b.C_2)$, but from $F_1 \bullet C_1 \sim_c^\bullet F_2 \bullet C_2$, we can deduce $\nu b.(F_1 \bullet C_1) \sim_c^\bullet \nu b.(F_2 \bullet C_2)$ by congruence of $\sim_c^\bullet$. Depending on whether the scope of b has to be extended or not, it is not the same as $F_1 \bullet (\nu b.C_1) \sim_c^\bullet F_2 \bullet (\nu b.C_2)$; at best, we have $F_1 \bullet (\nu b.C_1) \equiv \nu b.(F_1 \bullet C_1) \sim_c^\bullet \nu b.(F_2 \bullet C_2) \equiv F_2 \bullet (\nu b.C_2)$, hence the need for $\equiv$. We do not have this issue in Lemma 9, since only messages, and not concretions, are involved.

For $\sim_c^\bullet$ to be a simulation, we have to prove the following result on τ-actions (by induction on the derivation of $P \sim_c^\bullet Q$), using Lemma 8 in the communication case.

▶ **Lemma 11.** *If $P \sim_c^\bullet Q$ and $P \xrightarrow{\tau} P'$, then there exists Q' such that $Q \xrightarrow{\tau} Q'$ and $P' \equiv \sim_c^\bullet \equiv Q'$.*

We can then prove $\sim_c^\bullet$ is a simulation up to $\equiv$. Suppose $P \sim_c^\bullet Q$. If $P \xrightarrow{a} F$, then for all $C = \nu\widetilde{b}.\langle R\rangle S$, we apply Lemma 8 with $P_2 = P$, $Q_2 = Q$, and $P_1 = Q_1 = \nu\widetilde{b}.\overline{a}\langle R\rangle S$. This yields an F' such that $Q \xrightarrow{a} F'$ and $F \bullet C \equiv \sim_c^\bullet \equiv F' \bullet C$. Similarly, if $P \xrightarrow{\overline{a}} C$, then for all $F = (X)R$, we apply Lemma 8 with $P_1 = P$, $Q_1 = Q$, and $P_2 = Q_2 = a(X)R$. We can then deduce that $(\equiv \sim_c^\bullet \equiv)^*$ is a bisimulation, and finally conclude $\sim = \equiv \sim_c^\bullet \equiv$, as explained in Section 2.3. Since $\equiv \sim_c^\bullet \equiv$ is a congruence, then $\sim$ is a congruence.

3 Application to a Calculus with Passivation

3.1 The HOπP Calculus

HOπP [11] extends HOπ with passivation, an operation that may stop a running process and capture its state. The granularity of passivation is the *locality* $a[P]$, a new construct added to the syntax of HOπ. The semantics of $a[P]$ is as follows: P can freely reduce and communicate with any other process; it may also be captured at any time by a process $a(X)R$, substituting its contents P for X in R. Formally, we extend the locality construct to all agents, and we add the rules Loc and Passiv to the LTS of Figure 1.

$$a[(X)P] \triangleq (X)a[P] \qquad a[\nu\widetilde{b}.\langle P\rangle Q] \triangleq \nu\widetilde{b}.\langle P\rangle a[Q] \text{ if } a \notin \widetilde{b}$$

$$a[P] \xrightarrow{\overline{a}} \langle P\rangle 0 \ \textsc{Passiv} \qquad \frac{P \xrightarrow{\alpha} A}{a[P] \xrightarrow{\alpha} a[A]} \ \textsc{Loc}$$

The rule Loc and the definition of $a[C]$ imply that the scope of restricted names may cross locality boundaries, but structural congruence is left unchanged. In particular, $\nu b.a[P]$ is not congruent to $a[\nu b.P]$. Indeed, the combination of lazy scope extrusion and passivation may generate two distinct behaviors from these terms. See [11, Section 2.3] for more details.

3.2 Context Bisimilarity

The definition of context bisimulation is more complex in HOπP than in HOπ because of the discriminating power added by passivation. We briefly explain the differences; more details and examples can be found in [11, Section 2.4]. First, we can distinguish between processes with different free names using passivation and lazy scope extrusion [2]. Indeed, suppose a is free in P but not in Q, and consider the context $b[\nu a.\overline{c}\langle\Box\rangle R]$. Then a communication on c extends the scope of a outside b for P but not for Q, which gives us processes of the form $\nu a.(b[R] \mid P')$ and $b[\nu a.R] \mid Q'$ for some P' and Q'. If we then capture the locality b and duplicate its content, we obtain $\nu a.(R \mid R \mid P')$ in one case, and $(\nu a.R) \mid (\nu a.R) \mid Q'$ in the other: for the first process, a is shared, but not for the second one, and by choosing R accordingly, we obtain different behavior. Therefore, two processes P and Q are equivalent only if $\mathrm{fn}(P) = \mathrm{fn}(Q)$.

Next, when a message is sent outside a locality, the continuation stays in the locality (by definition of $a[C]$). The continuation can then be put into a completely different context using passivation. As a result, the message and its continuation may end up in different contexts, but still share a common information (the extruded names). To be able to express this situation specific to calculi with passivation, we introduce *bisimulation contexts* $\mathbb{E}$, i.e., evaluation contexts used for observational purposes.

$$\mathbb{E} ::= \Box \mid \nu a.\mathbb{E} \mid \mathbb{E} \mid P \mid P \mid \mathbb{E} \mid a[\mathbb{E}]$$

Instead of comparing $F \bullet C$ with $F \bullet C'$ in the output case, we now compare $F \bullet \mathbb{E}\{C\}$ with $F \bullet \mathbb{E}\{C'\}$. The extra context $\mathbb{E}$ represents the potential passivation of the continuations of C and C'. The definition of context bisimulation for HOπP is then as follows.

▶ **Definition 12.** A relation $\mathcal{R}$ on closed processes is a context simulation if $P \mathcal{R} Q$ implies $\mathrm{fn}(P) = \mathrm{fn}(Q)$ and:

- for all $P \xrightarrow{\tau} P'$, there exists Q' such that $Q \xrightarrow{\tau} Q'$ and $P' \mathcal{R} Q'$;
- for all $P \xrightarrow{a} F$, for all C, there exists F' such that $Q \xrightarrow{a} F'$ and $F \bullet C \mathcal{R} F' \bullet C$;
- for all $P \xrightarrow{\overline{a}} C$, for all F, $\mathbb{E}$, there exists C' such that $Q \xrightarrow{\overline{a}} C'$ and $F \bullet \mathbb{E}\{C\} \mathcal{R} F \bullet \mathbb{E}\{C'\}$.

A relation $\mathcal{R}$ is a context bisimulation if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are context simulations. Context bisimilarity, written $\sim$, is the largest context bisimulation.

The usual approach to prove soundness of $\sim$ consists in proving that its transitive and congruence closure is a context bisimulation. This proof technique does not carry to the weak case. In [11], we prove soundness of a weak complementary bisimilarity, which coincides with a weak variant of $\sim$, by defining a weak complementary LTS for HOπP, with elaborate labels and subtle side-conditions in the LTS rules to handle lazy scope extrusion. The resulting LTS has almost twice as many rules as the contextual one.

We show here how to directly apply Howe's method with the contextual semantics, as in HOπ. We give these results for the strong bisimilarity $\sim$ to ease the presentation; the proofs for the weak case are in [10, Appendix B]. As usual when adapting Howe's method to calculi with passivation [5, 11], we have to extend Howe's closure to bisimulation contexts. We define $\mathbb{E}_1 \sim^\bullet \mathbb{E}_2$ as the smallest congruence satisfying the following rules.

$$\frac{\mathbb{E}_1 \sim^\bullet \mathbb{E}_2 \qquad P_1 \sim^\bullet P_2}{\mathbb{E}_1 \mid P_1 \sim^\bullet \mathbb{E}_2 \mid P_2} \qquad\qquad \frac{P_1 \sim^\bullet P_2 \qquad \mathbb{E}_1 \sim^\bullet \mathbb{E}_2}{P_1 \mid \mathbb{E}_1 \sim^\bullet P_2 \mid \mathbb{E}_2}$$

We can then write a pseudo-simulation lemma similar to Lemma 8, as follows.

▶ **Lemma 13** (Pseudo-Simulation Lemma). *Let* $P_1 \sim_c^\bullet Q_1$ *and* $P_2 \sim_c^\bullet Q_2$. *If* $P_1 \xrightarrow{\overline{a}} C_1$ *and* $P_2 \xrightarrow{a} F_1$, *then for all* $\mathbb{E}_1 \sim_c^\bullet \mathbb{E}_2$, *there exist* C_2, F_2 *such that* $Q_1 \xrightarrow{\overline{a}} C_2$, $P_2 \xrightarrow{a} F_2$, *and* $F_1 \bullet \mathbb{E}_1\{C_1\} \sim_c^\bullet F_2 \bullet \mathbb{E}_2\{C_2\}$.

Unlike the case with HOπ, we do not have a choice in the induction strategy for the proof of Lemma 13: we cannot prove it by doing first the induction on the derivation for the receiving processes $P_2 \sim_c^\bullet Q_2$. Indeed, suppose $F_1 \bullet \mathbb{E}_1\{C_1\} \sim_c^\bullet F_2 \bullet \mathbb{E}_2\{C_2\}$ holds for all $\mathbb{E}_1 \sim_c^\bullet \mathbb{E}_2$, and we want to prove $b[F_1] \bullet \mathbb{E}_1\{C_1\} \sim_c^\bullet b[F_2] \bullet \mathbb{E}_2\{C_2\}$. With congruence of $\sim_c^\bullet$, we can only deduce $b[F_1 \bullet \mathbb{E}_1\{C_1\}] \sim_c^\bullet b[F_2 \bullet \mathbb{E}_2\{C_2\}]$, and we cannot move the boundaries of b with $\equiv$. Therefore, when reasoning by induction on the receiving processes $P_2 \sim_c^\bullet Q_2$, we cannot apply the resulting abstractions F_1, F_2 to concretions. However, we can apply them to messages, as in the following lemma, identical to Lemma 9.

▶ **Lemma 14.** *Let* $P_1^1 \sim_c^\bullet Q_1^1$ *and* $P_2 \sim_c^\bullet Q_2$ *such that* $P_2 \xrightarrow{a} F_1$. *There exists* F_2 *such that* $Q_2 \xrightarrow{a} F_2$, *and* $F_1 \circ P_1^1 \sim_c^\bullet F_2 \circ Q_1^1$.

Indeed, if $F_1 \circ P_1^1 \sim_c^\bullet F_2 \circ Q_1^1$, then $b[F_1 \circ P_1^1] \sim_c^\bullet b[F_2 \circ Q_1^1]$ by congruence of $\sim_c^\bullet$. We then prove Lemma 13 by induction on the derivation for the sending processes $P_1 \sim_c^\bullet Q_1$. We do not have problems with localities when doing the induction on the derivation of $P_1 \sim_c^\bullet Q_1$, thanks to the bisimulation contexts: if $F_1 \bullet \mathbb{E}_1\{C_1\} \sim_c^\bullet F_2 \bullet \mathbb{E}_2\{C_2\}$ holds for all $\mathbb{E}_1 \sim_c^\bullet \mathbb{E}_2$, then it also holds for $\mathbb{E}_1\{b[\Box]\} \sim_c^\bullet \mathbb{E}_2\{b[\Box]\}$, and we have $F_1 \bullet \mathbb{E}_1\{b[C_1]\} \sim_c^\bullet F_2 \bullet \mathbb{E}_2\{b[C_2]\}$, as wished. Note that it also implies $F_1 \bullet \mathbb{E}_1\{\nu b.C_1\} \sim_c^\bullet F_2 \bullet \mathbb{E}_2\{\nu b.C_2\}$ by taking $\mathbb{E}_1\{\nu b.\Box\} \sim_c^\bullet \mathbb{E}_2\{\nu b.\Box\}$, therefore restriction poses no problem, and Lemma 13 is formulated without structural congruence, unlike Lemma 8. In addition to Lemma 13, we also prove a lemma similar to Lemma 11 for τ-actions, and then deduce that $\sim_c^\bullet$ is a simulation. We conclude as for HOπ.

Completeness. The strong and weak variants of the context bisimilarity $\sim$ coincide with respectively the strong and weak complementary bisimilarities of [11], which are themselves complete (see [11, Section 5.2]). Consequently, the strong and weak context bisimilarities are also complete.

4 Application to a Calculus with Join Patterns

4.1 Syntax and Semantics

Join patterns allow several messages to be received at once by the same process. The syntax of HOπJ is given in Figure 2. We replace the receiving process $a(X)P$ of HOπ by a process $\pi \rhd P$, where π is a join pattern $a_1(X_1) | \ldots | a_n(X_n)$. A higher-order communication takes place when messages are available simultaneously on the names $a_1 \ldots a_n$. We write $\prod_{i\in\{1..n\}} x_i$ or $\prod \widetilde{x}$ (where x ranges over some entity) for the parallel composition $x_1 | \ldots | x_n$ if $n > 1$, or for simply x_1 if $n = 1$. We also abbreviate $\pi = a_1(X_1) \mid \ldots \mid a_n(X_n)$ as $\prod \widetilde{a(X)}$. The syntax of abstractions is changed accordingly ($F \triangleq (\pi)P$), and concretions now accumulate the messages of several emitting processes in parallel. A concretion is of the form $\nu\widetilde{b}.\langle a_1, P_1\rangle \ldots \langle a_n, P_n\rangle Q$, meaning that each process P_i is sent on the name a_i, and the scope of the names $\widetilde{b}$ has to be extended to encompass the recipient of the messages. We abbreviate $\nu\widetilde{b}.\langle a_1, P_1\rangle \ldots \langle a_n, P_n\rangle Q$ as $\nu\widetilde{b}.\langle \widetilde{a, P}\rangle Q$.

The semantics of HOπJ is given by the LTS rules of Figure 2, where the symmetric of rules Par, HO, and Part-HO are omitted. An input $P \xrightarrow{\widetilde{a}} F$ is labelled with the multiset $\widetilde{a}$

Syntax: $P ::= \mathbf{0} \mid X \mid P \mid P \mid \nu a.P \mid \overline{a}\langle P\rangle P \mid \pi \triangleright P \qquad \pi ::= \pi \mid \pi \mid a(X)$

Agents: $F ::= (\pi)P \qquad C ::= D \mid \nu a.D \qquad D ::= \langle a, P\rangle Q \mid \langle a, P\rangle D$

Parallel composition of concretions

$$\nu\widetilde{b}.\langle\widetilde{a, R}\rangle P \mid \nu\widetilde{b'}.\langle\widetilde{a', R'}\rangle Q \triangleq \nu\widetilde{b} \cup \widetilde{b'}.\langle\widetilde{a, R} \uplus \widetilde{a', R'}\rangle(P \mid Q)$$
$$\text{if } \widetilde{b} \cap \mathrm{fn}(Q) = \widetilde{b'} \cap \mathrm{fn}(P) = \widetilde{b} \cap \widetilde{b'} = \emptyset$$

Structural congruence for join patterns

$$\pi_1 \mid \pi_2 \equiv \pi_2 \mid \pi_1 \qquad \pi_1 \mid (\pi_2 \mid \pi_3) \equiv (\pi_1 \mid \pi_2) \mid \pi_3$$

Pseudo-application

$$(\prod \widetilde{a(X)})P \bullet \nu\widetilde{b}.\langle\widetilde{a, R}\rangle Q \vdash \nu\widetilde{b}.(P\{\widetilde{R}/\widetilde{X}\} \mid Q) \text{ if } \widetilde{b} \cap \mathrm{fn}(P) = \emptyset$$
$$(\prod \widetilde{a(X)} \mid \pi)P \bullet \nu\widetilde{b}.\langle\widetilde{a, R}\rangle Q \vdash (\pi)\nu\widetilde{b}.(P\{\widetilde{R}/\widetilde{X}\} \mid Q) \text{ if } \widetilde{b} \cap \mathrm{fn}(P) = \emptyset$$

LTS rules: $\alpha_{\mathrm{j}} ::= \tau \mid \widetilde{a} \mid \widetilde{\overline{a}}$

$$\pi \triangleright P \xrightarrow{\widetilde{a}} (\pi)P \ \textsc{In} \qquad \overline{a}\langle Q\rangle P \xrightarrow{\overline{a}} \langle a, Q\rangle P \ \textsc{Out} \qquad \frac{P \xrightarrow{\alpha_{\mathrm{j}}} A}{P \mid Q \xrightarrow{\alpha_{\mathrm{j}}} A \mid Q} \ \textsc{Par}$$

$$\frac{P \xrightarrow{\widetilde{\overline{a}}} C_1 \qquad Q \xrightarrow{\widetilde{\overline{b}}} C_2}{P \mid Q \xrightarrow{\widetilde{\overline{a}} \uplus \widetilde{\overline{b}}} C_1 \mid C_2} \ \textsc{Par-Out} \qquad \frac{P \xrightarrow{\widetilde{a}} F \qquad Q \xrightarrow{\widetilde{\overline{a}}} C \qquad F \bullet C \vdash P'}{P \mid Q \xrightarrow{\tau} P'} \ \textsc{HO}$$

$$\frac{P \xrightarrow{\alpha_{\mathrm{j}}} A \qquad a \notin \alpha_{\mathrm{j}}}{\nu a.P \xrightarrow{\alpha_{\mathrm{j}}} \nu a.A} \ \textsc{Restr} \qquad \frac{P \xrightarrow{\widetilde{a} \uplus \widetilde{b}} F \qquad Q \xrightarrow{\widetilde{\overline{b}}} C \qquad \widetilde{a} \neq \emptyset \qquad F \bullet C \vdash F'}{P \mid Q \xrightarrow{\widetilde{a}} F'} \ \textsc{Part-HO}$$

Figure 2 Syntax and operational semantics of HOπJ.

of names on which messages are expected, and an output $P \xrightarrow{\widetilde{\overline{a}}} C$ is labelled by the multiset $\widetilde{\overline{a}}$ of conames on which messages are sent. Operators are extended to all agents as in HOπ, with the addition of parallel composition of concretions, to deal with the case where two processes P and Q in parallel reduce to C_1 and C_2. The parallel composition of C_1 and C_2 is defined as a concretion C which merges the messages and extruded names of C_1 and C_2, and composes in parallel their continuations (Figure 2, rule Par-Out).

A process P, receiving on names $\widetilde{a}$ (i.e., such that $P \xrightarrow{\widetilde{a}} (\pi)P'$), may communicate with a process Q emitting on names $\widetilde{b}$ (i.e., such that $Q \xrightarrow{\widetilde{\overline{b}}} C$) if $\widetilde{b} \subseteq \widetilde{a}$. We have two possible outcomes: either $\widetilde{b} = \widetilde{a}$ and the resulting agent is a process (rule HO), or $\widetilde{b} \subsetneq \widetilde{a}$ – some inputs of the join patterns are not filled with Q – and we obtain an abstraction (rule Part-HO). For instance, we have $\overline{a}\langle R\rangle\mathbf{0} \mid (a(X) \mid b(Y)) \triangleright P \xrightarrow{b} (b(Y))P\{R/X\}$. The definition of $\bullet$ in Figure 2 takes into account these two cases. Besides, the pseudo-application of an abstraction to a concretion may generate several results, depending on how the matching between the

outputs and the input is done. For instance, $\overline{a}\langle R_1\rangle\mathbf{0}\,|\,\overline{a}\langle R_2\rangle\mathbf{0}\,|\,(a(X)\,|\,a(Y))\triangleright P$ can reduce to either $P\{R_1/X\}\{R_2/Y\}$, or $P\{R_2/X\}\{R_1/Y\}$ (assuming R_1 and R_2 closed). Consequently, we write $\bullet$ as a predicate $F \bullet C \vdash P$ (respectively $F \bullet C \vdash F'$), meaning that P (respectively F') can be obtained as a result of the pseudo-application of F to C.

4.2 Context Bisimilarity

The definition of context bisimilarity for HOπJ is the same as for HOπ, adapted to the fact that $\bullet$ may generate several results for a given F and C.

▶ **Definition 15.** A relation $\mathcal{R}$ on closed processes is a context simulation if $P\ \mathcal{R}\ Q$ implies:

- for all $P \xrightarrow{\tau} P'$, there exists Q' such that $Q \xrightarrow{\tau} Q'$ and $P'\ \mathcal{R}\ Q'$;
- for all $P \xrightarrow{\widetilde{a}} F$, for all C, for all P' such that $F \bullet C \vdash P'$, there exist F', Q' such that $Q \xrightarrow{\widetilde{a}} F'$, $F' \bullet C \vdash Q'$, and $P'\ \mathcal{R}\ Q'$;
- for all $P \xrightarrow{\widetilde{\widetilde{a}}} C$, for all F, for all P' such that $F \bullet C \vdash P'$, there exist C', Q' such that $Q \xrightarrow{\widetilde{\widetilde{a}}} C'$, $F \bullet C' \vdash Q'$, and $P'\ \mathcal{R}\ Q'$.

A relation $\mathcal{R}$ is a context bisimulation if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are context simulations. Context bisimilarity, written $\sim$, is the largest context bisimulation.

A similar context bisimulation has been defined for Kell [17], a higher-order calculus with passivation and join patterns. It is sound and complete in the strong case; the soundness proof of [17] does not rely on Howe's method, but instead shows that the reflexive, transitive, and congruence closure of the bisimilarity is itself a bisimulation. This direct method unfortunately does not scale to the weak case, as explained in [11]. Here, we prove that $\sim$ is a congruence using Howe's method. As in the previous section, even though we present the results in the strong case for simplicity, the complete proofs in [10, Appendix C] are for the weak case. To our knowledge, it is the first proof of soundness of a weak bisimilarity for a higher-order calculus with join patterns.

Bisimulation up to $\equiv$ is defined as in HOπ, by replacing $\mathcal{R}$ by $\equiv\mathcal{R}\equiv$ in the clauses. To prove that $\sim$ is sound with Howe's method, we use the following pseudo-simulation lemma.

▶ **Lemma 16** (Pseudo-Simulation Lemma). *Let $P \sim_c^\bullet Q$ and $\widetilde{R} \sim_c^\bullet \widetilde{R'}$ such that $P \xrightarrow{\widetilde{a}} F$, $R_i \xrightarrow{\widetilde{\widetilde{a}}_i} C_i$ for all i, $\widetilde{a} = \biguplus_i \widetilde{\widetilde{a}}_i$, and let P' such that $F \bullet \prod_i C_i \vdash P'$. Then there exist F', $\widetilde{C'}$, and Q' such that we have $Q \xrightarrow{\widetilde{a}} F'$, $R_i' \xrightarrow{\widetilde{\widetilde{a}}_i} C_i'$ for all i, $F' \bullet \prod_i C_i' \vdash Q'$, and $P' \equiv\sim_c^\bullet\equiv Q'$.*

We extend relations to multisets of same size in a pointwise way: $\widetilde{R} \sim_c^\bullet \widetilde{R'}$ means the two multisets are of the same size, and $R_i \sim_c^\bullet R_i'$ holds for every i. Note that Lemma 16 is a direct extension of Lemma 8 to multisets of sending processes; indeed, if we replace $\widetilde{R}$ and $\widetilde{R'}$ with single processes, we obtain the same formulation as Lemma 8 (with the exception that $\bullet$ is a predicate).

The proofs by serialization of Lemma 8, where we proceed by induction on the derivations for the the sender and then on the receiver (or conversely), do not apply to a calculus with join patterns, where a receiver communicates with several emitters – we cannot focus on a sender in particular, we have to consider them together. As a result, we consider another proof method, where we reason by induction on the derivations of $P \sim_c^\bullet Q$ and all the $\widetilde{R} \sim_c^\bullet \widetilde{R'}$ simultaneously. We distinguish two kinds of cases, depending on whether we need the induction hypothesis (detailed proofs are in [10, Appendix C]). Using the same definitions as in Lemmas 8 and 9, the cases where we do not need induction are those where each

$R_i \sim_c^\bullet R'_i$ verifies either (1) or (2) (bisimilar, or congruent outputs), and $P \sim_c^\bullet Q$ verifies either (3) or (4) (bisimilar, or congruent inputs). In these cases, we can conclude using substitutivity of $\sim_c^\bullet$ and the definition of $\sim$. The remaining cases are dealt with by using the induction hypothesis, and then congruence of $\sim_c^\bullet$ and $\equiv$. Again, we rely on structural congruence to change the scope of names when needed (we have the same issue as described in Remark 2.4).

Using Lemma 16, we can prove that $\sim_c^\bullet$ is a simulation up to $\equiv$, and then conclude that $\equiv\sim_c^\bullet\equiv = \sim$ as in HOπ.

Completeness. In [10, Appendix D], we prove that a weak variant of $\sim$ is complete, using the usual technique of [16]. We can prove completeness in the strong case with a similar proof.

▶ Remark. Proving Lemma 8 in HOπ is possible by reasoning simultaneously on $P_1 \sim_c^\bullet Q_1$ and $P_2 \sim_c^\bullet Q_2$, as described above. However, this method does not work for HOπP (Lemma 13) as pseudo-application and locality contexts do not commute (even up to structural congruence). One way to make the simultaneous induction works in calculi with passivation would be to add bisimulation contexts in the input clause, as follows:

- for all $P \xrightarrow{a} F$, for all C, there exists F' such that $Q \xrightarrow{a} F'$ and for all $\mathbb{E}$, we have $\mathbb{E}\{F\} \bullet C \mathrel{\mathcal{R}} \mathbb{E}\{F'\} \bullet C$.

With such a definition, we can prove soundness of the resulting bisimilarity in a calculus with passivation and join patterns (such as Kell) with the simultaneous induction. However, this extra use of bisimulation context adds complexity to the bisimulation. We conjecture they are not necessary in the input case.

5 Related Work

Howe's method in process calculi. Howe's method has been originally used to prove congruence in a lazy functional programming language [7]. Baldamus and Frauenstein [1] are the first to adapt the method to process calculi for variants of Plain CHOCS [18], and prove in particular the soundness of a weak late delay context bisimilarity. Hildebrandt and Godskesen [5] then adapt Howe's method for their calculus Homer, to prove the congruence of a (delay) input-early context bisimilarity (see Section 2.3). In [11], we use Howe's method to prove congruence of strong and weak complementary bisimilarities in HOπ and HOπP. The Howe's proof of [11] is somewhat similar to the serialized proof of Sections 2 and 3, except for the symmetric formulation of the pseudo-simulation lemma. However, there is no equivalent to the simultaneous induction proof of Section 4 in [11].

Bisimilarities in calculi with passivation. In addition to the context (or complementary) bisimilarities already discussed for Kell [17], Homer [5], and HOπP [11], *environmental bisimilarities* [15] have also been defined by Piérard and Sumii for calculi with passivation [12, 13]. Such relations compare P and Q using an environment $\mathcal{E}$, which represents the knowledge that an observer has about these processes, like the messages they have sent. The observer then uses $\mathcal{E}$ to challenge P and Q. For instance, the observer is able to compare inputs from P and Q with any messages built from the processes inside $\mathcal{E}$. In [12], the authors propose a sound weak environmental bisimilarity for HOπP. Their approach is not complete, seemingly because of the interplay between "by need" scope extrusion and passivation. In [13], they consider a variant of HOπP with name creation instead of name restriction, for which they define a sound and complete weak environmental bisimilarity. With name

creation, a name generated in a given locality becomes automatically known from the whole system. Name creation is therefore less expressive than name restriction with lazy scope extrusion, where we can control more finely the scope of generated names. In particular, it is not possible to implement internal choice or recursion using name creation, as shown in [8]. Finally, Koutavas and Hennessy recently developed a correct and complete symbolic bisimulation for a higher-order process calculus with passivation [8]. Their approach avoids the quantification over contexts at the cost of a more complex calculus, with local ports to recover the expressivity lost by using name creation.

Bisimilarities in calculi with join patterns. In [4], Fournet and Laneve define bisimilarities for the Join-Calculus, a first-order process calculus with join patterns. They define a weak bisimilarity which is sound w.r.t. the weak barbed congruence defined in [3], and also complete if name matching is added to the calculus. To our knowledge, only Kell [17] combines higher-order communication with join patterns. In [9], we define a weak complementary bisimilarity for Kell, which tests inputs by passing them messages one by one. This strategy requires processes to choose which input to perform without having all the necessary information (i.e., all the messages they are going to receive), and the resulting bisimilarity is therefore too discriminating (i.e., not complete).

6 Conclusion

In this paper, we showed how to directly use Howe's method to prove congruence properties of a context bisimilarity, without relying on an auxiliary relation such as complementary bisimilarity. We proposed a symmetric formulation of the pseudo-simulation lemma, which we can prove either with a serialized or with a simultaneous induction on the derivations for the emitting and receiving processes. The latter seems necessary in calculi with join patterns, while the former seems more appropriate for calculi with passivation. The resulting soundness proofs are much simpler than in complementary semantics [11], and they scale better to calculi with join patterns. Indeed, we compare receiving patterns by passing them several messages at once, and not only one by one as in the complementary case [9]. Finally, the bisimilarities of this paper are also complete in the weak case, unlike the input-early bisimilarity of [5], or the bisimilarity of [9] for join patterns. The use of Howe's method remains an open problem for calculi with both passivation and join patterns, such as Kell, if we do not want to make the definition of the bisimilarity more complex by using bisimulation contexts in the input case (see the remark at the end of Section 4).

References

1 Michael Baldamus and Thomas Frauenstein. Congruence proofs for weak bisimulation equivalences on higher–order process calculi. Technical report, Berlin University of Technology, 1995.

2 Giuseppe Castagna, Jan Vitek, and Francesco Zappa Nardelli. The Seal Calculus. *Information and Computation*, 201(1):1–54, 2005.

3 Cédric Fournet and Georges Gonthier. The reflexive cham and the join-calculus. In *POPL'96*, pages 372–385. ACM Press, 1996.

4 Cédric Fournet and Cosimo Laneve. Bisimulations in the join-calculus. *Theorectical Computer Science*, 266(1-2):569–603, 2001.

5 Jens C. Godskesen and Thomas Hildebrandt. Extending howe's method to early bisimulations for typed mobile embedded resources with local names. In *FSTTCS'05*, volume 3821 of *LNCS*, pages 140–151. Springer, 2005.

6 Andrew D. Gordon. Bisimilarity as a theory of functional programming. *Electronic Notes in Theoretical Computer Science*, 1:232–252, 1995.

7 Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.

8 Vasileios Koutavas and Matthew Hennessy. Symbolic bisimulation for a higher-order distributed language with passivation. In *CONCUR'13*, pages 167–181. Springer-Verlag, 2013.

9 Sergueï Lenglet. *Bisimulations dans les calculs avec passivation*. PhD thesis, Université de Grenoble, 2010.

10 Sergueï Lenglet and Alan Schmitt. Howe's method for contextual semantics. Technical Report RR-8750, Inria, 2015.

11 Sergueï Lenglet, Alan Schmitt, and Jean-Bernard Stefani. Characterizing contextual equivalence in calculi with passivation. *Information and Computation*, 209(11):1390–1433, 2011.

12 Adrien Piérard and Eijiro Sumii. Sound bisimulations for higher-order distributed process calculus. In *FOSSACS'11*, volume 6604 of *LNCS*, pages 123–137. Springer, 2011.

13 Adrien Piérard and Eijiro Sumii. A higher-order distributed calculus with name creation. In *LICS'12*, pages 531–540. IEEE, 2012.

14 Davide Sangiorgi. Bisimulation for higher-order process calculi. *Information and Computation*, 131(2):141–178, 1996.

15 Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. *ACM Transactions on Programming Languages and Systems*, 33(1), 2011.

16 Davide Sangiorgi and David Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

17 Alan Schmitt and Jean-Bernard Stefani. The Kell Calculus: A Family of Higher-Order Distributed Process Calculi. In *Global Computing 2004 workshop*, volume 3267 of *LNCS*, 2004.

18 Bent Thomsen. Plain chocs: A second generation calculus for higher order processes. *Acta Informatica*, 30(1):1–59, 1993.

Forward and Backward Bisimulations for Chemical Reaction Networks

Luca Cardelli[1], Mirco Tribastone[2], Max Tschaikowski[3], and Andrea Vandin[4]

1 **Microsoft Research & University of Oxford, UK**
luca@microsoft.com

2-4 **IMT Institute for Advanced Studies Lucca, Italy**
{mirco.tribastone,max.tschaikowski,andrea.vandin}@imtlucca.it

Abstract

We present two quantitative behavioral equivalences over species of a chemical reaction network (CRN) with semantics based on ordinary differential equations. *Forward CRN bisimulation* identifies a partition where each equivalence class represents the exact sum of the concentrations of the species belonging to that class. *Backward CRN bisimulation* relates species that have identical solutions at all time points when starting from the same initial conditions. Both notions can be checked using only CRN syntactical information, i.e., by inspection of the set of reactions. We provide a unified algorithm that computes the coarsest refinement up to our bisimulations in polynomial time. Further, we give algorithms to compute quotient CRNs induced by a bisimulation. As an application, we find significant reductions in a number of models of biological processes from the literature. In two cases we allow the analysis of benchmark models which would be otherwise intractable due to their memory requirements.

1998 ACM Subject Classification D.2.4 Software/Program Verification, G.1.7 Ordinary Differential Equations, J.2 Physical Sciences and Engineering

Keywords and phrases Chemical reaction networks, ordinary differential equations, bisimulation, partition refinement

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.226

1 Introduction

At the interface between computer science and systems biology is the idea that biological systems can be interpreted as computational processes [23, 12], leading to a number of formal methods applied to study biomolecular systems [5, 18, 25]. In this context, chemical reaction networks (CRNs), a popular mathematical model of interaction in natural sciences, can also be seen as a kernel concurrent language for natural programming.

In this paper we present, for the first time to our knowledge, quantitative bisimulation equivalences for CRNs with the well-known interpretation based on ordinary differential equations (ODEs). (To make the paper self-contained, all background is given in Section 2.) In this semantics, each species is associated with an ODE giving the deterministic evolution of its concentration starting from an initial condition. Our bisimulations are equivalences over species that induce a reduced CRN that exactly preserves the dynamics of the original one. This is an important goal, especially in order to cope with the potentially very large number of species and reactions in many biological networks [16, 17].

We study two equivalences, developed in the Larsen-Skou style of probabilistic bisimulation [29], that are based on two distinct ideas of observable behavior. *Forward CRN*

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 226–239

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

bisimulation yields an aggregated ODE where the solution gives the exact sum of the concentrations of the species belonging to each equivalence class. In *backward CRN bisimulation*, instead, equivalent species have the same solution *at all time points*; in other words, backward CRN bisimulation relates species whose ODE solutions are equal whenever they start from identical initial conditions. The use of "forward" and "backward" has a long tradition in models of computation based on labelled transition systems [19]. In the case of quantitative variants, for instance those defined for process algebra with a continuous-time Markov chain (CTMC) semantics [26, 27, 8, 4], forward bisimulations are equivalences that induce a CTMC aggregation in the sense of ordinary lumpability [7], where the probability of an equivalence class is equal to the sum of the probabilities of the states belonging to that class. This is found by checking conditions on the *outgoing transitions* of related states in the transition diagram. A backward bisimulation induces a CTMC aggregation in the sense of weak lumpability [21], where all states in the same equivalence class have a time-invariant conditional probability distribution; exact lumpability is a special case where the conditional probability distribution is uniform, in the sense that any two states of each equivalence class have the same probability at any point in time whenever they have the same initial probabilities. It is found by relating states according to conditions on their *predecessor states* [21, 34, 7].

Despite being similar in spirit, technically our bisimulations are fundamentally different for two reasons. First, they concern a continuous-state semantics based on ODEs instead of a discrete-state CTMC. Second, they operate at the structural, syntactical level, because they are defined with quantities that can be computed by only inspecting the reactions of a CRN. Still, the repercussions of our bisimulations on the semantics are explained according to certain theories of aggregation. In particular, forward CRN bisimulation yields an aggregated system in the sense of ODE lumpability [36, 30]. This theory covers linear transformations of the original state variables in general; here we consider an instance, which we call *ordinary fluid lumpability*, where the transformation is induced by a partition of state variables. (Forward bisimulation is presented in Section 3.1.) Backward bisimulation (presented in Section 3.2) is related to *exact fluid lumpability*, introduced in the context of process algebra with fluid semantics [37], identifying process terms with the same ODE solution when initialized equally. The disadvantage of forward CRN bisimulation is that it is lossy (yet exact) because, similarly to the forward stochastic analogues, from the aggregated ODE system in general it is not possible to recover the solutions for the individual species within the same equivalence class. On the other hand, it does not impose restrictions on the initial conditions, which instead are present in our backward variant. As a further important difference, forward CRN bisimulation (again, like its stochastic analogues) turns out to be a sufficient condition for ODE lumpability. Instead, backward CRN bisimulation enjoys a full characterization, in the sense that there exists a backward CRN bisimulation between two species if and only if they have the same ODE solutions (provided that they start from equal initial conditions). More in general, by means of a number of examples we will show that the two equivalences are complementary because not comparable. In other words, there exist models that can be reduced up to forward CRN bisimulation but not by the backward variant, and vice versa; at the same time, there are models that can be reduced by both.

To enhance the usefulness of these notions, we present (in Section 5) a *template* partition-refinement algorithm that is parametric with respect to the equivalence of interest, computing the coarsest refinement up to either variant in polynomial time. To use our equivalences as an automatic model reduction tool, we further give two algorithms (in Section 4) that provide the quotient CRN induced by either bisimulation. With a prototype implementation available at `http://sysma.imtlucca.it/crnreducer/`, we show (in Section 6) that we are able to

reduce a number of case studies taken from the literature. Our bisimulations yielded quotient CRNs with number of reactions and species up to four orders of magnitude smaller than the original CRNs, leading to speed-ups in the ODE solution runtimes of up to five orders of magnitude. In two cases, it was possible to analyze models that were otherwise intractable directly within our experimental environment due to excessive memory requirements.

Related work. Behavioral equivalences have been recently proposed in [32] for comparing CRNs; however, the analysis is carried out at the qualitative level, i.e., ignoring the dynamical evolution. In [37] is introduced the notion of *label equivalence* for process algebra with fluid semantics, which captures exact fluid lumpability (processes are equivalent whenever their ODE solutions are equal at all time points). However, unlike backward CRN bisimulation, label equivalence is only a sufficient condition for ODE reduction. Indeed, it works at a coarser level of granularity as it relates *sets* of ODE variables, each corresponding to the behavior of a sequential process. Instead, backward CRN bisimulation relates individual ODE variables. Further, the conditions for equivalence, specific to the process algebra, are difficult to check automatically because of the universal quantifiers over the ODE variables. More important, no algorithm for computing the coarsest partition was developed. Similar considerations apply to the process-algebra specific ordinary fluid lumpability in [38].

Cardelli's notion of *emulation* between two CRNs is a (structural) mapping of species and reactions that, like backward CRN bisimulation, guarantees the equality between the ODE solutions at all time points [11]. An emulation requires a source and a target CRN — the modeler is intended to have the suspicion that, for some given CRN, another CRN might be related to it. But emulation cannot be used when one wants to discover equivalences between species *within the same given CRN*. Thus, emulation is not useful for model reduction because a-priori information about the structure of a quotient CRN is not available. Furthermore, no algorithm is provided in [11] to find emulations automatically. Since backward CRN bisimulation fully characterizes exact fluid lumpability, it is not difficult to show that backward CRN bisimulation generalizes emulation in the sense that any emulation between two CRNs can be understood in terms of a backward CRN bisimulation over the species of a "union CRN" that contains all the reactions of the two CRNs of interest.

Model reductions have been studied in related models for biomolecular networks (e.g. [17, 22, 10]), most notably for rule-based systems such as BioNetGen [5] and the κ calculus [18]. These offer an intensional modeling approach, by providing graph-rewrite rules of interaction instead of a complete enumeration of all chemical reactions involved. *Differential fragments* for κ are self-consistent aggregates found by a static analysis on the model, identifying sums of chemical species for which an ODE system can be explicitly written [17]. In this sense, they are analogous to our CRN bisimulations, but with notable differences. First, fragmentation works directly at the rule-based level. This has the advantage that the analysis is performed on a set of rewrite rules, which is typically much more compact than the fully enumerated CRN. However, fragmentation is domain-specific, hence the model must be conveniently expressed as a biomolecular system (e.g., with complex formation or internal state modification). On the other hand, CRN bisimulations work for a generic language-independent CRN, which however must be explicitly given. Further, unlike CRN bisimulations, fragmentation is performed on a "static" view of the model, without information on the reaction rates. The ODE aggregations of both forward CRN bisimulation and fragmentation introduce loss of information (in contrast to backward CRN bisimulation). But, unlike our forward variant, in fragmentation the same species may be present in more than one fragment. Additionally, species may occur in fragments with multiplicity numbers. Thus, fragmentation can be seen

as a form of *improper lumping* that is not necessarily induced by a partition of the original state-space variables [30]. Overall, because of these differences, it is not difficult to find models that can be reduced by our CRN bisimulations and not by fragmentation, and vice versa. This is presented in detail in Section 6.

2 Background

Notation. We write $A \to B$ and B^A for the functions from A to B. When $f \in A \to B$ and $a \in A$, we set $f_a := f(a)$. Moreover, for any $X \subseteq A$ and $b \in B$, we define $f(X) := \{b \in B \mid \exists a \in X.(f(a) = b)\}$. Sets and multisets are denoted by $\{\ldots\}$ and $\{|\ldots|\}$, respectively. Also, we shall not distinguish among an equivalence relation and the partition induced by it, and shall use the symbol $\sim_{\mathcal{H}}$ to denote the equivalence relation with $\mathcal{H} = S/\!\sim_{\mathcal{H}}$. Finally, given two partitions $\mathcal{H}_1$ and $\mathcal{H}_2$ of a given set S, we say that $\mathcal{H}_1$ is a *refinement* of $\mathcal{H}_2$ if for any $H_1 \in \mathcal{H}_1$ there exists a (unique) $H_2 \in \mathcal{H}_2$ such that $H_1 \subseteq H_2$.

2.1 Chemical Reaction Networks

Formally, a CRN (S, R) is a pair consisting of a finite set of species S taken from a countable infinite universe of all species, and a finite set of chemical reactions R. A reaction is a triple written in the form $\rho \xrightarrow{\alpha} \pi$, where ρ and π are the multisets of speciese *reactants* and *products*, respectively, and $\alpha > 0$ is the reaction rate. In particular, we focus on basic chemistry where only *elementary reactions* are considered, where at most two reactants (possibly of the same species) interact. No restrictions are instead imposed on products. Several models found in the literature (including those discussed in Section 6) belong to this class. Also, this is consistent with the physical considerations which stipulate that reactions with more than two reactants are very unlikely to occur in nature [24]. We denote by $\rho(X)$ the multiplicity of species X in the multiset ρ, and by $\mathcal{MS}(S)$ the set of finite multisets of species in S. To adhere to standard chemical notation, we shall use the operator $+$ to denote multiset union, e.g., $X + Y + Y$ (or just $X + 2Y$) denotes the multiset $\{|X, Y, Y|\}$. We may also use X to denote either the species X or the singleton $\{|X|\}$.

The (autonomous) ODE system $\dot{V} = F(V)$ underlying a CRN (S, R) is $F : \mathbb{R}^S_{\geq 0} \to \mathbb{R}^S$, where each component F_X, with $X \in S$ is defined as:

$$F_X(V) := \sum_{\rho \xrightarrow{\alpha} \pi \in R} (\pi(X) - \rho(X)) \cdot \alpha \cdot \prod_{Y \in S} V_Y^{\rho(Y)} \ .$$

This represents the well-known *mass-action* kinetics, where the reaction rate is proportional to the concentrations of the reactants involved. Since the ODE system of a CRN is given by polynomials, the vector field F is locally Lipschitz. Hence, the theorem of Picard-Lindelöf ensures that for any $V(0) \in \mathbb{R}^S_{\geq 0}$ there exists a unique non-continuable solution of $\dot{V} = F(V)$.

▶ **Example 1.** We now provide a simple CRN, (S_e, R_e), with $S_e = \{A, B, C, D, E\}$ and $R_e = \{A \xrightarrow{6} E, B \xrightarrow{6} D, A+B \xrightarrow{2} C, C+D \xrightarrow{5} 2C+D, E+D \xrightarrow{5} 2E+D\}$, which will be used as a running example throughout the paper. Its ODE system is given by

$$\dot{V}_A = -6\,V_A - 2\,V_A\,V_B \qquad \dot{V}_B = -6\,V_B - 2\,V_A\,V_B \qquad \dot{V}_C = 2\,V_A\,V_B + 5\,V_C\,V_D$$
$$\dot{V}_D = 6\,V_B \qquad \dot{V}_E = 6\,V_A + 5\,V_E\,V_D$$

In the following, we shall assume that the universe of all species is well-ordered with respect to $\sqsubseteq$. We then say that a function $\mu : S \to S$ is a *choice function* of a partition $\mathcal{H}$ of S, if $\mu(X) = \min_{\sqsubseteq} H$ for all $H \in \mathcal{H}$ and $X \in H$. Also, choice functions can be trivially lifted to multisets applying them element-wise, e.g., $\mu(X + Y) = \mu(X) + \mu(Y)$.

2.2 Fluid Lumpability

Ordinary Fluid Lumpability. We start by defining the notion of ordinary fluid lumpability, which is an instance of *ordinary lumpability* for ODEs [36] specialized to CRNs.

▶ **Definition 2** (Ordinary fluid lumpability)**.** Let (S, R) be a CRN, F be its vector field, and $\mathcal{H} = \{H_1, \ldots, H_m\}$ a partition of S. Then, $\mathcal{H}$ is *ordinary fluid lumpable* if for all $H \in \mathcal{H}$ there exists a polynomial $\wp_H$ in $|\mathcal{H}|$ variables such that $\sum_{X\in H} F_X(V) = \wp_H(\sum_{X\in H_1} V_X, \ldots, \sum_{X\in H_m} V_X)$ for all $V \in \mathbb{R}^S_{\geq 0}$.

Informally, a partition $\mathcal{H}$ is ordinary fluid lumpable if, for each $H \in \mathcal{H}$, the polynomial $\sum_{X\in H} F_X(V)$ in the variables $\{V_X \mid X \in S\}$ can be rewritten into a polynomial $\wp_H$ in the variables $\{\sum_{X\in H} V_X \mid H \in \mathcal{H}\}$. In particular, if $\mathcal{H}$ is known to be an ordinary fluid lumpable partition of (S, R) and V denotes the solution of $\dot{V} = F(V)$ subject to $V(0) \in \mathbb{R}^S_{\geq 0}$, the solution of the aggregated ODE system $(\dot{W}_{H_1}, \ldots, \dot{W}_{H_m}) = (\wp_{H_1}(W), \ldots, \wp_{H_m}(W))$ with $W_H(0) = \sum_{X\in H} V_X(0)$ is such that $W_H(t) = \sum_{X\in H} V_X(t)$ for all $t \in \text{domain}(V)$.

▶ **Example 3.** Consider the ODEs of (S_e, R_e) of Example 1, and let $\mathcal{H}_O = \{\{A\}, \{B\}, \{C, E\}, \{D\}\}$. By applying a variable renaming consistent with the blocks of $\mathcal{H}_O$, i.e., $V_{CE} = V_C + V_E$, and by exploiting the linearity of the differential operator we get

$$\dot{V}_A = -6V_A - 2V_AV_B \quad \dot{V}_B = -6V_B - 2V_AV_B \quad \dot{V}_{CE} = 2V_AV_B + 6V_A + 5V_DV_{CE} \quad \dot{V}_D = 6V_B$$

That is, we obtained an ODE system in terms of block variables only. ◀

Exact Fluid Lumpability. We extend to CRNs the notion of exact fluid lumpability in [37].

▶ **Definition 4** (Exact fluid lumpability)**.** Let (S, R) be a CRN, F its vector field, and $\mathcal{H}$ a partition of S. We call $V \in \mathbb{R}^S$ *constant on* $\mathcal{H}$ if $V_{X_i} = V_{X_j}$ for all $H \in \mathcal{H}$, and all $X_i, X_j \in H$. Then, $\mathcal{H}$ is *exactly fluid lumpable* if $F(V)$ is constant on $\mathcal{H}$ whenever V is constant on $\mathcal{H}$.

▶ **Example 5.** Consider the ODEs of (S_e, R_e) of Example 1, and let $\mathcal{H}_E = \{\{A, B\}, \{C\}, \{D\}, \{E\}\}$. It is easy to see that A and B have same concentrations at all time points if initialized equally. In these cases, we can replace the ODEs of (S_e, R_e) with the ones aggregated according to $\mathcal{H}_E$, obtained by removing $\dot{V}_B$ and replacing all occurrences of V_B with V_A:

$$\dot{V}_A = -6\,V_A - 2\,V_A\,V_A \quad \dot{V}_C = 2\,V_A\,V_A + 5\,V_C\,V_D \quad \dot{V}_D = 6\,V_A \quad \dot{V}_E = 6\,V_A + 5\,V_E\,V_D$$

That is, we obtained a (lossless) aggregated ODE system written in terms of a variable per block, chosen according to $\sqsubseteq$. ◀

We remark that the above definition expresses exact fluid lumpability in terms of properties of the ODE vector field of a CRN. Instead, in [37] exactly fluid lumpability was defined directly in terms of the desired dynamical property, i.e., that the ODE solutions within any equivalence class be equal at all time points. The following result is a new contribution showing that this dynamical property is fully characterized by the vector-field based definition.

▶ **Theorem 6.** *Let (S, R) be a CRN and F its vector field. A partition $\mathcal{H}$ of S is exactly fluid lumpable if and only if, for any $V(0) \in \mathbb{R}^S_{\geq 0}$ that is constant on $\mathcal{H}$, the underlying solution of $\dot{V} = F(V)$ is such that $V(t)$ is constant on $\mathcal{H}$ for all $t \in$ domain(V).*[1]

[1] All proofs are provided in the extended technical report [13].

3 CRN Bisimulations

Both notions of fluid lumpability given in Section 2 are not convenient to be used directly because they involve a universal quantifier over the (uncountable) state space. We address this problem by providing structural conditions that concern only the reactions of a CRN.

3.1 Forward CRN Bisimulation

We now introduce forward CRN bisimulation, an equivalence on species that will turn out to induce ordinary fluid lumpability. We start with the notions of *reaction* and *production rate*. The former collects the rates at which the concentration of a species X decreases when reacting with a given partner. The latter collects the positive contribution that X exerts to the concentration of a species Y, again when reacting with a certain partner.

▶ **Definition 7** (Reaction and production rates)**.** Let (S, R) be a CRN, $X, Y \in S$, and $\rho \in \mathcal{MS}(S)$. The *ρ-reaction rate* of X, and the *ρ-production rate* of Y-elements by X are defined respectively as

$$\mathbf{crr}[X,\rho] := (\rho(X)+1) \sum_{X+\rho \xrightarrow{\alpha} \pi \in R} \alpha, \qquad \mathbf{pr}(X,\rho,Y) := (\rho(X)+1) \sum_{X+\rho \xrightarrow{\alpha} \pi \in R} \alpha \cdot \pi(Y)$$

Finally, for $H \subseteq S$ we define $\mathbf{pr}[X,\rho,H] := \sum_{Y \in H} \mathbf{pr}(X,\rho,Y)$.

▶ **Definition 8** (Forward CRN Bisimulation)**.** Let (S, R) be a CRN, $\mathcal{R}$ an equivalence relation over S and $\mathcal{H} = S/\mathcal{R}$. Then, $\mathcal{R}$ is a forward CRN bisimulation (abbreviated FB) if for all $(X, Y) \in \mathcal{R}$, all $\rho \in \mathcal{MS}(S)$, and all $H \in \mathcal{H}$ it holds that

$$\mathbf{crr}[X,\rho] = \mathbf{crr}[Y,\rho] \quad \text{and} \quad \mathbf{pr}[X,\rho,H] = \mathbf{pr}[Y,\rho,H] \tag{1}$$

▶ **Example 9.** Consider $\mathcal{H}_O = \{\{A\},\{B\},\{C,E\},\{D\}\}$ of Example 3. It can be shown that $\mathcal{H}_O$ is an FB, as, e.g., $\mathbf{crr}[C,D] = \mathbf{crr}[E,D] = 5$, and $\mathbf{pr}[C,D,\{C,E\}] = \mathbf{pr}[E,D,\{C,E\}] = 10$. ◀

We are interested in the coarsest FB, as well as in the coarsest one refining a given initial partition of species.

▶ **Proposition 10.** *Let (S, R) be a CRN, I a set of indices, and $\mathcal{R}_i$ an FB for (S, R), for all $i \in I$. The transitive closure of their union $\mathcal{R} = (\bigcup_{i\in I} \mathcal{R}_i)^*$ is an FB for (S, R). In particular, if each R_i is such that $S/\mathcal{R}_i$ refines some partition $\mathcal{G}$ of S, then so does $S/\mathcal{R}$.*

▶ **Theorem 11** (Forward bisimulation implies ordinary fluid lumpability)**.** *Let (S, R) be a CRN. Then, $\mathcal{H}$ is an ordinarily fluid lumpable partition of S if $\mathcal{H}$ is an FB of S.*

FB is only a sufficient condition for lumpability, as discussed in the next example. (However, Section 6 shows that FB can be effectively applied to interesting existing models.)

▶ **Example 12.** Consider the CRN $(\{F, G\}, \{F \xrightarrow{\alpha_1} G, G \xrightarrow{\alpha_2} F\})$, having ODEs

$$\dot{V}_F = -\alpha_1 V_F + \alpha_2 V_G \qquad \dot{V}_G = -\alpha_2 V_G + \alpha_1 V_F$$

If $\alpha_1 \neq \alpha_2$, $\mathcal{H}_c = \{\{G, F\}\}$ is not an FB, as $\mathbf{crr}[F,\emptyset] = \alpha_1$ and $\mathbf{crr}[G,\emptyset] = \alpha_2$. Nevertheless, the above ODE system is lumpable. Indeed, by applying the variable renaming consistent with $\mathcal{H}_c$, i.e., $V_{FG} = V_F + V_G$, we get a single ODE for V_{FG}, i.e., $\dot{V}_{FG} = 0$. ◀

3.2 Backward CRN Bisimulation

We now introduce backward CRN bisimulation, an equivalence on species that will turn out to characterize exact fluid lumpability. We start with the notion of cumulative flux rate, which collects the overall contribution that reactions with a given multiset of reactants ρ exert to the concentration of a species X.

▶ **Definition 13** (Cumulative flux rate)**.** Let (S, R) be a CRN, $X \in S$, $\rho \in \mathcal{MS}(S)$, and $\mathcal{M} \subseteq \mathcal{MS}(S)$. Then, we define

$$\mathbf{fr}(X, \rho) := \sum_{\rho \xrightarrow{\alpha} \pi \in R} (\pi(X) - \rho(X)) \cdot \alpha, \qquad \mathbf{fr}[X, \mathcal{M}] := \sum_{\rho \in \mathcal{M}} \mathbf{fr}(X, \rho).$$

We call $\mathbf{fr}(X, \rho)$ and $\mathbf{fr}[X, \mathcal{M}]$ *ρ-flux rate* and *cumulative $\mathcal{M}$-flux rate* of X, respectively.

▶ **Definition 14** (Backward CRN bisimulation)**.** Let (S, R) be a CRN, $\mathcal{R}$ an equivalence relation over S, $\mathcal{H} = S/\mathcal{R}$ and μ the choice function of $\mathcal{H}$. Then, $\mathcal{R}$ is a backward CRN bisimulation (BB) if for any $(X, Y) \in \mathcal{R}$ it holds that

$$\mathbf{fr}[X, \mathcal{M}] = \mathbf{fr}[Y, \mathcal{M}] \quad \text{for all} \quad \mathcal{M} \in \{\rho \mid \rho \xrightarrow{\alpha} \pi \in R\}/\approx_{\mathcal{H}}, \tag{2}$$

where any two $\rho, \sigma \in \mathcal{MS}(S)$ satisfy $\rho \approx_{\mathcal{H}} \sigma$ if $\mu(\rho) = \mu(\sigma)$.

▶ **Example 15.** Consider $\mathcal{H}_E = \{\{A, B\}, \{C\}, \{D\}, \{E\}\}$ of Example 5. We first note that $\{\!|A|\!\} \approx_{\mathcal{H}_E} \{\!|B|\!\}$, as $\approx_{\mathcal{H}_E}$ relates multisets with same number of $\mathcal{H}_E$-equivalent species. Also, it can be shown that $\mathcal{H}_E$ is a BB, as, e.g., $\mathbf{fr}[A, \mathcal{M}] = \mathbf{fr}[B, \mathcal{M}] = -6$ for $\mathcal{M} = \{\{\!|A|\!\}, \{\!|B|\!\}\}$. ◀

As for FB, there exists a coarsest BB (that refines a given partition of S).

▶ **Proposition 16.** *Let (S, R) be a CRN, I a set of indices, and $\mathcal{R}_i$ a BB for (S, R), for all $i \in I$. The transitive closure of their union $\mathcal{R} = (\bigcup_{i \in I} \mathcal{R}_i)^*$ is a BB for (S, R). In particular, if each R_i is such that $S/\mathcal{R}_i$ refines some partition $\mathcal{G}$ of S, then so does $S/\mathcal{R}$.*

We now state the mentioned characterization of exact fluid lumpability in terms of BB.

▶ **Theorem 17** (Backward bisimulation characterizes exact fluid lumpability)**.** *Let (S, R) be a CRN. Then, $\mathcal{H}$ is an exactly fluid lumpable partition of S if and only if $\mathcal{H}$ is a BB of S.*

▶ Remark. We wish to stress that FB and BB are not comparable: First, $\mathcal{H}_O$ is not a BB, as $\mathbf{fr}[C, \{A+B\}] = 2$ and $\mathbf{fr}[E, \{A+B\}] = 0$; Second, $\mathcal{H}_E$ is not an FB, as $\mathbf{crr}(A, B) = 2$ and $\mathbf{crr}(B, B) = 0$; Third, for the same reasons, $\{\{A, B\}, \{C, E\}, \{D\}\}$ is neither an FB nor a BB. Similar examples on models of biological relevance are provided in Section 6. ◀

4 Reduced Chemical Reaction Networks up to CRN Bisimulations

We have shown that, given a CRN and a CRN bisimulation $\mathcal{R}$, we can analyze the aggregated ODE system according to $\mathcal{R}$. We now provide the notion of reduced CRN from which the aggregated ODEs can be directly generated, as depicted in Figure 1.

▶ **Definition 18** (Forward reduction)**.** Let (S, R) be a CRN, $\mathcal{H}$ an FB, and μ its choice function. The *$(\mathcal{H}, F)$-reduction of* (S, R) is given by $(S, R)^{(\mathcal{H},F)} = (S^{(\mathcal{H},F)}, R^{(\mathcal{H},F)})$, where $S^{(\mathcal{H},F)} = \mu(S)$ and $R^{(\mathcal{H},F)}$ is defined as follows: (F1) Discard all reactions $\rho \xrightarrow{\alpha} \pi$ such that $\rho \neq \mu(\rho)$; (F2) Replace all remaining reactions $\rho \xrightarrow{\alpha} \pi$ with $\rho \xrightarrow{\alpha} \mu(\pi)$; (F3) Fuse all reactions that have the same reactants and products by summing their rates.

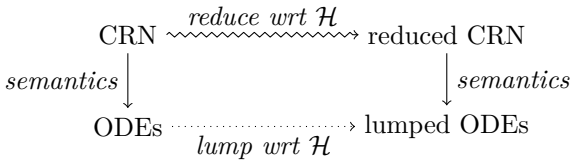


Figure 1 Relation among ($\mathcal{H}$-reduced) CRNs and ($\mathcal{H}$-lumped) semantics, with $\mathcal{H}$ a bisimulation.

The idea underlying forward reduction is to discard all reactions having non-representative reagents, and to replace the products of the remaining reactions with their representatives. This can be seen as a special case of Theorem 4.4 of [10].

▶ **Example 19.** Consider the FB $\mathcal{H}_O = \{\{A\},\{B\},\{C,E\},\{D\}\}$ used in Example 3. The $(\mathcal{H}_O, F)$-reduction of (S_e, R_e) is (with C being the representative of its block) $S_e^{(\mathcal{H}_O,F)} = \{A,B,C,D\}$, $R_e^{(\mathcal{H}_O,F)} = \{A \xrightarrow{6} C, B \xrightarrow{6} D, A{+}B \xrightarrow{2} C, C{+}D \xrightarrow{5} 2C{+}D\}$. Note that the reaction $E{+}D \xrightarrow{5} 2E{+}D$ is discarded, as E is not a representative species. ◀

We now state that the $(\mathcal{H}, F)$-reduction of an FB $\mathcal{H}$ induces the ODEs aggregated according to $\mathcal{H}$. For example, the $(\mathcal{H}_O, F)$-reduction of (S_e, R_e) induces the ODEs shown in Example 3, if applying the renaming $V_C = V_{CE}$.

▶ **Theorem 20** (Forward reduction induces aggregation)**.** *Let (S,R) be a CRN, $\mathcal{H}$ an FB and μ its choice function. Then, $(S,R)^{(\mathcal{H},F)}$ is computed in at most $\mathcal{O}\big(|R|\cdot|S|\cdot(\log(|R|)+\log(|S|))\big)$ steps. Crucially, if F is the vector field of (S,R) and $\hat{F}$ the one of $(S,R)^{(\mathcal{H},F)}$, then $\sum_{X\in H} F_X(V) = \hat{F}_{\mu(Y)}(\sum_{X\in H_1} V_X, \ldots, \sum_{X\in H_m} V_X)$ for all $V \in \mathbb{R}^S_{\geq 0}$, $H \in \mathcal{H}$ and $Y \in H$.*

For the backward reduction, the underlying idea is to keep track only of differential contributions that affect the representative species $\mu(S)$.

▶ **Definition 21** (Backward reduction)**.** Let (S,R) be a CRN, $\mathcal{H}$ a BB, and μ its choice function. The *$(\mathcal{H},B)$-reduction of* (S,R) is given by $(S,R)^{(\mathcal{H},B)} = (S^{(\mathcal{H},B)}, R^{(\mathcal{H},B)})$, where $S^{(\mathcal{H},B)} = \mu(S)$ and $R^{(\mathcal{H},B)}$ is obtained as follows: (B1) Replace all reactions $\rho \xrightarrow{\alpha} \pi$ with $\rho \xrightarrow{\alpha} \tilde{\pi}$ where $\tilde{\pi}(X_i) := \pi(X_i)$ if $X_i \in \mu(S)$ and $\tilde{\pi}(X_i) := \rho(X_i)$ otherwise; (B2) Replace all such obtained reactions $\rho \xrightarrow{\alpha} \pi$ with $\mu(\rho) \xrightarrow{\alpha} \mu(\pi)$; (B3) Fuse all reactions that have the same reactants and products by summing their rates.

▶ **Example 22.** Considering the CRN (S_e, R_e) and the BB $\mathcal{H}_E$, (B1) changes $B \xrightarrow{6} D$ in $B \xrightarrow{6} D{+}B$, and $A{+}B \xrightarrow{2} C$ in $A{+}B \xrightarrow{2} C{+}B$, while (B2) yields $\{A \xrightarrow{6} E, A \xrightarrow{6} D{+}A, A{+}A \xrightarrow{2} C{+}A, C{+}D \xrightarrow{5} 2C{+}D, E{+}D \xrightarrow{5} 2E{+}D\}$. Finally, (B3) does not introduce any change. ◀

▶ **Theorem 23** (Backward reduction induces aggregation)**.** *Let (S,R) be a CRN, $\mathcal{H}$ a BB and μ its choice function. Then, $(S,R)^{(\mathcal{H},B)}$ is computed in at most $\mathcal{O}\big(|R|\cdot|S|\cdot(\log(|R|)+\log(|S|))\big)$ steps. Crucially, if $\hat{F}$ denotes the vector field induced by $(S,R)^{(\mathcal{H},B)}$, it holds that $F_X(V) = \hat{F}_X(V)$ for all $X \in \mu(S)$ and $V \in \mathbb{R}^S_{\geq 0}$ that are constant on $\mathcal{H}$.*

5 Partition Refinement Algorithms for CRN Bisimulations

We study a polynomial-time algorithm for the computation of the coarsest bisimulations that refine an arbitrary input partition. We start introducing two auxiliary equivalence relations.

▶ **Definition 24** (Splitter equivalences)**.** Let (S,R) be a CRN and $\mathcal{H}$ a partition over S. Then, we write $X \sim^F_{\mathcal{H}} Y$ if (1) is fulfilled by (X,Y). Similarly, write $X \sim^B_{\mathcal{H}} Y$ if (X,Y) satisfies (2).

Algorithm 1 Template partition refinement algorithm for the construction of the coarsest CRN bisimulations that refine some given initial partition $\mathcal{G}$.

Require: A CRN (S, R), a partition $\mathcal{G}$ of S and $\chi \in \{F, B\}$.
  $\mathcal{H} \longleftarrow \mathcal{G}$
  while true do
    $\mathcal{H}' \longleftarrow S/(\sim^{\chi}_{\mathcal{H}} \cap \sim_{\mathcal{H}})$
    if $\mathcal{H}' = \mathcal{H}$ **then**
      return $\mathcal{H}$
    else
      $\mathcal{H} \longleftarrow \mathcal{H}'$
    end if
  end while

Algorithm 1 iteratively computes the coarsest forward or backward bisimulation (when $\chi = F$ or $\chi = B$, respectively) that refines a given input partition of species of a CRN. Note that, contrary to CRN reduction algorithms, one (parametric) algorithm suffices for both bisimulations. Using the above splitter equivalences, at each iteration the blocks of the current partition $S/\sim_{\mathcal{H}}$ are split in sub-blocks of $\sim^{\chi}_{\mathcal{H}}$-equivalent species $S/(\sim^{\chi}_{\mathcal{H}} \cap \sim_{\mathcal{H}})$. The algorithm terminates when no refinement is performed.

The freedom in choosing the initial partition $\mathcal{G}$ is useful in both bisimulations. For FB it allows to single out species that are the "observables" of the CRN. These are the species for which the modeler is interested in obtaining distinct ODE solutions, information which would otherwise be lost if such species are found in larger equivalence classes. BB is lossless, hence this issue does not arise. However BB requires the same initial conditions for equivalent species. In this case, an appropriate input partition may tell apart species for which it is known that the initial conditions are different.

▶ **Theorem 25** (Correctness). *Given a CRN (S, R) and a partition $\mathcal{G}$ of S, Algorithm 1 calculates the coarsest forward and backward bisimulation that refines $\mathcal{G}$. In both cases, the number of steps needed is polynomial in the number of species and reactions.*

Note that, due to space constraints, we only focussed on the existence of a polynomial-time algorithm, and in the next section we provide numerical evidence of its scalability. The proof of this theorem gives a bound of $\mathcal{O}(|R|^2 \cdot |S|^5)$ on the number of steps. Tighter bounds could be obtained by extending classical partition refinement approaches available for labeled transitions systems [31, 1] to CRNs, which is however the subject of future work.

6 Evaluation

We now evaluate FB and BB. We first study their effectiveness in reducing the ODEs of a number of biochemical models from the literature given in the `.net` format of BioNetGen [5], version 2.2.5-stable. Using selected models we discuss how FB and BB relate with each other, and provide a biological interpretation of the aggregations. Finally, we compare them against κ's fragmentation. All experiments are replicable using a prototype available at `http://sysma.imtlucca.it/crnreducer/`.

Numerical results. Table 1 lists our case studies: four synthetic benchmarks to obtain combinatorially larger CRNs by varying the number of phosphorylation sites (M1–M4) [33]; a model of pheromone signaling (M5, [35]); two signaling pathways through the Fcε complex

Table 1 Forward and backward reductions and corresponding speed-ups in ODE analysis. Speed-up entries "—" indicate that the original model could not be solved; entries "x" indicate that the coarsest bisimulation did not reduce the original model.

Original model			*Forward reduction*				*Backward reduction*			
Id	$\|R\|$	$\|S\|$	*Red.(s)*	$\|R\|$	$\|S\|$	*Speed-up*	*Red.(s)*	$\|R\|$	$\|S\|$	*Speed-up*
M1	3538944	262146	4.61E+4	990	222	—	7.65E+4	2708	222	—
M2	786432	65538	1.92E+3	720	167	—	3.68E+3	1950	167	—
M3	172032	16386	8.15E+1	504	122	1.16E+3	1.77E+2	1348	122	5.34E+2
M4	48	18	1.00E–3	24	12	1.00E+0	2.00E–3	45	12	1.00E+0
M5	194054	14531	3.72E+1	142165	10855	1.03E+0	1.32E+3	93033	6634	1.03E+0
M6	187468	10734	3.07E+1	57508	3744	1.92E+1	2.71E+2	144473	5575	3.53E+0
M7	32776	2506	1.26E+0	16481	1281	6.23E+0	1.66E+1	32776	2506	x
M8	41233	2562	1.12E+0	33075	1897	1.12E+0	1.89E+1	41233	2562	x
M9	5033	471	1.91E–1	4068	345	1.04E+0	4.35E–1	5033	471	x
M10	5797	796	1.61E–1	4210	503	1.47E+0	7.37E–1	5797	796	x
M11	5832	730	3.89E–1	1296	217	1.32E+1	6.00E–1	2434	217	7.55E+0
M12	487	85	2.00E–3	264	56	1.88E+0	6.00E–3	426	56	1.31E+0
M13	24	18	1.20E–2	24	18	x	7.00E–3	6	3	1.00E+0

(M6–M7, [20, 33]); two models of enzyme activation (M8–M9, [2]); a model of a tumor suppressor protein (M10, [3]); a model of tyrosine phosphorylation and adaptor protein binding (M11, [14, 15]); a MAPK model (M12, [28]); and an *influence network* (M13, [11]).

Headers $|R|$ and $|S|$ give the number of reactions and species of the CRN (and of its reductions), respectively. The reduction times (*Red.*) account also for the computation of the quotient CRNs. The speed-up is the ratio between the time to solve the ODEs of the original CRN and that of the reduced one including the time to reduce the CRN. Measurements were taken on a 2.6 GHz Intel Core i5 with 4 GB of RAM. The time interval of the ODE solution was taken from the original papers; for M1–M4, where this data was not available, time point 50.0 was used as an estimate of steady state. The initial conditions for the ODEs were also taken from the original papers. The initial partition for FB was chosen to be the trivial one containing the singleton block $\{S\}$ (i.e., no species was singled out). Instead, the initial partition for BB was chosen consistently with the ODE initial conditions; that is, two species may be equivalent only if they have the same initial conditions in the original CRN. This ensured that the backward reduced CRN was a lossless aggregation of the original CRN.

We make three main observations: (i) FB and BB can reduce a significant number of models. In the two largest models of our case studies, M1 and M2, the bisimulations were able to provide a compact aggregated ODE system which could be straightforwardly analyzed, while the solutions of the original models did not terminate due to out-of-memory errors, consistently with [33]. (ii) FB and BB are not comparable in general. For instance, both reduce M5 to 10855 and 6634 species, respectively, while M6 is reduced to 3744 species by FB, and to 5574 by BB. Also, FB was able to reduce M7–M10, while BB did not aggregate. The influence network M13 shows the opposite; in fact, none of the influence networks presented in [11] can be reduced up to FB (here we showed M13, which is the largest one from [11]). (iii) Models M1–M4 and M12 show that the intersection between FB and BB is nonempty.

Biological interpretation. Models M1 and M2 enjoy significant reductions and ODE analysis speed-ups. Here we use them to explain that FB and BB are effective at aggregating species representing symmetric sites in a complex. For this, let us consider M4, chosen for space reasons. A typical equivalence class is for instance $\{E(s!1).S(p1{\sim}P, p2{\sim}U!1), E(s!1).S(p1{\sim}U!1, p2{\sim}P)\}$. According to the syntax of the BioNetGen language, the CRN species are

formed from basic *molecules* S and E. Molecule S has two binding sites ($p1$, and $p2$) which can be either in *phosphorylated* state (P) or not (U); E has one stateless binding site (s) which can bind to those of S to form a complex. The two sites of S have equivalent capabilities in terms of binding with other species or changing state. For instance, the above equivalence class contains two species composed by S and E, with E bound to the unphosphorylated site of S (here the exclamation mark links the binding sites used to form the species). Models M1 and M2 exhibit a fast growth of the number of species due to a larger number of symmetric sites, requiring distinct species to track exactly which site exhibits a particular phosphorylation state. This form of symmetry has also been studied in [9] where the authors propose an approach to detect it directly at the κ level. However, an experimental comparison could not be performed because [9] is not yet implemented. Although both bisimulations give the same equivalence classes in these cases, the reduced CRNs have different reactions, since FB provides the dynamics of the sums of equivalent species, while BB considers the distinct dynamics of representative species. Instead, aggregation of identical binding sites is supported by BioNetGen. This can be seen in models M6 and M7, since they both have $Lig(l, l)$, a ligand with two copies of site l. Intuitively, the rule

$$Rec(a) + Lig(l, l) \to Rec(a!1).Lig(l!1, l) \tag{3}$$

gives rise to only one chemical complex in the underlying CRN, $Rec(a!1).Lig(l!1, l)$. This represents the (forward and backward) canonical representative of a ligand bound to a single receptor $Rec(a)$. To see this, let us rename the two sites and *expand* the rule appropriately:

$$\begin{aligned} &Rec(a) + Lig(l_1, l_2) \to Rec(a!1).Lig(l_1!1, l_2) \\ &Rec(a) + Lig(l_1, l_2) \to Rec(a!1).Lig(l_1, l_2!1) \end{aligned} \tag{4}$$

Then, this underlying CRN will distinguish the two sites. However, applying either of our CRN bisimulations leads to the CRN for Equation (3).

We remark that the original CRN sizes of M6 and M7 already account for the aggregations obtained with BioNetGen. Nevertheless, our CRN bisimulations allow for further (significant) reductions. For instance, part of the reductions for M6 are due to the presence of $Rec(a, b, g_1, g_2)$, a molecule with symmetric sites g_1 and g_2, similarly to those of M4.

Symmetric sites are not the only property captured by our bisimulations. For instance in both M8 and M9 one of the FB equivalence classes is given by:

$$\begin{aligned} \big\{ &J(k!1).R(x!1, i{\sim}\text{on}, l),\ J(k!1).L(r_1!2, r_2).R(x!1, i{\sim}\text{on}, l!2), \\ &J(k!1).L(r_1, r_2!2).R(x!1, i{\sim}\text{on}, l!2), \\ &J(k!1).L(r_1!2, r_2!3).R(x!1, i{\sim}\text{on}, l!3).R(x, i{\sim}\text{on}, l!2)\big\}. \end{aligned}$$

A biological interpretation is that a species containing the molecule J behaves in the same way as long as it is bound to a molecule R having binding site i in state "on". This is independent of whether R is further complexed with other molecules via its binding site l; For instance, the first species models that R is only bound to J, while in the second and third species it is also bound to L. Finally, in M5, one of the BB equivalence classes is

$$\begin{aligned} \big\{&\text{Dig2(p!1).Ste12(dig1,dig2,dna!1,mapk)},\ \text{Fus3(p!1).Ste12(dig1,dig2,dna!1,mapk)}, \\ &\text{Msg5(p!1).Ste12(dig1,dig2,dna!1,mapk)},\ \text{Sst2(p!1).Ste12(dig1,dig2,dna!1,mapk)}, \\ &\text{Ste12(p!1).Ste12(dig1,dig2,dna!1,mapk)},\ \text{Ste2(p!1).Ste12(dig1,dig2,dna!1,mapk)}\big\}. \end{aligned}$$

It captures that genes Dig2, Fus3, Msg5, Sst2, Ste12, and Ste2, bind to the protein Ste12 with equal rates. This yields equivalent dynamics for these Ste12-gene complexes, and all those formed by them which are equal up to the gene bound to Ste12.

Experimental comparison with κ-based reduction techniques. We now experimentally compare our CRN bisimulations and fragmentation in the case of rule-based biochemical models for which the underlying CRN can be fully enumerated. All models in Table 1 belong to this class; however, none of them was originally available in κ, the only language that supports fragmentation. Thus, we performed a manual translation of a selection of the case studies from the BioNetGen language into κ.[2]

We found:

- *Models that can be reduced by CRN bisimulations but not by fragmentation.* The κ encoding of M12 (a case where only cosmetic syntactical changes are required) returned 85 fragments, equal to the size of the CRN, while both FB and BB reduced to 56 species. The encodings of M6 and M7 necessitated expansions analogous to Equation (4) because κ does not currently support distinct sites with the same name. This led to bigger initial CRNs, for which fragmentation returned 58040 fragments for M6 and 10930 for M7.
- *Models that can be reduced by fragmentation but not by our bisimulations.* The κ model of early events of the EGF pathway in [6] is reduced from 356 species to 38 fragments [17], while no aggregation is obtained with either FB or BB.
- *Models that can be reduced by both our bisimulations and fragmentation.* The κ encodings of models M1–M4 present different reductions than using either bisimulation, specifically 38, 34, 30 and 10 fragments (versus 222, 167, 122, and 12 FB and BB equivalence classes, respectively). It can be shown that, in the latter examples, the reductions are complementary, in the sense that no two bisimilar species are included in the same fragment. While our bisimulations captured symmetric sites, fragments explain that the sites of S are *independent*, i.e., the state of a site does not affect the dynamics of the other. For instance, one of the fragments for model M4 is

$$\{S(p1{\sim}P, p2{\sim}P), S(p1{\sim}P, p2{\sim}U), E(s!1).S(p1{\sim}P, p2{\sim}U!1), F(s!1).S(p1{\sim}P, p2{\sim}P!1)\}$$

  which essentially collects all species where the $p1$ site of molecule S is phosphorylated.

7 Conclusion

Forward and backward bisimulations are equivalence relations over the species of a chemical reaction network inducing a partition of the underlying mass-action system of ordinary differential equations. An experimental evaluation has demonstrated their usefulness by showing their complementarity as well as significant model reductions in a number of biochemical models available in the literature. This has been supported by a prototype, which currently allows a ready-to-use tool-chain with BioNetGen, a state-of-the-art tool.

Ongoing work is studying stochastic counterparts of both forward and backward bisimulations, to obtain model reductions when the semantics of chemical reaction networks based on continuous-time Markov chains is considered. Also, we plan to investigate the applicability of our bisimulations in other model repositories, e.g., those using the well-known SBML interchange format (`http://sbml.org`).

Acknowledgement. The authors thank J. Krivine and J. Feret for helpful discussions, and the anonymous referees for suggestions that improved the paper.

This work was partially supported by the EU project QUANTICOL, 600708. L. Cardelli is partially funded by a Royal Society Research Professorship. Part of this research has

[2] All discussed κ-encodings are provided in the technical report [13] and are available for download.

been carried out while M. Tribastone, M. Tschaikowski, and A. Vandin were at University of Southampton, UK.

References

1 C. Baier, B. Engelen, and M. E. Majster-Cederbaum. Deciding bisimilarity and similarity for probabilistic processes. *J. Comput. Syst. Sci.*, 60(1):187–231, 2000.

2 D. Barua, J. R. Faeder, and J. M. Haugh. A bipolar clamp mechanism for activation of jak-family protein tyrosine kinases. *PLoS Computational Biology*, 5(4), 2009.

3 D. Barua and W. S. Hlavacek. Modeling the effect of apc truncation on destruction complex function in colorectal cancer cells. *PLoS Comput Biol*, 9(9):e1003217, 09 2013.

4 M. Bernardo. A survey of Markovian behavioral equivalences. In *Formal Methods for Perf. Eval.*, volume 4486 of *LNCS*, pages 180–219. Springer Berlin Heidelberg, 2007.

5 M. L. Blinov, J. R. Faeder, B. Goldstein, and W. S. Hlavacek. BioNetGen: software for rule-based modeling of signal transduction based on the interactions of molecular domains. *Bioinformatics*, 20(17):3289–3291, 2004.

6 M. L. Blinov, J. R. Faeder, B. Goldstein, and W. S. Hlavacek. A network model of early events in epidermal growth factor receptor signaling that accounts for combinatorial complexity. *Biosystems*, 83:136–151, 2006.

7 P. Buchholz. Exact and Ordinary Lumpability in Finite Markov Chains. *Journal of Applied Probability*, 31(1):59–75, 1994.

8 P. Buchholz. Markovian Process Algebra: Composition and Equivalence. In *Proc. 2nd Workshop on Process Algebra and Performance Modelling*, Erlangen, Germany, 1994.

9 F. Camporesi and J. Feret. Formal reduction for rule-based models. *Electronic Notes in Theoretical Computer Science*, 276:29–59, 2011. MFPS XXVII.

10 Ferdinanda Camporesi, Jérôme Feret, Heinz Koeppl, and Tatjana Petrov. Combining model reductions. *Electr. Notes Theor. Comput. Sci.*, 265:73–96, 2010.

11 L. Cardelli. Morphisms of reaction networks that couple structure to function. *BMC Systems Biology*, 8(1):84, 2014.

12 L. Cardelli and A. Csikász-Nagy. The cell cycle switch computes approximate majority. *Sci. Rep.*, 2, 2012.

13 L. Cardelli, M. Tribastone, M. Tschaikowski, and A. Vandin. Forward and Backward Bisimulations for Chemical Reaction Networks. Extended Version. `http://arxiv.org/abs/1507.00163`. `http://arxiv.org/abs/1507.00163`, 2015.

14 J. Colvin, M. I. Monine, J. R. Faeder, W. S. Hlavacek, D. D. Von Hoff, and R. G. Posner. Simulation of large-scale rule-based models. *Bioinformatics*, 25(7):910–917, 2009.

15 J. Colvin, M. I. Monine, R. N. Gutenkunst, W. S. Hlavacek, D. D. Von Hoff, and R. G. Posner. Rulemonkey: software for stochastic simulation of rule-based models. *BMC Bioinformatics*, 11:404, 2010.

16 H. Conzelmann, J. Saez-Rodriguez, T. Sauter, B. Kholodenko, and E. Gilles. A domain-oriented approach to the reduction of combinatorial complexity in signal transduction networks. *BMC Bioinformatics*, 7(1):34, 2006.

17 V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine. Abstracting the differential semantics of rule-based models: Exact and automated model reduction. In *LICS*, pages 362–381, 2010.

18 V. Danos and C. Laneve. Formal molecular biology. *TCS*, 325(1):69–110, 2004.

19 R. De Nicola, U. Montanari, and F. Vaandrager. Back and forth bisimulations. In *CONCUR*, volume 458 of *LNCS*, pages 152–165. Springer, 1990.

20 J. R. Faeder, W. S. Hlavacek, I. Reischl, M. L. Blinov, H. Metzger, A. Redondo, C. Wofsy, and B. Goldstein. Investigation of early events in FcεRI-mediated signaling using a detailed mathematical model. *The Journal of Immunology*, 170(7):3769–3781, 2003.

21 J. Feret, T. Henzinger, H. Koeppl, and T. Petrov. Lumpability abstractions of rule-based systems. *TCS*, 431:137–164, 2012.

22 Jerome Feret, Heinz Koeppl, and Tatjana Petrov. Stochastic fragments: A framework for the exact reduction of the stochastic semantics of rule-based models. *International Journal of Software and Informatics*, 7(4):527–604, 2013.

23 J. Fisher and T.A. Henzinger. Executable cell biology. *Nature Biotechnology*, 25(11):1239–1249, 2007. See also correspondence in Nature Biotechnology 26(7):737-8;738-9, 2008.

24 D. Gillespie. The chemical Langevin equation. *The Journal of Chemical Physics*, 113(1):297–306, 2000.

25 J. Heath, M. Kwiatkowska, G. Norman, D. Parker, and O. Tymchyshyn. Probabilistic model checking of complex biological pathways. *TCS*, 391(3):239–257, 2008.

26 H. Hermanns and M. Rettelbach. Syntax, semantics, equivalences, and axioms for MTIPP. In *Proceedings of Process Algebra and Probabilistic Methods*, pages 71–87, Erlangen, 1994.

27 J. Hillston. *A Compositional Approach to Performance Modelling*. CUP, 1996.

28 P. Kocieniewski, J. R. Faeder, and T. Lipniacki. The interplay of double phosphorylation and scaffolding in MAPK pathways. *Journal of Theoretical Biology*, 295:116–124, 2012.

29 K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

30 M. S. Okino and M. L. Mavrovouniotis. Simplification of mathematical models of chemical reaction systems. *Chemical Reviews*, 2(98):391–408, 1998.

31 R. Paige and R. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

32 S. W. Shin, C. Thachuk, and E. Winfree. Verifying chemical reaction network implementations: A pathway decomposition approach. In *VEMPD*, Vienna Summer of Logic, 2014.

33 M. W. Sneddon, J. R. Faeder, and T. Emonet. Efficient modeling, simulation and coarse-graining of biological complexity with NFsim. *Nature Methods*, 8(2):177–183, 2011.

34 J. Sproston and S. Donatelli. Backward Bisimulation in Markov Chain Model Checking. *IEEE Trans. Software Eng.*, 32(8):531–546, 2006.

35 R. Suderman and E. J. Deeds. Machines vs. ensembles: Effective MAPK signaling through heterogeneous sets of protein complexes. *PLoS Comput Biol*, 9(10):e1003278, 10 2013.

36 J. Toth, G. Li, H. Rabitz, and A. S. Tomlin. The effect of lumping and expanding on kinetic differential equations. *SIAM Journal on Applied Mathematics*, 57(6):1531–1556, 1997.

37 M. Tschaikowski and M. Tribastone. Exact fluid lumpability for Markovian process algebra. In *CONCUR*, LNCS, pages 380–394, 2012.

38 M. Tschaikowski and M. Tribastone. A unified framework for differential aggregations in Markovian process algebra. *JLAMP*, 84(2):238–258, 2015.

Lax Bialgebras and Up-To Techniques for Weak Bisimulations

Filippo Bonchi[1], Daniela Petrişan[2], Damien Pous[1], and Jurriaan Rot[3]

1 **LIP, CNRS, ENS Lyon, Université de Lyon, UMR 5668, France**
2 **Radboud University, The Netherlands**
3 **LIACS – Leiden University, CWI, The Netherlands**

Abstract

Up-to techniques are useful tools for optimising proofs of behavioural equivalence of processes. Bisimulations up-to context can be safely used in any language specified by GSOS rules. We showed this result in a previous paper by exploiting the well-known observation by Turi and Plotkin that such languages form *bialgebras*. In this paper, we prove the soundness of up-to contextual closure for weak bisimulations of systems specified by *cool rule formats*, as defined by Bloom to ensure congruence of weak bisimilarity. However, the weak transition systems obtained from such cool rules give rise to *lax bialgebras*, rather than to bialgebras. Hence, to reach our goal, we extend our previously developed categorical framework to an ordered setting.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases up-to techniques, weak bisimulation, (lax) bialgebras

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.240

1 Introduction

Bisimilarity ($\sim$) is a fundamental equivalence for concurrent systems: two processes are (strongly) bisimilar if they cannot be distinguished by an external observer interacting with them. Formally, it is defined by coinduction, as the greatest fixpoint of a suitable predicate transformer B – a monotone function on binary relations. In particular, to prove processes bisimilar, it suffices to exhibit a *bisimulation* relating them, i.e., a relation R such that $R \subseteq B(R)$; this latter requirement codes for the standard game where the processes must answer to the labelled transitions of each other.

Up-to techniques are enhancements of this coinduction principle; they were introduced by Milner to simplify behavioural equivalence proofs of CCS processes, see [16]. The range of applicability of up-to techniques goes well beyond concurrency theory: they have been used to obtain decidability results [8], to optimise state-of-the-art automata algorithms [7] or in conjuction with parameterized coinduction for mechanizations of coinductive proofs [13].

An up-to technique is a monotone map A on the poset of relations, and a *bisimulation up-to* A is a relation R such that $R \subseteq B(A(R))$. If A is *sound*, that is, if every bisimulation up-to A is included in a bisimulation, one can prove bisimilarity results by exhibiting bisimulations up to A. This may be computationally less expensive than finding actual bisimulations. Typical examples for (strong) bisimilarity include *up-to bisimilarity*, where A is given by $A(R) = \sim R \sim$, and *up-to transitive closure*, where $A(R)$ is the least transitive relation containing R. When the systems at hand are specified in some process algebra, via an algebraic signature, a third example is *up-to context* – where one maps a relation to its

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 240–253

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

closure with respect to the contexts of the language. Such a technique is sound in CCS, for instance.

In practice, one is often interested in *weak bisimilarity*, a coarser notion allowing to abstract over internal transitions, labelled with the special action τ. When the player proposes a transition $\xrightarrow{a}$, the opponent must answer with a *saturated* transition $\stackrel{a}{\Rightarrow}$, which is roughly a transition $\xrightarrow{a}$ possibly combined with internal actions $\xrightarrow{\tau}$. This slight dissymmetry results in a much more delicate theory of up-to techniques. For instance, up-to weak bisimilarity and up-to transitive closure are no longer sound for weak bisimulations. And up-to contextual closure has to be restricted: the external choice from CCS cannot be freely used [20].

Simpler properties also become harder with weak bisimilarity. Consider structural operational semantics [1]: if the semantics of a language is specified by rules adhering to certain formats, then certain well-behavedness properties are automatically inferred. For instance, in languages specified using De-Simone or GSOS rule formats [9, 5], strong bisimilarity is guaranteed to be a congruence. However, those two formats do not ensure congruence of weak bisimilarity, and more advanced formats had to be designed to this end, like Bloom's *cool GSOS format* [4].

Proving soundness of up-to techniques can be rather complicated. To simplify this task, Sangiorgi and Pous devised the stronger notion of *compatible* up-to techniques, which are always sound and, moreover, closed under composition. Proving compatibility of a composite technique can thus be broken into simpler, independent proofs [18, 19]. We recently generalised this framework to a fibrational setting [6], allowing to obtain once and for all the compatibility of a wide range of techniques for strong bisimulation and simulation, for systems modelled as *bialgebras*.

Concerning weak bisimilarity, we proved in [6] that for positive GSOS specification, if the strong $\rightarrow$ and saturated $\Rightarrow$ transition systems form bialgebras, then up-to context is a compatible technique. Unfortunately, the bearing of this result in practical situations is rather limited, since in many important cases the saturated transition system does not form a bialgebra. Intuitively, in a bialgebra *all and only* the transitions of a composite system can be derived by transitions of its components. For $\Rightarrow$, one implication fails: a composite system performs weak transitions which are not derived from transitions of its components (see Example 2). These systems give rise to so called *lax bialgebras*; this is the key observation that lead to the rather involved refinement of the theory we propose here.

Contributions. In this paper: a) We extend the previously developed framework [6] to an ordered setting, b) we prove that *up-to context* is compatible for lax models of positive [1] GSOS specifications, and, c) as an application, we obtain soundness of up-to context for weak bisimulations of systems specified by the *cool rule format* from [23].

Outline. We give some necessary preliminaries in Section 2. Then we move to an ordered setting in Section 3, where we use lax bialgebras. In Section 4 we consider the special case of lax bialgebras stemming from (lax models of) positive GSOS specifications. We finally assemble in Section 5 all the technical pieces into our main result, Theorem 20.

2 Preliminaries

2.1 Transitions systems and bisimulations

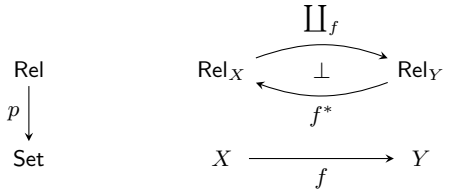
A labelled transition system (LTS) with labels in L consists of a set of states X and a transition function $\xi\colon X \to (\mathcal{P}_\omega X)^L$ that, for every state $x \in X$ and label $a \in L$, assigns a

finite set of possible successor states. We write $x \xrightarrow{a} y$ whenever $y \in \xi(x)(a)$. A *(strong) bisimulation* is a relation $R \subseteq X^2$ on the states of an LTS such that for every pair $(x, y) \in R$: (1) if $x \xrightarrow{a} x'$ then $y \xrightarrow{a} y'$ for some y' with $(x', y') \in R$, and (2) vice versa. A *weak bisimulation* is a relation $R \subseteq X^2$ such that for every pair $(x, y) \in R$: (1) if $x \xrightarrow{a} x'$ then $y \overset{a}{\Rightarrow} y'$ for some y' with $(x', y') \in R$ and (2) if $y \xrightarrow{a} y'$ then $x \overset{a}{\Rightarrow} x'$ for some x' with $(x', y') \in R$. Here $\Rightarrow$ is the *saturation* [16] of $\rightarrow$, defined by the following rules where τ denotes a special label in L.

$$\frac{x \xrightarrow{a} y}{x \overset{a}{\Rightarrow} y} \qquad \frac{}{x \overset{\tau}{\Rightarrow} x} \qquad \frac{x \overset{\tau}{\Rightarrow} y \overset{\tau}{\Rightarrow} z}{x \overset{\tau}{\Rightarrow} z} \qquad \frac{x \overset{\tau}{\Rightarrow} x' \overset{a}{\Rightarrow} y' \overset{\tau}{\Rightarrow} y}{x \overset{a}{\Rightarrow} y} \tag{1}$$

Transition systems are an instance of the abstract notion of *coalgebras*: given a functor $F\colon C \to C$ on some category C, an F-coalgebra is a pair (X, ξ) where X is an object and $\xi\colon X \to FX$ a morphism. Indeed, LTSs are coalgebras for the functor $(\mathcal{P}_\omega -)^L\colon \mathsf{Set} \to \mathsf{Set}$.

Next, we recall the basic infrastructure of relations that allows us to study both strong and weak bisimulations within a coalgebraic setting. Consider the category Rel whose objects are relations $R \subseteq X^2$ and morphisms from $R \subseteq X^2$ to $S \subseteq Y^2$ are maps from X to Y sending pairs in R to pairs in S. For each set X we denote by Rel_X the category (which in this case is just a preorder) of binary relations on X, ordered by subset inclusion. For a function $f\colon X \to Y$ in Set, we have the following situation in Rel:



where p maps a relation $R \subseteq X^2$ to X, and the functors (monotone maps) f^* and $\coprod_f$, which we will call *reindexing* and *direct image*, are given by inverse and direct image, respectively: $f^*(S) = (f \times f)^{-1}(S)$ for all $S \in \mathsf{Rel}_Y$ and $\coprod_f(R) = (f \times f)[R]$ for all $R \in \mathsf{Rel}_X$. Moreover we have that $\coprod_f$ is a left adjoint for f^*.[1]

A functor $\overline{F}\colon \mathsf{Rel} \to \mathsf{Rel}$ is a *lifting* of $F\colon \mathsf{Set} \to \mathsf{Set}$ whenever $p \circ \overline{F} = F \circ p$; this means that $\overline{F}$ maps a relation on X to a relation on FX. Any lifting $\overline{F}$ can thus be restricted to a functor $\overline{F}_X\colon \mathsf{Rel}_X \to \mathsf{Rel}_{FX}$, which is just a monotone function between posets. For every functor $F\colon \mathsf{Set} \to \mathsf{Set}$, there is a *canonical lifting* denoted by $\mathsf{Rel}(F)\colon \mathsf{Rel} \to \mathsf{Rel}$. In this paper the canonical lifting will play an important role but, for the sake of simplicity, we avoid giving the general definition and refer the interested reader to [14]. As an example, the canonical lifting of $(\mathcal{P}_\omega -)^L$ is defined for all relations $R \subseteq X^2$, and $f, g \in (\mathcal{P}_\omega X)^L$ as

$$f\ \mathsf{Rel}((\mathcal{P}_\omega -)^L)(R)\ g \quad \text{iff} \quad \begin{array}{l} \forall a \in L. \forall x \in f(a).\ \exists y \in g(a). xRy \\ \forall a \in L. \forall y \in g(a).\ \exists x \in f(a). xRy \end{array} \tag{2}$$

We can now define bisimulations for any Set-functor F in terms of its canonical lifting. For an F-coalgebra (X, ξ), a (Hermida-Jacobs) bisimulation [11] is a coalgebra for the functor

$$\mathsf{Rel}(F)_\xi \triangleq \xi^* \circ \mathsf{Rel}(F)_X\colon \mathsf{Rel}_X \to \mathsf{Rel}_X\,.$$

[1] The categorically minded reader may observe that the forgetful functor $p\colon \mathsf{Rel} \to \mathsf{Set}$ is a bifibration, but the concrete definitions given above suffice for understanding the forthcoming technical developments.

The functor $\mathsf{Rel}(F)_\xi$ is also called a predicate transformer. Bisimilarity is defined as the largest bisimulation or, in other terms, as the final $\mathsf{Rel}(F)_\xi$-coalgebra. Since morphisms in Rel_X are just inclusions, a coalgebra for $\mathsf{Rel}(F)_\xi$ is a relation R such that $R \subseteq \mathsf{Rel}(F)_\xi(R)$ and, with (2), it is easy to check that for $FX = (\mathcal{P}_\omega X)^L$ this corresponds to the usual definition of strong bisimulations on transition systems.

The above notion can be further generalised by taking an arbitrary lifting $\overline{F}\colon \mathsf{Rel} \to \mathsf{Rel}$ of F: an *$\overline{F}_\xi$-bisimulation* then is a coalgebra for the endofunctor

$$\overline{F}_\xi \triangleq \xi^* \circ \overline{F}_X \colon \mathsf{Rel}_X \to \mathsf{Rel}_X \,.$$

With this more abstract approach, we can capture various interesting coinductive predicates other than strong bisimilarity, such as simulations [12] and weak bisimulations. Indeed, weak bisimulations are coalgebras for the functor $\overline{F \times F}_\xi \colon \mathsf{Rel}_X \to \mathsf{Rel}_X$ where $F = (\mathcal{P}_\omega -)^L$, $\xi = \langle \to, \Rightarrow \rangle \colon X \to FX \times FX$ is the pairing of the strong transition system $\to$ and its saturation $\Rightarrow$, and the functor $\overline{F \times F}$ is the lifting of $F \times F$ to Rel given for a relation R by

$$(f,g)\ \overline{F \times F}(R)\ (f',g') \quad \text{iff} \quad \begin{array}{l} \forall a \in L. \forall x \in f(a).\ \exists y \in g'(a). xRy \\ \forall a \in L. \forall x \in f'(a).\ \exists y \in g(a). xRy \end{array} \tag{3}$$

2.2 Up-To techniques and Compatible functors

In the previous section we have seen how bisimulations can be regarded as coalgebras (post-fixpoints) for a functor (monotone map) $\overline{F}_\xi \colon \mathsf{Rel}_X \to \mathsf{Rel}_X$. In this perspective, an up-to technique is a functor $A\colon \mathsf{Rel}_X \to \mathsf{Rel}_X$ and an *$\overline{F}_\xi$-bisimulation up to A* is an $\overline{F}_\xi A$-coalgebra. For instance, a bisimulation up to equivalence is a $\mathsf{Rel}(F)_\xi E$-coalgebra, where $E\colon \mathsf{Rel}_X \to \mathsf{Rel}_X$ is the functor mapping a relation to its equivalence closure.

We say that A is *$\overline{F}_\xi$-compatible* if there exists a distributive law (natural transformation) $\rho\colon A\overline{F}_\xi \Rightarrow \overline{F}_\xi A$. If A is $\overline{F}_\xi$-compatible then A is a sound up-to technique: every $\overline{F}_\xi$-bisimulation up-to A is included in an $\overline{F}_\xi$-bisimulation. This is stated in [19, Theorem 6.3.9] for the case of lattices, but it holds more generally in any category with countable coproducts and, rather than considering just endofunctors F on Set and their liftings $\overline{F}$ to Rel, one can take endofunctors and liftings in arbitrary fibrations [6]. For the sake of simplicity we will avoid using fibrations: the reader should only know that the above result holds also for Pre-endofunctors and their liftings to the category $\mathsf{Rel}^\uparrow$ which we introduce in Section 3.

By tuning F, $\overline{F}$ and A one can consider different sorts of, respectively, state-based systems (such as LTSs, deterministic or weighted automata), coinductive predicates (such as bisimilarity, similarity or language equivalence) and up-to techniques (such as up-to transitivity, up-to equivalence, up-to bisimilarity). In [6], we provided several techniques for proving the compatibility of particular techniques. For up-to context, the state space of the coalgebra needs to have some algebraic structure, for instance, the LTSs of process algebras. This is captured systematically by bialgebras: given functors $F, T\colon \mathsf{Set} \to \mathsf{Set}$ and a distributive law $\rho\colon TF \Rightarrow FT$, a *$\rho$-bialgebra* consists of a set X, an algebra $\alpha\colon TX \to X$ and a coalgebra $\xi\colon X \to FX$ such that the following diagram commutes.

$$\begin{array}{ccccc} TX & \xrightarrow{\alpha} & X & \xrightarrow{\xi} & FX \\ {\scriptstyle T\xi}\downarrow & & & & \uparrow{\scriptstyle F\alpha} \\ TFX & & \xrightarrow[\rho_X]{} & & FTX \end{array}$$

The function mapping a relation on X to its contextual closure can be obtained as

$$Ctx \triangleq \coprod_\alpha \circ \mathsf{Rel}(T)_X \colon \mathsf{Rel}_X \to \mathsf{Rel}_X \,.$$

To prove the compatibility of *Ctx* w.r.t. different $\overline{F}_\xi$, we showed the theorem below, where we adopt the following terminology: a natural transformation $\overline{\sigma}\colon \overline{F} \Rightarrow \overline{G}$ between Rel-functors is a *lifting* of $\sigma\colon F \Rightarrow G$ when for every $R \in \mathsf{Rel}$ we have that $p(\overline{\sigma}_R) = \sigma_{p(R)}$.

▶ **Theorem 1** (see [6]). *Let (X, α, ξ) be a ρ-bialgebra and $\overline{T}, \overline{F}\colon \mathcal{E} \to \mathcal{E}$ be liftings of T and F. If $\overline{\rho}\colon \overline{T}\,\overline{F} \Rightarrow \overline{F}\,\overline{T}$ is a lifting of ρ, then $\coprod_\alpha \circ \overline{T}$ is $\overline{F}_\xi$-compatible.*

2.3 Abstract GSOS specifications and their models

Abstract GSOS specifications are natural transformations of the form $\lambda\colon S(F \times \mathrm{Id}) \Rightarrow FT$, where T is the free monad over S. As shown in [22], they generalise the concrete GSOS rules, for which $FX = (\mathcal{P}_\omega X)^L$, and S is a polynomial functor – a coproduct of products – representing an algebraic signature, and hence TX is the set of terms over this signature with variables in X. A *model* of a specification λ is a triple (X, α, ξ), where $\xi\colon X \to FX$ and $\alpha\colon SX \to X$, making the following diagram commute:

$$\begin{array}{ccccc} SX & \xrightarrow{\alpha} & X & \xrightarrow{\xi} & FX \\ {\scriptstyle S\langle \xi, \mathrm{id}\rangle}\downarrow & & & & \uparrow{\scriptstyle F\alpha^\sharp} \\ S(FX \times X) & & \xrightarrow[\lambda_X]{} & & FTX \end{array} \tag{4}$$

▶ **Example 2.** Consider the parallel operator of CCS [16], whose semantics is defined by the following GSOS rules

$$\frac{p \xrightarrow{\mu} p'}{p|q \xrightarrow{\mu} p'|q} \qquad \frac{q \xrightarrow{\mu} q'}{p|q \xrightarrow{\mu} p|q'} \qquad \frac{p \xrightarrow{a} p' \quad q \xrightarrow{\overline{a}} q'}{p|q \xrightarrow{\tau} p'|q'}$$

where μ ranges over arbitrary actions, namely inputs $a, b, \ldots$ outputs $\overline{a}, \overline{b}, \ldots$ or the internal action τ. Take $SX = X \times X$ (for the binary parallel operator) and $F = (\mathcal{P}_\omega -)^L$ where L is the set of all actions. For every set X, the corresponding distributive law $\lambda_X\colon S(FX \times X) \to FTX$ maps $(f, x, g, y) \in (\mathcal{P}_\omega X)^L \times X \times (\mathcal{P}_\omega X)^L \times X$ to the function

$$\mu \mapsto \begin{cases} \{(x', y) \mid x' \in f(\mu)\} \cup \{(x, y') \mid y' \in g(\mu)\} & \mu \neq \tau \\ \{(x', y) \mid x' \in f(\tau)\} \cup \{(x, y') \mid y' \in g(\tau)\} \cup \{(x', y') \mid \exists a.\, x' \in f(a), y' \in g(\overline{a})\} & \mu = \tau \end{cases}$$

Now take X to be the set of *all* CCS processes, $\xi\colon X \to (\mathcal{P}_\omega X)^L$ the LTS generated by the standard semantics of CCS [16] and $\alpha\colon X \times X \to X$ to be the algebra mapping a pair of processes (p, q) to their parallel composition $p|q$. It is easy to see that diagram (4) commutes, i.e., (X, α, ξ) is a model for λ.

On the contrary, if we take ξ to be the saturation of the standard CCS semantics, diagram (4) does not commute anymore: take the pairs of CCS processes $(a.b.0, \overline{a}.\overline{b}.0) \in SX$. Following the topmost line, one first maps it to $a.b.0|\overline{a}.\overline{b}.0$ in the weak LTS that, for instance, contains the transition $\stackrel{\tau}{\Rightarrow} 0|0$. Following the other path in the diagram one obtains first the tuple $(((a \mapsto \{b.0\}), a.b.0),\ ((\overline{a} \mapsto \{\overline{b}.0\}), \overline{a}.\overline{b}.0))$ where $\mu \mapsto S$ denotes the function assigning to the action μ the set S and to all the others actions the empty set. This tuple is mapped by λ_X to the function

$$a \mapsto \{(b.0, \overline{a}.\overline{b}.0)\} \qquad \overline{a} \mapsto \{(a.b.0, \overline{b}.0)\} \qquad \tau \mapsto \{(b.0, \overline{b}.0)\}$$

and then by $F\alpha^\sharp$ to

$$a \mapsto \{b.0|\overline{a}.\overline{b}.0\} \qquad \overline{a} \mapsto \{a.b.0|\overline{b}.0\} \qquad \tau \mapsto \{b.0|\overline{b}.0\}$$

Observe that with τ, one cannot reach the state $0|0$.

An abstract GSOS specification λ and a model (X, α, ξ) for it induce respectively a distributive law $\rho\colon T(F \times \mathrm{Id}) \Rightarrow (F \times \mathrm{Id})T$ of the monad T over the copointed functor $F \times \mathrm{Id}$ and a bialgebra $(X, \alpha^\sharp, \langle \xi, \mathrm{id}\rangle)$ for ρ [22, 15]. Using these facts and the characterization of weak bisimulations given in (3) we were able to prove the following result.

▶ **Proposition 3** (see [6]). *Let $FX = (\mathcal{P}_\omega X)^L$, and let $\lambda\colon S(F \times \mathrm{Id}) \Rightarrow FT$ be a GSOS specification with two models (X, α, ξ_1) and (X, α, ξ_2). If λ is positive (see, e.g., [1]) then:*

1. *There exists a distributive law $\rho\colon T(F \times F \times \mathrm{Id}) \Rightarrow (F \times F \times \mathrm{Id})T$ s.t. $(X, \alpha^\sharp, \langle \xi_1, \xi_2, \mathrm{id}\rangle)$ is a ρ-bialgebra.*
2. *There exists $\overline{\rho}\colon \mathsf{Rel}(T)(\overline{F \times F} \times \mathrm{Id}) \Rightarrow (\overline{F \times F} \times \mathrm{Id})\mathsf{Rel}(T)$ lifting ρ, where $\overline{F \times F}$ is defined as in (3).*
3. *By the previous points and Theorem 1, $Ctx = \coprod_{\alpha^\sharp} \circ \mathsf{Rel}(T)_X$ is $(\overline{F \times F} \times \mathrm{Id})_{\langle \xi_1, \xi_2, \mathrm{id}\rangle}$-compatible.*

The above result ensures compatibility w.r.t. $(\overline{F \times F} \times \mathrm{Id})_{\langle \xi_1, \xi_2, \mathrm{id}\rangle}$, which is not exactly $\overline{F \times F}_{\langle \xi_1, \xi_2\rangle}$. As discussed in [6], weak bisimulations are coalgebras for either of these two predicate transformers. The extra Id is harmless for the above result and for Theorem 20.

Proposition 3 gives us compatibility of up-to *Ctx* for weak bisimulation whenever ξ_2, given by the saturation of ξ_1, is a model for the GSOS specification. However, Example 2 shows that already for the simple case of parallel composition in CCS, ξ_2 is *not* a model for the GSOS specification. This motivates the need for relaxing the hypothesis of Proposition 3: in the rest of the paper, we will introduce the notions of lax bialgebras and lax models and we will show the analogues of Theorem 1 and Proposition 3 in an ordered setting.

3 Bialgebras and compatibility in an ordered setting

We recalled how to prove soundness of up-to techniques in a modular way, by considering lifting functors and distributive laws along $p\colon \mathsf{Rel} \to \mathsf{Set}$. Now we extend those results to an ordered setting. The first step (Section 3.1) consists in replacing the base category Set with Pre, the category of preorders. (An object in Pre is a set equipped with a preorder, that is, a reflexive and transitive relation; morphisms are monotone maps.) Accordingly, we move from the category Rel of relations to its subcategory $\mathsf{Rel}^\uparrow$ of up-closed relations (Section 3.2). We finally obtain the ordered counterpart to Theorem 1, using the notion of lax bialgebra (Section 3.3, Theorem 15).

3.1 Lifting functors from sets to preorders

We first explain how to lift functors and distributive laws from Set to Pre. Extensions of Set-functors to preorders or posets have been studied via relators as in [12, 21] and using presentations of functors and (enriched) Kan extensions [2, 3]. We are interested in extending not only functors, but also natural transformations to an ordered setting. The description using (lax) relation liftings [12] allows us to leverage some of our results in [6] to extend natural transformations.

For a weak pullback preserving Set-endofunctor T we can consider its canonical relation lifting $\mathsf{Rel}(T)\colon \mathsf{Rel} \to \mathsf{Rel}$. Then, using the following well-known result, we obtain an extension of T to Pre, hereafter called *the canonical* Pre*-lifting of* T and denoted by $\mathsf{Pre}(T)$.

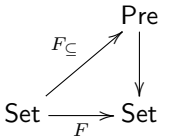
▶ **Lemma 4.** *If T preserves weak pullbacks then $\mathsf{Rel}(T)$ restricts to a functor $\mathsf{Pre}(T)$ on Pre.*

However, sometimes we are interested in liftings of functors to Pre that are not restrictions of the canonical relation lifting. One such example is the lifting of the LTS functor $(\mathcal{P}_\omega -)^L$

to Pre that maps a preordered set $(X, \leq)$ to $((\mathcal{P}_\omega X)^L, \sqsubseteq)$, where $\sqsubseteq$ is given by

$$f \sqsubseteq g \text{ iff } \forall a \in L : \text{ if } x \in f(a) \text{ then there is } y \in g(a) \text{ such that } x \leq y\,. \tag{5}$$

This lifting is also a restriction to Pre of relation lifting for $(\mathcal{P}_\omega -)^L$, albeit not the canonical one, but the *lax relation lifting*, as defined in [12]. To describe it, recall from [12] that a Set-functor F is called *ordered* when it factors through a functor $F_\subseteq \colon \mathsf{Set} \to \mathsf{Pre}$.



We denote by $\subseteq_{FX}$ the order on FX given by $F_\subseteq(X)$. The lax relation lifting of F is defined as the functor $\mathsf{Rel}_\subseteq(F) \colon \mathsf{Rel} \to \mathsf{Rel}$ that maps a relation R on X to $\subseteq_{FX} \otimes \mathsf{Rel}(F)(R) \otimes \subseteq_{FX}$, where $\otimes$ denotes composition of relations. In [12, Lemma 5.5] it is shown that $\mathsf{Rel}_\subseteq(F)$ restricts to a functor $\mathsf{Pre}_\subseteq(F)$ on Pre, if the order $\subseteq_{FX}$ has an additional property, namely it is stable, see [12, Definition 4.3]. This property is duly satisfied by all the ordered functors considered in this paper. We call the restriction of $\mathsf{Rel}_\subseteq(F)$ to Pre the *lax* Pre*-lifting of* F and denote it by $\mathsf{Pre}_\subseteq(F)$.

▶ **Example 5** (see [12])**.** The LTS functor $(\mathcal{P}_\omega -)^L$ has a stable order $\subseteq_{(\mathcal{P}_\omega X)^L}$ given by pointwise inclusion. The lax Pre-lifting of $(\mathcal{P}_\omega -)^L$ with respect to this order coincides with the lifting described above in (5).

We now show how to lift a natural transformation $\rho \colon F \Rightarrow G$ between Set-functors to a natural transformation $\varrho \colon \mathcal{F} \Rightarrow \mathcal{G}$ between Pre-functors. If F and G preserve weak pullbacks and $\mathcal{F}$ and $\mathcal{G}$ are the canonical Pre-extensions $\mathsf{Pre}(F)$ and $\mathsf{Pre}(G)$, then ϱ is obtained via the restriction of the natural transformation $\mathsf{Rel}(\rho)$ between the corresponding canonical relation liftings ($\mathsf{Rel}(-)$ is functorial, see [14]). The situation is slightly more complex for non-canonical liftings, such as the lax lifting of the LTS functor. In this case we can use Lemma 7 below whenever ρ enjoys the following monotonicity property.

▶ **Definition 6.** Let $F, G \colon \mathsf{Set} \to \mathsf{Set}$ be ordered functors that factor through $F_\subseteq, G_\subseteq \colon \mathsf{Set} \to \mathsf{Pre}$ respectively. We say that a natural transformation $\rho \colon F \Rightarrow G$ is *monotone* if it lifts to a natural transformation $\varrho \colon F_\subseteq \Rightarrow G_\subseteq$ defined by $\varrho_X = \rho_X$.

Spelling out Definition 6 we obtain that ρ is monotone iff for every $t, u \in FX$:

$$t \subseteq_{FX} u \quad \text{implies} \quad \rho(t) \subseteq_{GX} \rho(u)$$

where $\subseteq_{FX}$ and $\subseteq_{GX}$ denote the orders on FX and GX given by $F_\subseteq$ and $G_\subseteq$ respectively.

▶ **Lemma 7.** *Let $F, G \colon \mathsf{Set} \to \mathsf{Set}$ be ordered functors with orders given by $F_\subseteq, G_\subseteq \colon \mathsf{Set} \to \mathsf{Pre}$ respectively, and assume $\rho \colon F \Rightarrow G$ is a monotone natural transformation. Then ρ lifts to a natural transformation $\overline{\rho} \colon \mathsf{Rel}_\subseteq(F) \Rightarrow \mathsf{Rel}_\subseteq(G)$. Furthermore, if the lax relation liftings of F and G restrict to Pre-endofunctors $\mathsf{Pre}_\subseteq(F)$ and $\mathsf{Pre}_\subseteq(G)$ then ρ lifts to a natural transformation $\varrho \colon \mathsf{Pre}_\subseteq(F) \Rightarrow \mathsf{Pre}_\subseteq(G)$.*

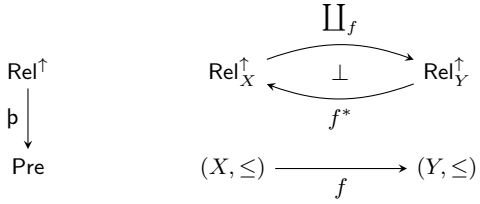
3.2 Relation liftings for Pre-endofunctors

In the previous section we have seen how to extend Set functors, such as those involved in GSOS specifications, to preorders. To reason about relation liftings in this setting we ought to consider a category of relations with a forgetful functor to Pre. On a preorder $(X, \leq)$ we consider relations that are *up-closed* with respect to $\leq$, as defined next.

▶ **Definition 8.** Given a preorder $(X, \leq)$ we define an *up-closed relation* on X as a relation $R \subseteq X^2$ such that for every $x', x, y, y' \in X$ with $x \leq x'$, $y \leq y'$ and xRy we have that $x'Ry'$. A morphism between up-closed relations R and S on $(X, \leq)$, respectively $(Y, \leq)$, is a monotone map $f\colon (X, \leq) \to (Y, \leq)$ such that $R \subseteq (f \times f)^{-1}(S)$.

We denote by $\mathsf{Rel}^{\uparrow}$ the category of up-closed relations. We have an obvious forgetful functor $\flat\colon \mathsf{Rel}^{\uparrow} \to \mathsf{Pre}$ mapping every up-closed relation to its underlying preorder. For each preorder $(X, \leq)$ we denote by $\mathsf{Rel}^{\uparrow}_X$ the subcategory of $\mathsf{Rel}^{\uparrow}$ whose objects are mapped by $\flat$ to $(X, \leq)$ and morphisms are mapped by $\flat$ to the identity on $(X, \leq)$. Notice that $\mathsf{Rel}^{\uparrow}_X$ is a category, with morphisms given by inclusions of relations, hence, a preorder.

For a monotone map $f\colon (X, \leq) \to (Y, \leq)$ in Pre, we have the following situation in $\mathsf{Rel}^{\uparrow}$, similar to the situation described for Rel in Section 2:



Here, the *reindexing* functor f^* is given by inverse image, i.e., $f^*(S) = (f \times f)^{-1}(S)$ for all $S \in \mathsf{Rel}^{\uparrow}_Y$ while the *direct image* functor $\coprod_f$ is defined on a up-closed relation $R \in \mathsf{Rel}^{\uparrow}_X$ as the least up-closed relation containing $(f \times f)[R]$. Just as in the case of Rel, the functor $\coprod_f$ is a left adjoint of f^*, and $\flat\colon \mathsf{Rel}^{\uparrow} \to \mathsf{Pre}$ is a bifibration. Observe that if the preorder on Y is discrete, then $\coprod_f$ is given simply by direct image.

▶ **Remark 9.** *For every discrete preorder (X, Δ_X), any relation on X is automatically up-closed. We can reformulate this in a conceptual way, using that the forgetful functor $U\colon \mathsf{Pre} \to \mathsf{Set}$ has a left adjoint $D\colon \mathsf{Set} \to \mathsf{Pre}$ mapping a set X to the discrete preorder (X, Δ_X). Then the adjunction $D \dashv U$ lifts to an adjunction $\overline{D} \dashv \overline{U} : \mathsf{Rel}^{\uparrow} \to \mathsf{Rel}$.*

Pre has an enriched structure, in the sense that the homsets are equipped with an order themselves. Given morphisms $f, g\colon (X, \leq) \to (Y, \leq)$ we say that $f \leq g$ if $f(x) \leq_Y g(x)$ for every $x \in X$. This order is preserved by the reindexing functors:

▶ **Lemma 10.** *For any Pre-morphisms $f, g\colon (X, \leq) \to (Y, \leq)$ such that $f \leq g$ there exists a (unique) natural transformation $f^* \Rightarrow g^*$.*

We now show how to port liftings of functors from Rel and Pre to $\mathsf{Rel}^{\uparrow}$.

▶ **Lemma 11.** *For a weak pullback preserving Set-functor T, the canonical Pre-lifting $\mathsf{Pre}(T)$ has a lifting $\overline{\mathsf{Pre}(T)}$ to $\mathsf{Rel}^{\uparrow}$ acting on a relation as the canonical relation lifting $\mathsf{Rel}(T)$.*

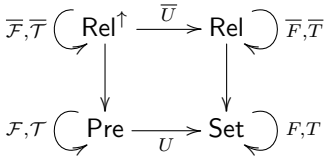
Some of the liftings used in Section 5 to describe weak bisimulations are not canonical, nor lax relation liftings. In Equation (3) we saw how to obtain the weak bisimulation game via a relation lifting $\overline{F \times F}$ of the functor $F \times F$ with $FX = (\mathcal{P}_\omega X)^L$. The next example gives a lifting of $F \times F$ to Pre, such that the relation lifting (3) restricts to up-closed relations, thus yielding a functor on $\mathsf{Rel}^{\uparrow}$ for the weak bisimulation game.

▶ **Example 12.** For $F = (\mathcal{P}_\omega -)^L$ we consider the Pre-endofunctor $\mathsf{Pre}(F) \times \mathsf{Pre}_{\subseteq}(F)$, where $\mathsf{Pre}(F)$ is the canonical Pre-lifting of F and $\mathsf{Pre}_{\subseteq}(F)$ is the lax Pre-lifting of Example 5. In Appendix A, we show that for any preorder $(X, \leq)$ and $R \in \mathsf{Rel}^{\uparrow}_{(X,\leq)}$ we have that $\overline{F \times F}(R)$ as defined in (3) is an up-closed relation on $\mathsf{Pre}(F)(X, \leq) \times \mathsf{Pre}_{\subseteq}(F)(X, \leq)$.

Thus we obtain a lifting $\overline{\mathsf{Pre}(F) \times \mathsf{Pre}_{\subseteq}(F)}$ of $\mathsf{Pre}(F) \times \mathsf{Pre}_{\subseteq}(F)$ to $\mathsf{Rel}^{\uparrow}$ such that $\overline{U}\,\overline{\mathsf{Pre}(F) \times \mathsf{Pre}_{\subseteq}(F)} = (\overline{F \times F})\,\overline{U}$. This means that coalgebras for $\overline{\mathsf{Pre}(F) \times \mathsf{Pre}_{\subseteq}(F)}_{\langle \xi_1, \xi_2 \rangle}$ correspond to weak bisimulations, whenever ξ_2 is the saturation of ξ_1.

In Theorem 20 we will need liftings of natural transformations to $\mathsf{Rel}^{\uparrow}$. We show next how to obtain them leveraging existing liftings to Rel and Pre introduced in Sections 2 and 3.1.

▶ **Lemma 13.** *Consider* Set*-functors* F, T *with respective liftings* $\overline{F}, \overline{T}$ *on* Rel*;* $\mathcal{F}, \mathcal{T}$ *on* Pre*. Assume that* $\mathcal{F}$ *and* $\mathcal{T}$ *lift to* $\overline{\mathcal{F}}$ *and* $\overline{\mathcal{T}}$ *on* $\mathsf{Rel}^{\uparrow}$*, such that* $\overline{U}\overline{\mathcal{T}} = \overline{T}\overline{U}$ *and* $\overline{U}\overline{\mathcal{F}} = \overline{F}\overline{U}$*, as in the diagram*



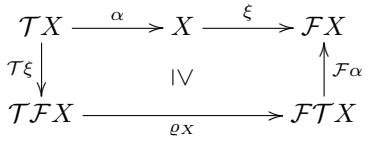
Assume further that we have a natural transformation $\rho\colon TF \Rightarrow FT$ *that lifts to both* $\varrho\colon \mathcal{T}\mathcal{F} \Rightarrow \mathcal{F}\mathcal{T}$ *and* $\overline{\rho}\colon \overline{T}\overline{F} \Rightarrow \overline{F}\overline{T}$*. Then* ϱ *also lifts to a natural transformation* $\overline{\varrho}\colon \overline{\mathcal{T}}\overline{\mathcal{F}} \Rightarrow \overline{\mathcal{F}}\overline{\mathcal{T}}$*.*

In the sequel, we use notations for liftings as in the above lemma: for a functor F, we denote by calligraphic $\mathcal{F}$ a lifting along $\mathsf{Pre} \to \mathsf{Set}$ and by $\overline{\mathcal{F}}$ a lifting of $\mathcal{F}$ along $\mathsf{Rel}^{\uparrow} \to \mathsf{Pre}$; for natural transformations, we use ϱ for a lifting of ρ to Pre and $\overline{\varrho}$ for a lifting of ϱ to $\mathsf{Rel}^{\uparrow}$.

3.3 Lax bialgebras and compatibility of contextual closure

As explained in the Introduction, we moved to an order enriched setting because we want to reason about systems for which the saturated transition system forms a lax bialgebra:

▶ **Definition 14.** Given $\mathcal{T}, \mathcal{F}\colon \mathsf{Pre} \to \mathsf{Pre}$ such that there is a distributive law $\varrho\colon \mathcal{T}\mathcal{F} \Rightarrow \mathcal{F}\mathcal{T}$, a *lax bialgebra* for ϱ consists of a preorder X, an algebra $\alpha\colon \mathcal{T}X \to X$ and a coalgebra $\xi\colon X \to \mathcal{F}X$ such that we have the next lax diagram, with $\leq$ denoting the order on $\mathcal{F}X$.



In this setting, the contextual closure of an up-closed relation is defined by the functor

$$Ctx \triangleq \coprod_{\alpha} \circ \overline{\mathsf{Pre}(T)}_X : \mathsf{Rel}^{\uparrow}_X \to \mathsf{Rel}^{\uparrow}_X$$

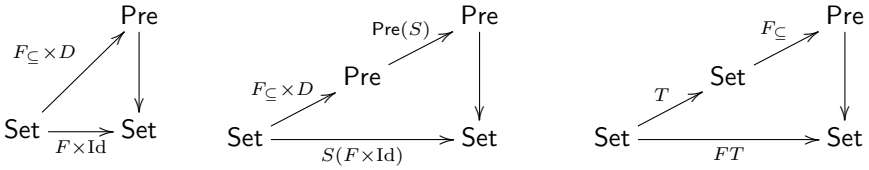
where $\overline{\mathsf{Pre}(T)}$ is the lifting of $\mathsf{Pre}(T)$ to $\mathsf{Rel}^{\uparrow}$ that, by Lemma 11, exists whenever T preserves weak-pullbacks. For any Pre-functor $\mathcal{F}$ and lifting $\overline{\mathcal{F}}$, we can prove $\overline{\mathcal{F}}_{\xi}$-compatibility of up-to Ctx using the following result which extends Theorem 1 to a lax setting.

▶ **Theorem 15.** *Let* $\mathcal{T}, \mathcal{F}$ *be* Pre*-endofunctors with liftings* $\overline{\mathcal{T}}, \overline{\mathcal{F}}$ *to* $\mathsf{Rel}^{\uparrow}$*. Assume that* $\varrho\colon \mathcal{T}\mathcal{F} \Rightarrow \mathcal{F}\mathcal{T}$ *is a natural transformation such that there exists a lifting* $\overline{\varrho}\colon \overline{\mathcal{T}}\overline{\mathcal{F}} \Rightarrow \overline{\mathcal{F}}\overline{\mathcal{T}}$ *of* ϱ*. If* (X, α, ξ) *is a lax* ϱ*-bialgebra, then the functor* $\coprod_{\alpha} \circ \overline{\mathcal{T}}$ *is* $\overline{\mathcal{F}}_{\xi}$*-compatible.*

4 Monotone GSOS in an ordered setting

In this section we describe how to obtain a distributive law in Pre and a lax bialgebra from an abstract GSOS specification in Set and a *lax* model for it. The key property is monotonicity (Definition 6) of the abstract GSOS specification.

Let $\lambda\colon S(F\times \mathrm{Id}) \Rightarrow FT$ be an abstract GSOS specification. Suppose F has a stable order given by a factorization through $F_\subseteq : \mathsf{Set} \to \mathsf{Pre}$ and let $\subseteq_{FX}$ denote the induced order on FX. Then the functors $F\times \mathrm{Id}$, $S(F\times \mathrm{Id})$ and FT are ordered, as follows:



(6)

where $D\colon \mathsf{Set} \to \mathsf{Pre}$ is the functor assigning to a set the discrete order (Remark 9). Recall that $\mathsf{Pre}_\subseteq(F)$ is the lax Pre-lifting of F with respect to the order given by $F_\subseteq$ and consider the canonical Pre-lifting $\mathsf{Pre}(T)$ of the monad T; then the lax Pre-liftings of the functors $F\times \mathrm{Id}$, $S(F\times \mathrm{Id})$ and FT with respect to the orders in (6) are given by $\mathsf{Pre}_\subseteq(F)\times \mathrm{Id}$, $\mathsf{Pre}(S)(\mathsf{Pre}_\subseteq(F)\times \mathrm{Id})$, respectively $\mathsf{Pre}_\subseteq(F)\mathsf{Pre}(T)$.

If the GSOS specification λ is *monotone* with respect to the orders in (6) (recall Definition 6) then, by Lemma 7, λ lifts to $\dot\lambda\colon \mathsf{Pre}(S)(\mathsf{Pre}_\subseteq(F)\times \mathrm{Id}) \Rightarrow \mathsf{Pre}_\subseteq(F)\mathsf{Pre}(T)$.

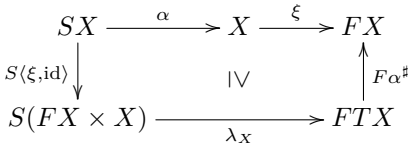
If S is a polynomial functor representing a signature, then λ is monotone if and only if for any operator σ (of arity n) we have

$$\frac{b_1 \subseteq_{FX} c_1 \quad \dots \quad b_n \subseteq_{FX} c_n}{\lambda_X(\sigma(\mathbf{b},\mathbf{x})) \subseteq_{FTX} \lambda_X(\sigma(\mathbf{c},\mathbf{x}))} \tag{7}$$

where $\mathbf{b},\mathbf{x} = (b_1,x_1),\dots,(b_n,x_n)$ with $x_i \in X$ and similarly for $\mathbf{c},\mathbf{x}$. When $F = (\mathcal{P}_\omega -)^L$ with the pointwise inclusion order $\subseteq_{(\mathcal{P}_\omega X)^L}$ from Example 5, then condition (7) corresponds to the *positive GSOS* format [10] which, as expected, is GSOS without negative premises.

▶ **Lemma 16.** *A monotone GSOS specification induces a distributive law $\rho : T(F\times \mathrm{Id}) \Rightarrow (F\times \mathrm{Id})T$ that lifts to a distributive law $\varrho : \mathsf{Pre}(T)(\mathsf{Pre}_\subseteq(F)\times \mathrm{Id}) \Rightarrow (\mathsf{Pre}_\subseteq(F)\times \mathrm{Id})\mathsf{Pre}(T)$.*

▶ **Definition 17.** Let $\lambda\colon S(F\times \mathrm{Id}) \Rightarrow FT$ be a monotone abstract GSOS specification. A lax model for λ is a triple (X,α,ξ) such that the next diagram is lax w.r.t. the order $\subseteq_{FX}$.



(8)

▶ **Example 18.** Consider the GSOS specification λ given in Example 2. Since in the corresponding rules there are no negative premises, it conforms to condition (7), namely it is a positive GSOS specification. Lemma 16 ensures that we have a distributive law $\varrho\colon \mathsf{Pre}(T)(\mathsf{Pre}_\subseteq(F)\times \mathrm{Id}) \Rightarrow (\mathsf{Pre}_\subseteq(F)\times \mathrm{Id})\mathsf{Pre}(T)$.

Recall that ξ_2 is the saturation of the standard semantics of CCS and that (X,α,ξ_2) is not a model for λ, since not *all* the weak transitions of a composite process $p|q$ can be deduced by the ones of the components p and q. However, (X,α,ξ_2) is a lax model. Intuitively, the fact that the inequality (8) holds means that *only* the weak transitions of $p|q$ can be deduced by those of p and q, i.e., $p|q$ contains all the weak transitions that can be deduced from those of p and q and the rules for parallel composition.

By unfolding the definitions of α and $\subseteq_{(\mathcal{P}_\omega X)^L}$, (8) is equivalent to

$$F\alpha^\sharp \lambda_X(\xi_2(p),p,\xi_2(q),q)(\mu) \subseteq \xi_2(p|q)(\mu)$$

for all CCS processes p, q and actions $\mu \in L$. When $\mu = \tau$ (the others cases are simpler) this is equivalent to

$$\{p'|q \mid p \stackrel{\tau}{\Rightarrow} p'\} \cup \{p|q' \mid q \stackrel{\tau}{\Rightarrow} q'\} \cup \{p'|q' \mid p \stackrel{a}{\Rightarrow} p',\, q \stackrel{\overline{a}}{\Rightarrow} q'\} \subseteq \{r \mid p|q \stackrel{\tau}{\Rightarrow} r\} \tag{9}$$

which holds by simple calculations. Notice that (9) means exactly that the weak transition system should be closed w.r.t. the rule of the GSOS specification: whenever $\Rightarrow$ satisfies the premises of a rule, then it should also satisfy its consequences.

For a non-example, consider the GSOS rules for the non-deterministic choice of CCS.

$$\frac{p \stackrel{\mu}{\to} p'}{p + q \stackrel{\mu}{\to} p'} \qquad \frac{q \stackrel{\mu}{\to} q'}{p + q \stackrel{\mu}{\to} q'}$$

This specification is also positive, but the saturated transition system ξ_2 is not a lax model. Intuitively, *not only* the weak transitions of $p + q$ can be deduced by the weak transitions of p and q: indeed from $p \stackrel{\tau}{\Rightarrow} p$ one can infer that $p + q \stackrel{\tau}{\Rightarrow} p$ which is not a transition of $p + q$.

The inclusion (9) in the previous example suggests a more concrete characterization for the validity of (8): every transition that can be derived by instantiating a GSOS rule to the transitions in ξ should be already present in ξ, namely, the transition structure is closed under the application of GSOS rules. In contrast to (strict) models (see (4)), in a lax model the converse does not hold: not all the transitions are derivable from the GSOS rules.

Lax models for a monotone GSOS specification λ induce lax bialgebras for the distributive law ϱ obtained as in Lemma 16.

▶ **Lemma 19.** *Let (X, α, ξ) be a lax model for a monotone specification $\lambda\colon S(F \times \mathrm{Id}) \Rightarrow FT$. Then we have a lax bialgebra in* Pre *for the induced distributive law ϱ carried by (X, Δ_X), i.e., the set X with the discrete order, with the algebra map given by $\alpha^\sharp\colon \mathsf{Pre}(T)X \to X$ and the coalgebra map given by $\langle \xi, \mathrm{id}\rangle\colon X \to \mathsf{Pre}_{\subseteq}(F)X \times X$.*

5 Weak bisimulations up-to context for cool GSOS

We put together the results of Sections 3 and 4 to obtain our main result: if the saturation of a model of a positive GSOS specification is a lax model, then up-to context is compatible for weak bisimulation.

▶ **Theorem 20.** *Let $\lambda\colon S(F \times \mathrm{Id}) \Rightarrow FT$ be a positive GSOS specification. Let ξ_2 be the saturation of an LTS ξ_1. If (X, α, ξ_1) and (X, α, ξ_2) are, respectively, a model and a lax model for λ, then Ctx is $(\overline{\mathsf{Pre}(F)} \times \mathsf{Pre}_{\subseteq}(F) \times \mathrm{Id})_{\langle \xi_1, \xi_2, \mathrm{id}\rangle}$-compatible.*

Proof. We apply Theorem 15. To this end we have to provide the following ingredients:

1. a distributive law ϱ between Pre-endofunctors;
2. a lax bialgebra for ϱ;
3. a lifting $\overline{\varrho}$ of ϱ between $\mathsf{Rel}^{\uparrow}$-liftings of the aforementioned functors.

We will explain each step in turn.

1. From a monotone $\lambda\colon S(F \times \mathrm{Id}) \Rightarrow FT$ we first obtain a natural transformation $\tilde{\lambda}\colon S(F \times F \times \mathrm{Id}) \Rightarrow (F \times F)T$ by pairing the natural transformations $\lambda \circ S\langle \pi_1, \pi_3\rangle\colon S(F \times F \times \mathrm{Id}) \Rightarrow FT$ and $\lambda \circ S\langle \pi_2, \pi_3\rangle\colon S(F \times F \times \mathrm{Id}) \Rightarrow FT$. Let $G\colon \mathsf{Set} \to \mathsf{Set}$ denote the functor $F \times F \times \mathrm{Id}$. From the GSOS specification $\tilde{\lambda}$ we obtain a distributive law $\rho\colon TG \Rightarrow GT$ in Set. Since λ is monotone w.r.t. the order given by $F_{\subseteq}$, we have that $\tilde{\lambda}$ can be seen as a monotone abstract GSOS specification for the functor $F \times F$ with the order $\Delta_{FX} \times \subseteq_{FX}$

on $FX \times FX$ given by the product of the discrete order and the one obtained from $F_\subseteq$. We consider the Pre-lifting $\mathcal{G}$ of G defined as $\mathcal{G} = \mathsf{Pre}_\subseteq(F \times F) \times \mathrm{Id}$ where $\mathsf{Pre}_\subseteq(F \times F)$ is the lax Pre-lifting of $F \times F$ w.r.t. the order given above.[2] By Lemma 16 we get a lifting $\varrho : \mathsf{Pre}(T)\mathcal{G} \to \mathcal{G}\mathsf{Pre}(T)$ of ρ, with $\mathsf{Pre}(T)$ the canonical Pre-extension of T.

2. Since (X, α, ξ_1) and (X, α, ξ_2) are, respectively, a model and a lax model for λ, we have

$$\begin{array}{ccccc} SX & \xrightarrow{\alpha} & X & \xrightarrow{\xi_1} & FX \\ {\scriptstyle S\langle \xi_1, \mathrm{id}\rangle}\downarrow & & & & \uparrow{\scriptstyle F\alpha^\sharp} \\ S(FX \times X) & & \xrightarrow[\lambda_X]{} & & FTX \end{array} \qquad \begin{array}{ccccc} SX & \xrightarrow{\alpha} & X & \xrightarrow{\xi_2} & FX \\ {\scriptstyle S\langle \xi_2, \mathrm{id}\rangle}\downarrow & & \mid\vee & & \uparrow{\scriptstyle F\alpha^\sharp} \\ S(FX \times X) & & \xrightarrow[\lambda_X]{} & & FTX \end{array} \tag{10}$$

Notice that the left model is strict, yet we can also see it as a lax model for the discrete order on F. Hence we can pair the two coalgebra structures to obtain a lax model

$$\begin{array}{ccccc} SX & \xrightarrow{\alpha} & X & \xrightarrow{\langle \xi_1, \xi_2\rangle} & FX \times FX \\ {\scriptstyle S\langle \xi_1, \xi_2, \mathrm{id}\rangle}\downarrow & & \mid\vee & & \uparrow{\scriptstyle F\alpha^\sharp \times F\alpha^\sharp} \\ S(FX \times FX \times X) & & \xrightarrow[\tilde{\lambda}_X]{} & & (F \times F)TX \end{array} \tag{11}$$

for the monotone GSOS specification $\tilde{\lambda}$ considered above. We apply Lemma 19 for the lax model in (11) to obtain a lax bialgebra as in the next diagram with the carrier (X, Δ_X).

$$\begin{array}{ccccc} \mathsf{Pre}(T)X & \xrightarrow{\alpha^\sharp} & X & \xrightarrow{\langle \xi_1, \xi_2, \mathrm{id}\rangle} & \mathcal{G}X \\ {\scriptstyle \mathsf{Pre}(T)\langle \xi_1, \xi_2, \mathrm{id}\rangle}\downarrow & & \mid\vee & & \uparrow{\scriptstyle \mathcal{G}\alpha^\sharp} \\ \mathsf{Pre}(T)\mathcal{G}X & & \xrightarrow[\varrho_X]{} & & \mathcal{G}\mathsf{Pre}(T)X \end{array}$$

3. We consider the $\mathsf{Rel}^\uparrow$ liftings $\overline{\mathsf{Pre}(T)}$ and $\overline{\mathcal{G}}$ of $\mathsf{Pre}(T)$ and $\mathcal{G}$ obtained from Lemma 11, respectively Example 12. Using Proposition 3 we know that the distributive law ρ lifts to a distributive law $\overline{\rho} : \overline{TG} \Rightarrow \overline{GT}$ in Rel. To obtain the lifting of $\overline{\varrho}$ to $\mathsf{Rel}^\uparrow$ we apply Lemma 13 for the liftings $\overline{T}$, $\overline{G}$, $\overline{\mathsf{Pre}(T)}$ and $\overline{\mathcal{G}}$ and the liftings $\overline{\rho}$ and ϱ of ρ to Rel, respectively Pre. ◀

By Remark 9, since the order on X is discrete, we have that $\mathsf{Rel}^\uparrow_X \cong \mathsf{Rel}_X$. Hence the functor Ctx is indeed the usual predicate transformer for contextual closure and coalgebras for $(\overline{\mathsf{Pre}(F)} \times \mathsf{Pre}_\subseteq(F) \times \mathrm{Id})_{\langle \xi_1, \xi_2, \mathrm{id}\rangle}$ correspond to the usual weak bisimulations.

▶ **Example 21.** Recall from Example 18 that $\to$ and $\Rightarrow$ are, respectively, a model and a lax model for the positive GSOS specification of Example 2. By Theorem 20, it follows that up-to context (for the parallel composition of CCS) is compatible for weak bisimulation.

We can apply Theorem 20 to prove analogous results for the other operators of CCS with the exception of $+$ which is not part of a lax model, see Example 18. More generally, for any process algebra specified by a positive GSOS, one simply needs to check that the saturated transistion systems is a lax model. As explained in Section 4, this means that whenever $\Rightarrow$ satisfies the premises of a rule, it also satisfies its consequence. By [23, Lemma WB],

[2] Notice that $\mathcal{G} = \mathsf{Pre}(F) \times \mathsf{Pre}_\subseteq(F) \times \mathrm{Id}$ where $\mathsf{Pre}(F)$ and $\mathsf{Pre}_\subseteq(F)$ are the canonical, respectively the lax Pre-liftings of F w.r.t. the order given by $F_\subseteq$.

this holds for all calculi that conform to the so-called *simply WB cool* format [4], amongst which it is worth mentioning the fragment of CSP consisting of action prefixing, internal and external choice, parallel composition, abstraction and the 0 process ([23, Example 1]).

▶ **Corollary 22.** *For a simply WB cool GSOS language, up-to context is a compatible technique for weak bisimulation.*

6 Conclusion

We have shown that up-to context is compatible (and thus sound) for weak bisimulation whenever the strong and the weak transition systems are a model and a lax model for a positive GSOS specification, as it is the case for calculi adhering to the cool GSOS format [4, 23]. For our proof, we construct a tool-kit of abstract results that can be safely reused for proving compatibility for other coinductive notions. For instance, with our technology it is trivial to show that up-to context is compatible for bisimilarity and similarity for lax models of positive GSOS specifications, while in [6] this was proved just for (strict) models. For dynamic bisimilarity [17], one can use the lifting in (3) with a different saturated transition system that is obtained as in (1) but without the axiom $x \stackrel{\tau}{\Rightarrow} x$. Then for all the rules of CCS (including $+$), whenever this system satisfies the premises, it also satisfies its consequence, so it is a lax model; hence up-to context is compatible for dynamic bisimulation. We leave branching bisimilarity [24] for future work.

References

1 L. Aceto, W. Fokkink, and C. Verhoef. Structural operational semantics. In *Handbook of Process Algebra*, pages 197–292. Elsevier, 2001.

2 A. Balan and A. Kurz. Finitary functors: From Set to Preord and Poset. In *CALCO*, volume 6859 of *LNCS*, pages 85–99. Springer, 2011.

3 A. Balan, A. Kurz, and J. Velebil. Positive fragments of coalgebraic logics. In *CALCO*, volume 8089 of *LNCS*, pages 51–65. Springer, 2013.

4 B. Bloom. Structural operational semantics for weak bisimulations. *Theor. Comput. Sci.*, 146(1&2):25–68, 1995.

5 B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. In *POPL*, pages 229–239. ACM, 1988.

6 F. Bonchi, D. Petrişan, D. Pous, and J. Rot. Coinduction up-to in a fibrational setting. In *CSL-LICS*, page 20. ACM, 2014.

7 F. Bonchi and D. Pous. Checking NFA equivalence with bisimulations up to congruence. In *POPL*, pages 457–468. ACM, 2013.

8 D. Caucal. Graphes canoniques de graphes algébriques. *ITA*, 24:339–352, 1990.

9 R. de Simone. Higher-level synchronising devices in Meije-SCCS. *Theoretical Computer Science*, 37(0):245–267, 1985.

10 M. Fiore and S. Staton. Positive structural operational semantics and monotone distributive laws. In *CMCS*, page 8, 2010.

11 C. Hermida and B. Jacobs. Structural induction and coinduction in a fibrational setting. *Inf. and Comp.*, 145:107–152, 1997.

12 J. Hughes and B. Jacobs. Simulations in coalgebra. *TCS*, 327(1-2):71–108, 2004.

13 C.-K. Hur, G. Neis, D. Dreyer, and V. Vafeiadis. The power of parameterization in coinductive proof. In *POPL'13*, pages 193–206, New York, NY, USA, 2013. ACM.

14 B. Jacobs. Introduction to coalgebra. Towards mathematics of states and observations, 2014. Draft.

15 B. Klin. Bialgebras for structural operational semantics: An introduction. *TCS*, 412(38):5043–5069, 2011.
16 R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
17 U. Montanari and V. Sassone. CCS dynamic bisimulation is progressing. In *MFCS*, pages 346–356, 1991.
18 D. Pous. Complete lattices and up-to techniques. In *APLAS*, volume 4807 of *LNCS*, pages 351–366. Springer, 2007.
19 D. Pous and D. Sangiorgi. Enhancements of the bisimulation proof method. In *Advanced Topics in Bisimulation and Coinduction*, pages 233–289. Cambridge Univ. Press, 2012.
20 D. Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge Univ. Press, 2011.
21 A. M. Thijs. *Simulation and fixpoint semantics*. PhD thesis, Univ. of Groningen, 1996.
22 D. Turi and G. D. Plotkin. Towards a mathematical operational semantics. In *LICS*, pages 280–291. IEEE, 1997.
23 R. van Glabbeek. On cool congruence formats for weak bisimulations. *Theoretical Computer Science*, 412(28):3283–3302, 2011. Festschrift in Honour of Jan Bergstra.
24 R. van Glabbeek and W. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, 1996.

A Details for Example 12

Assume we have the following situation

$$\begin{array}{ccc} (h,k) & \overset{\overline{F\times F}(R)}{\cdots\cdots\cdots} & (h',k') \\ {\scriptstyle \mathsf{Rel}(F)(\leq)\times\mathsf{Rel}_{\subseteq}(F)(\leq)}\Big| & & \Big|{\scriptstyle \mathsf{Rel}(F)(\leq)\times\mathsf{Rel}_{\subseteq}(F)(\leq)} \\ (f,g) & \overset{\overline{F\times F}(R)}{\text{———}} & (f',g') \end{array}$$

This means that for all $a \in L$ we have the following

$$\begin{array}{rcl} (f,g)\ \overline{F\times F}(R)\ (f',g') & \Leftrightarrow & \forall x \in f(a).\ \exists y \in g'(a).xRy \\ & & \forall x \in f'(a).\ \exists y \in g(a).xRy \end{array}$$

$$\begin{array}{cc} (f,g)\ \mathsf{Rel}(F)(\leq) \times \mathsf{Rel}_{\subseteq}(F)(\leq)\ (h,k) & (f',g')\ \mathsf{Rel}(F)(\leq) \times \mathsf{Rel}_{\subseteq}(F)(\leq)\ (h',k') \\ \Updownarrow & \Updownarrow \\ \forall x \in f(a).\ \exists y \in h(a).x \leq y & \forall x \in f'(a).\ \exists y \in h'(a).x \leq y \\ \forall y \in h(a).\ \exists x \in f(a).x \leq y & \forall y \in h'(a).\ \exists x \in f'(a).x \leq y \\ \forall x \in g(a).\ \exists y \in k(a).x \leq y & \forall x \in g'(a).\ \exists y \in k'(a).x \leq y \end{array} \tag{12}$$

and we need to show

$$\begin{array}{l} \forall x \in h(a).\ \exists y \in k'(a).xRy \\ \forall x \in h'(a).\ \exists y \in k(a).xRy \end{array} \tag{13}$$

Using the fact the R is up-closed we can prove this using (12).

▶ Remark. Notice that some of the relations in (12) were not actually used in the proof. In order for the lifting $\overline{F \times F}(R)$ to restrict to up-closed relations, we need to carefully choose the Pre-liftings for $F \times F$. Indeed, we could replace the lifting $\mathsf{Pre}(F)$ with the lax relation lifting given by pointwise reverse inclusion $\mathsf{Pre}_{\supseteq}(F)$. However the proof would break if we would consider instead the Pre-lifting of $F \times F$ given by $\mathsf{Pre}_{\subseteq}(F) \times \mathsf{Pre}_{\subseteq}(F)$, since the functor $\mathsf{Pre}_{\subseteq}(F) \times \mathsf{Pre}_{\subseteq}(F)$ does not have a $\mathsf{Rel}^{\uparrow}$ lifting that also extends $\overline{F \times F}$.

On the Satisfiability of Indexed Linear Temporal Logics*

Taolue Chen[1], Fu Song[2], and Zhilin Wu[3,4]

1 **Department of Computer Science, Middlesex University London, UK**
2 **Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China**
3 **State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China**
4 **LIAFA, Université Paris Diderot, France**

Abstract

Indexed Linear Temporal Logics (ILTL) are an extension of standard Linear Temporal Logics (LTL) with quantifications over index variables which range over a set of process identifiers. ILTL has been widely used in specifying and verifying properties of parameterised systems, e.g., in parameterised model checking of concurrent processes. However there is still a lack of theoretical investigations on properties of ILTL, compared to the well-studied LTL. In this paper, we start to narrow this gap, focusing on the satisfiability problem, i.e., to decide whether a model exists for a given formula. This problem is in general undecidable. Various fragments of ILTL have been considered in the literature typically in parameterised model checking, e.g., ILTL formulae in prenex normal form, or containing only non-nested quantifiers, or admitting limited temporal operators. We carry out a thorough study on the decidability and complexity of the satisfiability problem for these fragments. Namely, for each fragment, we either show that it is undecidable, or otherwise provide tight complexity bounds.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Satisfiability, Indexed linear temporal logic, Parameterised systems

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.254

1 Introduction

Many concurrent systems are comprised of a finite number, but arbitrary many, of processes running in parallel. They are often referred to as *parameterised systems* [5]. In this context, the parameterised model checking problem, loosely speaking, is to decide whether a given temporal property holds irrespective of the number of participating processes.

Parameterised systems and their verification require specifications of temporal properties. For the verification of standard concurrent processes, temporal logics, typically Linear Temporal Logic (LTL), Computation Tree Logic (CTL), or their combination CTL*, have become the *de facto* specification methods. Specifications of parameterised systems largely follow this paradigm. In the setting, logics are usually extended with quantifiers over a set

* Corresponding authors: Taolue Chen, Fu Song and Zhilin Wu. Taolue Chen is partially supported by an oversea grant from the State Key Laboratory of Novel Software Technology, Nanjing Unviersity. Fu Song is partially supported by Shanghai Pujiang Program (No. 14PJ1403200), NSFC Projects (No. 61402179), Shanghai ChenGuang Program (No. 13CG21). Zhilin Wu is partially supported by the NSFC projects (Grant No. 61100062, 61272135, and 61472474), and the visiting researcher program of China Scholarship Council from 2014.06.10 to 2015.06.09.

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 254–267

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

of process identifiers, giving rise to various *indexed* versions of temporal logics. This dates back to [22] which introduced an extension of LTL with spatial operators ranging over the processes of a parameterised system. It was shown that the satisfiability problem of this logic is undecidable. After that, there has been a large body of work on indexed temporal logics. To name just a few, indexed $\mathsf{CTL}^*\backslash X$, an extension of CTL^* with quantifiers over process identifiers but excluding the "next" operator X, was introduced by Browne et al. in [5]. They studied the relation between indexed $\mathsf{CTL}^*\backslash X$ and bisimulation among parameterised systems. Emerson et al. investigated the parameterised model checking problem of fragments of indexed $\mathsf{CTL}^*\backslash X$ in prenex normal form over rings [12]. They also studied symmetry properties in model-checking systems against indexed LTL and indexed CTL^* with non-nested index quantifiers and only local atomic propositions [13]. German et al. showed that the parameterised model-checking problem of indexed LTL without global atomic propositions or nested quantifiers is undecidable [16]. Clarke et al. considered the parameterised model checking problem of indexed $\mathsf{LTL}\backslash X$ over token passing systems with respect to general topologies [6]. Very recently, Aminof et al. studied the same problem for indexed $\mathsf{CTL}^*\backslash X$ [1] unifying and extending the results in [12, 6].

Since indexed temporal logics play a fundamental role in specification and verification of parameterised systems, it is of great importance to investigate their basic (meta-)properties, along the same line as LTL, CTL, or CTL^*. In this paper, we focus on one such property, i.e., the satisfiability problem of the indexed LTL (ILTL), which, given an ILTL formula, aims to determine whether there exists a model satisfying the formula. In theory, satisfiability is probably one of the first questions one intends to answer, especially when the computational aspect of the logic is concerned. In practice, decision procedures for satisfiability have potential applications in *synthesis* of concurrent programs from their logical specifications, and play an important role in checking the consistency of specifications in an early stage of system design [27, 28]. However, in spite of its importance, to the best of our knowledge, the satisfiability problem of ILTL has not been studied systematically. The current work aims to fill in such a gap.

One immediate result is that ILTL is undecidable in general (see Proposition 1 in Section 2). We then consider the following two natural fragments of ILTL, i.e.,

- $\mathsf{ILTL}_{\mathsf{pnf}}$, the ILTL formulae in prenex normal form, and
- NN-ILTL, the ILTL formulae where the quantifiers are non-nested.

We remark these two fragments are largely disjoint (module some trivial cases), and they are two representative classes of properties which are indeed extensively used in the verification of parameterised systems [12, 16]. Furthermore, in most of the work on parameterised model checking, e.g., [1, 11, 12, 14], indexed temporal logics are considered excluding the global atomic propositions, or with only a limited subset of temporal operators (for instance, the "next" operator X is usually disallowed). From this practical point of view, it is of paramount importance to consider these fragments, which are the main objects of the current paper.

In this paper, we mainly focus on the decidability and complexity of the satisfiability problem of $\mathsf{ILTL}_{\mathsf{pnf}}$ and NN-ILTL. For each of these two fragments, we further study the impact of global atomic propositions and temporal operators on the decidability/complexity. These results implicitly depict that what kind of specifications can be automatically checked for consistency at the early stage of parameterised system design, and how efficiently this can be done.

Contribution. The results obtained in this paper are summarised in Figure 1 which is organised hierarchically in term of syntax inclusion. In particular we show that

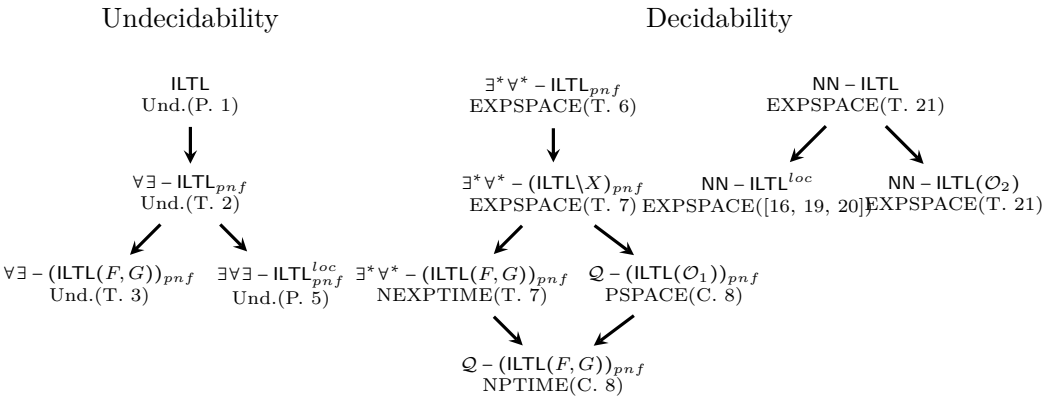


Figure 1 Summary of the results: Und. (Undecidable), $T.$ (Theorem), $C.$ (Corollary), $\mathcal{Q} \in \{\exists\forall^*\} \cup \{\exists^*\forall^k \mid k \in \mathbb{N}\}$, $\mathcal{O}_1$ is $\{X, F, G\}$ or $\{U, R\}$, $\mathcal{O}_2$ is $\{X, F, G\}$ or $\{U, R\}$ or $\{F, G\}$.

- the satisfiability problem of formulae in prenex normal form starting with $\forall\exists$ is undecidable even with only "future" and "global" temporal operators ($\forall\exists$ – (ILTL(F,G))$_{pnf}$ in Figure 1). This extends to formulae starting with $\exists\forall\exists$ with only local atomic propositions ($\exists\forall\exists$ – ILTL$^{loc}_{pnf}$ in Figure 1),
- the satisfiability problem of the formulae in prenex normal form starting with $\exists^*\forall^*$ is decidable with an exponential blow-up in complexity comparing to their counterparts of LTL w.r.t. various combinations of temporal operators ($\exists^*\forall^*$ – ILTL$_{pnf}$, $\exists^*\forall^*$ – (ILTL\X)$_{pnf}$, and $\exists^*\forall^*$ – (ILTL(F,G))$_{pnf}$ in Figure 1),
- the satisfiability problem of the formulae in prenex normal form starting with $\exists\forall^*, \exists^*\forall^k$, where $k \geq 0$ is a fixed number, is decidable with the same complexity as their counterparts of LTL w.r.t. various combinations of temporal operators ($\mathcal{Q}$ – (ILTL$(\mathcal{O}_1)$)$_{pnf}$ and $\mathcal{Q}$ – (ILTL(F,G))$_{pnf}$ in Figure 1),
- the satisfiability problem of the formulae with non-nested quantifiers is EXPSPACE-complete, and this even holds for formulae allowing the "future" and "global" temporal operators only (NN – ILTL$(\mathcal{O}_2)$ in Figure 1).

We outline some techniques we use to obtain the aforementioned results. The upper bound of ILTL$_{\mathsf{pnf}}$ with the quantifier prefixes $\exists^*\forall^*$ is obtained by instantiating the universal quantifiers with all the possible combinations of existentially quantified ones, thus, removing the universal quantifiers (Theorem 6). For the upper bound of NN – ILTL, a concept of potentially Eulerian directed graphs is introduced which plays an essential role in the decision procedure (Theorem 21). Moreover, we exploit the index quantifiers and a property regarding the expressiveness of "future" and "global" temporal operators (cf. Lemma 4) to obtain undecidability and complexity lower bounds (Theorem 3 and Theorem 21).

Related work. We have already discussed various indexed extensions of standard temporal logics and the related results [22, 5, 14, 16]. Since process identifiers can also be seen as a sort of data values, ILTL is also related to temporal logics over data words or words over infinite alphabets.

The most relevant work includes: LTL with freeze quantifiers (i.e. registers) over a singly attributed data word [9, 15]; LTL with navigation mechanisms for a single (or a tuple of) data attribute(s) over multi-attributed data words (i.e., data words where each position carries multiple data values which can be referred to by a fixed set of attributes) [21, 8, 7];

LTL, CTL and CTL* with variable quantifications (called variable LTL/CTL/CTL*), where the variables range over an infinite data domain [17, 18, 10, 25]. Decidability and complexity issues of these logics and variants thereof were studied.

These logics are interpreted over data words where each position carries only a *fixed number* of data values, whereas ILTL is interpreted over computation traces in parameterised systems. While computation traces can also be seen as data words by treating process identifiers as data values, these data words are significantly different than the traditional ones studied before. Namely, each position of these data words carries an *unbounded* number of data values, and all the data values occur in every position. To our best knowledge, data words of this special structure and their logics have not been considered in the infinite alphabet community.

Structure. The rest of the paper is organised as follows. Section 2 presents the preliminaries. Section 3 is devoted to ILTL formulae in prenex normal form, and Section 4 is for ILTL formulae with non-nested quantifiers. Due to space limitation, most of the proofs are omitted and will appear in the journal version of this paper.

2 Preliminaries

For $k \in \mathbb{N}$, let $[k] = \{0, \ldots, k-1\}$. For a sequence $\alpha = \alpha_0\alpha_1 \ldots$ and $j \in \mathbb{N}$, we use $\alpha[j]$ to denote the element of α at position j.

Let $\mathcal{I}$ be an infinite set of process identifiers and $\mathcal{X}$ be a set of index variables which range over $\mathcal{I}$. Let AP be a finite set of *global* atomic propositions and AP' be a finite set of *local* atomic propositions. We assume that $AP \cap AP' = \varnothing$. The intention of AP' is to specify process-specific properties, so each occurrence of AP' in formulae is parameterised with an index variable from $\mathcal{X}$.

The formulae of indexed LTL (ILTL) are defined by the following BNF,

$$\varphi ::= true \mid false \mid p \mid \neg p \mid p'(x) \mid \neg p'(x) \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid X\varphi \mid \varphi \ U \ \varphi \mid \varphi \ R \ \varphi \mid \exists x.\varphi \mid \forall x.\varphi,$$

where $p \in AP, p' \in AP', x \in \mathcal{X}$.

Moreover, standard "future" (F) and "global" (G) temporal operators can be introduced as abbreviations: $F\psi \equiv true \ U \ \psi$, $G\psi \equiv false \ R \ \psi$.

Let $\mathsf{free}(\varphi)$ denote the set of free variables occurring in φ. An ILTL formula containing no free variables is called a *closed* ILTL formula. In addition, the size of φ, denoted by $|\varphi|$, is defined as the number of symbols occurring in φ. For an ILTL formula φ, let $\neg\varphi$ denote its complement (negation), and let $\overline{\varphi}$ denote the *positive normal form* of $\neg\varphi$, that is obtained by pushing the negation inside of operators. For instance, if $\varphi = \exists x.\ Fp'(x)$, then $\overline{\varphi} = \forall x.\ G\neg p'(x)$.

ILTL formulae are typically used to specify and verify parameterised systems. Naturally, ILTL formulae are interpreted over computation traces of parameterised systems. In the present paper, we adopt the definition of the computation traces from [16]. A *computation trace* over $AP \cup AP'$ is a tuple $trc = (\alpha, I, (\beta_i)_{i\in I})$, where $\alpha \in (2^{AP})^\omega$ is an ω-sequence of valuations over the global atomic propositions from AP, $I \subseteq \mathcal{I}$ is a *finite* set of process identifiers, and for each $i \in I$, $\beta_i \in (2^{AP'})^\omega$ is a local computation trace, i.e. an ω-sequence of valuations over the local atomic propositions from AP'.

Let φ be an ILTL formula, $trc = (\alpha, I, (\beta_i)_{i\in I})$ be a computation trace, $\theta : \mathsf{free}(\varphi) \to I$ be an assignment of the process identifiers (from I) to the free variables in φ, and $n \in \mathbb{N}$. Then (trc, θ, n) satisfies φ, denoted by $(trc, \theta, n) \vDash \varphi$, is defined as follows.

- $(trc, \theta, n) \vDash p$ (resp. $\neg p$) if $p \in \alpha[n]$ (resp. $p \notin \alpha[n]$),
- $(trc, \theta, n) \vDash p'(x)$ (resp. $\neg p'(x)$) if $p' \in \beta_{\theta(x)}[n]$ (resp. $p' \notin \beta_{\theta(x)}[n]$),
- $(trc, \theta, n) \vDash \exists x.\varphi_1$ if there is $i \in I$ such that $(trc, \theta[i/x], n) \vDash \varphi_1$, where $\theta[i/x]$ is the same as θ, except for assigning i to x,
- $(trc, \theta, n) \vDash \forall x.\varphi_1$ if for each $i \in I$, $(trc, \theta[i/x], n) \vDash \varphi_1$,
- $(trc, \theta, n) \vDash \varphi_1 \vee \varphi_2$ if $(trc, \theta, n) \vDash \varphi_1$ or $(trc, \theta, n) \vDash \varphi_2$,
- $(trc, \theta, n) \vDash \varphi_1 \wedge \varphi_2$ if $(trc, \theta, n) \vDash \varphi_1$ and $(trc, \theta, n) \vDash \varphi_2$,
- $(trc, \theta, n) \vDash X\varphi$ if $(trc, \theta, n+1) \vDash \varphi$,
- $(trc, \theta, n) \vDash \varphi_1 \ U \ \varphi_2$ if there is $k \geq n$ s.t. $(trc, \theta, k) \vDash \varphi_2$, and for all $j : n \leq j < k$, $(trc, \theta, j) \vDash \varphi_1$,
- $(trc, \theta, n) \vDash \varphi_1 \ R \ \varphi_2$ if for all $k \geq n$, $(trc, \theta, k) \vDash \varphi_2$, or there is $k \geq n$ s.t. $(trc, \theta, k) \vDash \varphi_1$, and for all $j : n \leq j \leq k$, $(trc, \theta, j) \vDash \varphi_2$.

Note that if φ is a closed ILTL formula, then θ has an empty domain and thus is omitted. Namely we simply write $(trc, n) \vDash \varphi$. In addition, for a closed ILTL formula φ, we use $trc \vDash \varphi$ to abbreviate $(trc, 0) \vDash \varphi$. For a closed ILTL formula φ, let $\mathcal{L}(\varphi)$ denote the set of computation traces trc such that $trc \vDash \varphi$. The *satisfiability* problem of ILTL is:

Given a closed ILTL *formula* φ*, decide whether* $\mathcal{L}(\varphi)$ *is empty.*

As a warm-up, we show that the satisfiability problem of ILTL is undecidable in general[1], which is obtained by a reduction from the PCP problem [29]. Recall that PCP problem is, given an instance $(u_j, v_j)_{1 \leq j \leq n}$, where u_j, v_j are finite words over an alphabet Σ, to decide whether there exists a sequence of indices $j_1 \dots j_m$ such that $u_{j_1} \dots u_{j_m} = v_{j_1} \dots v_{j_m}$. The main idea of the reduction is to encode a solution $j_1 \dots j_m$ of the PCP problem as a computation trace $trc = (\alpha, I, (\beta_i)_{i \in I})$ such that $\alpha = w_{j_1} w_{j_2} \dots w_{j_m} atom(\#) \ \overline{w_{j_1}} \dots \overline{w_{j_m}} (atom(\$))^\omega$, where $w_{j_1} w_{j_2} \dots w_{j_m}$ corresponds to $u_{j_1} \dots u_{j_m}$, $\overline{w_{j_1}} \dots \overline{w_{j_m}}$ corresponds to $v_{j_1} \dots v_{j_m}$, and a local atomic proposition p' is used in $(\beta_i)_{i \in I}$ to guarantee the equality of $u_{j_1} \dots u_{j_m}$ and $v_{j_1} \dots v_{j_m}$.

▶ **Proposition 1.** *The satisfiability problem of* ILTL *is undecidable.*

In this paper, we shall consider the following fragments of ILTL with abbreviations:

- $\mathsf{ILTL}_{\mathsf{pnf}}$ denotes the fragment of ILTL where formulae are in *prenex normal form*, that is $\{\forall, \exists\}$ quantifications appear only at the beginning of the formula. In particular, let $\mathcal{Q} \subseteq \{\exists, \forall\}^*$. Then $\mathcal{Q} - \mathsf{ILTL}_{\mathsf{pnf}}$ denotes the fragment of $\mathsf{ILTL}_{\mathsf{pnf}}$ where the quantifier prefixes belong to $\mathcal{Q}$.
- NN – ILTL denotes the fragment of ILTL where the quantifiers are *not* nested, that is, for each formula $Q_1 x.\varphi_1$ such that $Q_2 y.\varphi_2$ is a subformula of φ_1, it holds that x is not a free variable of φ_2, where $Q_1, Q_2 \in \{\forall, \exists\}$.
- $\mathsf{ILTL}(\mathcal{O})$ for $\mathcal{O} \subseteq \{X, F, G, U, R\}$ denotes the fragment of ILTL where only temporal operators from $\mathcal{O}$ are used. Moreover, we use $\mathsf{ILTL}\backslash X$ as an abbreviation of $\mathsf{ILTL}(U, R)$, where the X operator is forbidden.
- $\mathsf{ILTL}^{\mathsf{loc}}$ denotes the fragment of ILTL where there are no global atomic propositions, that is, $AP = \varnothing$.

[1] Proposition 1 is subsumed by Theorem 2 and Theorem 3 in the next section. We choose to present the weaker and easier result first, instead of giving the strongest result (Theorem 3) directly. This might hopefully illustrate the idea of the proof and facilitate readers' understanding.

These notations might be combined to define more (refined) fragments, e.g. $(\mathsf{ILTL}(F,G))_{\mathsf{pnf}}$ denotes the fragment of $\mathsf{ILTL}_{\mathsf{pnf}}$ where only temporal operators F and G are used.

For establishing the complexity lower-bounds, we shall use the *tiling problems*, which have various versions to capture different complexity classes [4]. Among others, the *exponential-size square tiling problem* is specified by a tuple $(n, \Delta, H, V, t_S, t_F)$, where $n \in \mathbb{N}$ is encoded in unary, Δ is a finite set of tiles, $H, V \subseteq \Delta \times \Delta$ are called the horizontal and vertical constraints, $t_S, t_F \in \Delta$ are the initial tile and final tile respectively. The task is to decide whether there is a tiling of the square $[2^n] \times [2^n]$, that is, a function $f : [2^n] \times [2^n] \to \Delta$, satisfying the following conditions,

- horizontal constraint: for every $j_1, j_2 : j_1 \in [2^n], 0 \leq j_2 < 2^n - 1$, $(f(j_1, j_2), f(j_1, j_2+1)) \in H$,
- vertical constraint: for every $j_1, j_2 : 0 \leq j_1 < 2^n - 1, j_2 \in [2^n]$, $(f(j_1, j_2), f(j_1+1, j_2)) \in V$,
- initial and final constraint: $f(0,0) = t_S$, $f(2^n - 1, 2^n - 1) = t_F$.

This problem is known to be NEXPTIME-complete.

Likewise, the *exponential-size corridor tiling problem* is given by a tuple $(n, \Delta, H, V, t_S, t_F)$. The task is to decide whether there is $k \geq 1$ such that the integer plane of size $k \times 2^n$ can be tiled, that is, by a function $f : [k] \times [2^n] \to \Delta$, so that the following conditions hold,

- horizontal constraint: for each $j_1 \in [k], 0 \leq j_2 < 2^n - 1$, $(f(j_1, j_2), f(j_1, j_2 + 1)) \in H$,
- vertical constraint: for each $0 \leq j_1 < k - 1, j_2 \in [2^n]$, $(f(j_1, j_2), f(j_1 + 1, j_2)) \in V$,
- initial and final constraint: $f(0,0) = t_S$, $f(k - 1, 2^n - 1) = t_F$.

This problem is known to be EXPSPACE-complete.

3 Formulae in Prenex Normal Form

In this section, we focus on $\mathsf{ILTL}_{\mathsf{pnf}}$ formulae in prenex normal form, i.e., formulae of the form $Q_1 x_1 \dots Q_k x_k . \psi$, where $Q_j \in \{\forall, \exists\}$ for all $1 \leq j \leq k$, and ψ is quantifier free.

3.1 Undecidability

Our first (negative) result states that even with one alternation of the existential and universal quantifiers, the satisfiability problem of $\mathsf{ILTL}_{\mathsf{pnf}}$ is already undecidable.

▶ **Theorem 2.** *The satisfiability problem of* $\forall\exists - \mathsf{ILTL}_{\mathsf{pnf}}$ *is undecidable.*

The ILTL formulae used in the proof of Proposition 1 are not in $\forall\exists - \mathsf{ILTL}_{\mathsf{pnf}}$. In the proof of Theorem 2, we adapt the reduction in Proposition 1 so that only formulae from $\forall\exists - \mathsf{ILTL}_{\mathsf{pnf}}$ are used in the reduction. We then investigate whether restricting certain temporal operators leads to decidability. Unfortunately, this is not the case. Indeed, the undecidability stands even when only the "future" and "global" temporal operators are present. Clearly, this implies that the satisfiability of indexed temporal logics without "next" temporal operators, which are prevailing in the study of parameterised model checking (e.g. $\mathsf{ICTL}^*\backslash X$ in [1]), is undecidable in general.

▶ **Theorem 3.** *The satisfiability problem of* $\forall\exists - (\mathsf{ILTL}(F,G))_{\mathsf{pnf}}$ *is undecidable.*

To prove this result, we first show the undecidability of $\forall\exists - (\mathsf{ILTL}\backslash X)_{\mathsf{pnf}}$, which is obtained by adapting the proof of Theorem 2 and encoding the "next" operator with the "until" operator. As the next step, we further encode the "until" operator with the "global" and "future" operators, with the help of the index quantifiers and the following Lemma 4 [24]. Lemma 4 shows that F, G are sufficiently strong to express some properties that could only be defined by the "until" operator at the first sight, although in a more technical and less intuitive way.

▶ **Lemma 4** ([24]). *Let $Z \subseteq \mathcal{X}$ be a finite set of variables and $\Sigma = 2^{AP \cup (AP' \times Z)}$. Suppose $L = A_1^* B_1 A_2^* B_2 \dots A_k^* B_k A_{k+1}^\omega$ such that $A_1, B_1, \dots, A_k, B_k, A_{k+1} \subseteq \Sigma$, and for each $j : 1 \le j \le k$, $B_j \subseteq (A_j \setminus A_{j+1})$. Then L can be defined by an $\mathsf{LTL}(F,G)$ formula φ over the set of atomic propositions $AP \cup (AP' \times Z)$. Moreover, if for each $j : 1 \le j \le k+1$, A_j is defined by a formula ψ_j, and for each $j : 1 \le j \le k$, B_j is defined by a formula ξ_j, then the $\mathsf{LTL}(F,G)$ formula φ can be constructed from these formulae in linear time w.r.t. $\sum_{1 \le j \le k+1} |\psi_j| + \sum_{1 \le j \le k} |\xi_j|$.*

To give an idea how the languages L in Lemma 4 can be expressed in $\mathsf{LTL}(F,G)$, let us look at the following example: Suppose $\Sigma = 2^{\{p_1'(x), p_2'(x)\}}$, and $L = A_1^* B_1 A_2^\omega$, where $A_1 = B_1 = \{\{p_1'(x)\}\}$, and $A_2 = \{\{p_2'(x)\}\}$. Intuitively, L specifies that $p_1'(x) \wedge \neg p_2'(x)$ always holds until $\neg p_1'(x) \wedge p_2'(x)$ holds, and the latter holds forever afterwards. If the "until" operator is allowed, then L can be defined easily by $p_1'(x) \wedge \neg p_2'(x) \wedge (p_1'(x) \wedge \neg p_2'(x)) \; U \; (G(\neg p_1'(x) \wedge p_2'(x)))$. On the other hand, without the "until" operator, L can be defined by

$$\varphi = p_1'(x) \wedge \neg p_2'(x) \wedge G[(p_1'(x) \wedge \neg p_2'(x)) \vee G(\neg p_1'(x) \wedge p_2'(x))] \wedge FG(\neg p_1'(x) \wedge p_2'(x)).$$

As mentioned in the introduction, we are also interested in the influence of the global atomic propositions (which are needed in the above reductions). What happens to Theorem 3 if only local atomic propositions are allowed? We show that in this case the undecidability stands at the cost of a higher level of alternations of quantifiers. Namely, we have

▶ **Proposition 5.** *The satisfiability problem of $\exists\forall\exists - \mathsf{ILTL}^{\mathsf{loc}}_{\mathsf{pnf}}$ is undecidable.*

The proof is obtained by encoding global atomic propositions with local atomic propositions, with the aid of the additional existential quantifier. Similar results also hold when the set of temporal operators is restricted.

3.2 Decidability

From Theorem 2, we know that the satisfiability problem of $\mathsf{ILTL}_{\mathsf{pnf}}$ is undecidable, even with the quantifier prefix $\forall\exists$. In this section, we will show that the undecidability result of Theorem 2 is tight in the sense that the satisfiability problem of $\exists^*\forall^* - \mathsf{ILTL}_{\mathsf{pnf}}$ is decidable (more precisely, EXPSPACE-complete).

▶ **Theorem 6.** *The satisfiability problem of $\exists^*\forall^* - \mathsf{ILTL}_{\mathsf{pnf}}$ is EXPSPACE-complete.*

Proof sketch. The EXPSPACE upper bound is obtained by instantiating the universal quantifiers with all the possible combinations of existentially quantified ones. Let $\varphi = \exists x_1 \dots x_k \forall y_1 \dots y_l.\ \psi$ be an $\exists^*\forall^* - \mathsf{ILTL}_{\mathsf{pnf}}$ formula, where ψ is quantifier-free. We construct an $\exists^* - \mathsf{ILTL}_{\mathsf{pnf}}$ formula φ' as $\exists x_1 \dots x_k.\ \bigwedge_f \psi_f$, where f ranges over all the functions from $\{1, \dots, l\}$ to $\{1, \dots, k\}$, and ψ_f is obtained from ψ by replacing each occurrence of $p'(y_j)$ with $p'(x_{f(j)})$, for every $j : 1 \le j \le l$ and $p' \in AP'$. Note that the size of φ' is exponential in $|\varphi|$. Moreover, it is not hard to see that the satisfiability of $\exists^* - \mathsf{ILTL}_{\mathsf{pnf}}$ can be reduced in linear time to that of LTL. Therefore, as the satisfiability of LTL is PSPACE-complete [23], we get the EXPSPACE upper bound for $\exists^*\forall^* - \mathsf{ILTL}_{\mathsf{pnf}}$.

For the lower bound, we reduce from the exponential-size corridor tiling problem. Let $(n, \Delta, H, V, t_S, t_F)$ be an instance of the exponential-size corridor tiling problem. Suppose $m = \lceil \log(|\Delta|) \rceil$. Let $AP' = \{p', p_1', \dots, p_n', q_1', \dots, q_m'\}$ be the set of local atomic propositions, where $p_1', \dots, p_n'$ are used to encode the addresses of each row (that is, the elements of $[2^n]$) of the tiling problem, $q_1', \dots, q_m'$ are used to encode the set of tiles in Δ, and p' is used as a marker. The reduction also uses two existential variables x_1, x_2 such that for each

$j : 1 \le j \le n$, exactly one of $p'_j(x_1)$ or $p'_j(x_2)$ holds at each position. Intuitively, for each address $\ell \in [2^n]$, $p'_j(x_1)$ (resp. $p'_j(x_2)$) holds iff the j-th bit of the binary encoding of ℓ is 0 (resp. 1). In addition, the universally quantified variables $y_1, \ldots, y_n$ are used to specify the horizontal and universal constraints of the tiling problem. ◀

As before, we now examine whether restricting temporal operators is beneficial to reduce the complexity. An easy observation is that since the lower bound proof of Theorem 6 only uses the operators X, F, G, the satisfiability problem of $\exists^*\forall^* - (\mathsf{ILTL}(X, F, G))_{\mathsf{pnf}}$ is EXPSPACE-complete. For the other restrictions of temporal operators, we obtain the following results.

▶ **Theorem 7.** *The satisfiability problem is*
- *EXPSPACE-complete for* $\exists^*\forall^* - (\mathsf{ILTL}\backslash X)_{\mathsf{pnf}}$,
- *NEXPTIME-complete for* $\exists^*\forall^* - (\mathsf{ILTL}(F, G))_{\mathsf{pnf}}$.

The upper-bounds are obtained by the similar argument of Theorem 6 and the complexity of respective fragments of LTL [23]. The lower bounds are obtained by (refined) reductions from the exponential-size corridor and exponential-size square tiling problems respectively.

Moreover, by a refined analysis of the proof of Theorem 6 and the complexity of various fragments of LTL [23], we obtain the following result.

▶ **Corollary 8.** *For each* $\mathcal{Q} \in \{\exists\forall^*\} \cup \{\exists^*\forall^k \mid k \in \mathbb{N}\}$, *the satisfiability problem is*
- *PSPACE-complete for* $\mathcal{Q} - (\mathsf{ILTL}(X, F, G))_{\mathsf{pnf}}$ *and* $\mathcal{Q} - (\mathsf{ILTL}\backslash X)_{\mathsf{pnf}}$,
- *NP-complete for* $\mathcal{Q} - (\mathsf{ILTL}(F, G))_{\mathsf{pnf}}$.

4 Non-Nested Quantifiers

In this section, we focus on the satisfiability of $\mathsf{NN} - \mathsf{ILTL}$, i.e., the fragment of ILTL where quantifiers are not nested. A "folklore" theorem, concerning $\mathsf{NN} - \mathsf{ILTL}^{\mathsf{loc}}$ (i.e., $\mathsf{NN} - \mathsf{ILTL}$ with *local* atomic propositions solely), states that the satisfiability problem is EXPSPACE-complete. In [16], the authors attributed this result to [19], where the temporal logics with knowledge operators were studied. Among others, the complexity upper bound for the satisfiability of $LKT_{(m)}$ ($m \ge 1$) over synchronous unbounded memory models[2] was given by introducing a concept called k-trees and reducing to the nonemptiness of Büchi tree automata (cf. Theorem 4.1 in [19]). As pointed out in [16], $\mathsf{NN} - \mathsf{ILTL}^{\mathsf{loc}}$ corresponds to $LKT_{(1)}$ in [19] (note in this case, only 1-trees are needed). Nevertheless, the EXPSPACE result of $\mathsf{NN} - \mathsf{ILTL}^{\mathsf{loc}}$ via [19] is unsatisfactory in that: (1) the construction of [19] is for temporal logics with knowledge operators which have specific syntax and semantics, and is rather technical and only sketched (referred to [26] indeed), hence it is fair to see the result for $\mathsf{NN} - \mathsf{ILTL}^{\mathsf{loc}}$ is not self-contained and difficult to access; (2) more importantly, the correctness proof of the construction in [19] is not available, and based on our efforts in discovering the proof and the results presented in the rest of this section, the correctness of the construction in [19] is not clear, at least to us. We propose a self-contained proof for the complexity results of $\mathsf{NN} - \mathsf{ILTL}$ satisfiability (which also extends the result for $\mathsf{NN} - \mathsf{ILTL}^{\mathsf{loc}}$). Our construction for the EXPSPACE upper bound is different from that in [19]. Some new concepts, e.g., the potential Eulerian directed graphs (Definition 14), are needed for us. Moreover, we strengthen the EXPSPACE lower bound to $\mathsf{NN} - \mathsf{ILTL}(F, G)$, that is, the

[2] where m is the number of "knowers".

fragment of NN – ILTL where only "future" and "global" temporal operators are available. This result, together with Theorem 3, shows that with index quantifiers, even very weak temporal operators are powerful enough to exhibit undecidability or complexity lower bounds.

Throughout this section, we assume φ to be an NN – ILTL formula. In addition, we assume that only one variable x occurs in φ (i.e., x is reused in distinct quantifiers). Without loss of generality, we assume that AP (resp. AP') is the set of global (resp. local) atomic propositions occurring in φ (otherwise, it is sufficient to consider the restriction of AP and AP' to those occurring in φ). We first introduce some notations.

A *directed multigraph* G is a pair (V, E) where V is a set of vertices, E is a *multiset* of ordered pairs $(v, v') \in V \times V$. The elements of E are called *arcs*. The distinct copies of the same pair (v, v') in E are called *parallel arcs*. For an arc $e \in E$ which is a copy of (v, v'), v and v' are called the *tail* and the *head* of e respectively. An arc-labelled directed multigraph is a tuple (V, E, L), where (V, E) is a directed multigraph and $L : E \to A$ (where A is a finite set) is an arc-labelling function. A (finite) *path* in a directed multigraph $G = (V, E)$ is a sequence $v_0 e_1 v_1 \dots v_{n-1} e_n v_n$ (where $n \geq 1$) such that for each $j : 1 \leq j \leq n$, e_j is an arc with the tail v_{j-1} and the head v_j. The length of a path is the number of arcs in the path. A *cycle* in G is a path $v_0 e_1 v_1 \dots v_{n-1} e_n v_n$ such that $v_0 = v_n$. A *directed graph* is a directed multigraph (V, E) without parallel arcs, that is, E is a *set* of pairs $(v, v') \in V \times V$. For a directed graph $G = (V, E)$, since each arc is uniquely identified by its head and its tail, a path can also be seen as a vertex sequence $v_0 v_1 \dots v_n$ such that for each $j : 0 \leq j < n$, $(v_j, v_{j+1}) \in E$. In addition, later on, sometimes we also write an arc-labelled directed graph $G = (V, E, L)$ as a pair (V, E') such that $E' = \{(v, L(v, v'), v') \mid (v, v') \in E\}$. A directed multigraph G is said to be *acyclic* if there are no cycles in G, and is said to be *strongly connected* if for every pair of vertices v, v', there is a path from v to v' and vice versa. A directed multigraph G is *connected* if the underlying *undirected* multigraph G, i.e. the multigraph obtained from G by ignoring the directions of arcs, is connected.

We then introduce some definitions related to φ. Let $cl(\varphi)$ denote the *closure* of formulae including the set of subformulae in φ, their complements, as well as $X(\psi_1 U \psi_2)$ and $X(\psi_1 R \psi_2)$ for $\psi_1 U \psi_2, \psi_1 R \psi_2 \in cl(\varphi)$ respectively. It is not difficult to observe that the size of $cl(\varphi)$ (the number of formulae in $cl(\varphi)$), denoted by $|cl(\varphi)|$, satisfies that $|cl(\varphi)| = O(|\varphi|)$.

▶ **Definition 9** (Atom). $\Psi \subseteq cl(\varphi)$ is an *atom* over φ if the following conditions hold:

- For each $\psi \in cl(\varphi)$, $\psi \in \Psi$ iff $\overline{\psi} \notin \Psi$.
- For each $\psi_1 \wedge \psi_2 \in cl(\varphi)$, $\psi_1 \wedge \psi_2 \in \Psi$ iff $\psi_1 \in \Psi$ and $\psi_2 \in \Psi$.
- For each $\psi_1 \vee \psi_2 \in cl(\varphi)$, $\psi_1 \vee \psi_2 \in \Psi$ iff $\psi_1 \in \Psi$ or $\psi_2 \in \Psi$.
- For each $\psi_1 U \psi_2 \in cl(\varphi)$, $\psi_1 U \psi_2 \in \Psi$ iff $\psi_2 \in \Psi$ or $\psi_1, X(\psi_1 U \psi_2) \in \Psi$.
- For each $\psi_1 R \psi_2 \in cl(\varphi)$, $\psi_1 R \psi_2 \in \Psi$ iff $\psi_2, \psi_1 \in \Psi$ or $\psi_2, X(\psi_1 R \psi_2) \in \Psi$.
- For each $\forall x.\psi \in cl(\varphi)$, if $\forall x.\psi \in \Psi$, then $\psi \in \Psi$.
- For each $\exists x.\psi \in cl(\varphi)$, if $\psi \in \Psi$, then $\exists x.\psi \in \Psi$.

▶ Remark. For formulae of the form $\exists x.\psi \in cl(\varphi)$, it is possible that $\exists x.\psi \in \Psi$, but $\psi \notin \Psi$. Let $\mathcal{A}$ denote the set of all atoms. It is not hard to see that $|\mathcal{A}| \leq 2^{|cl(\varphi)|}$.

▶ **Definition 10** (Macro state). A *macro state* S w.r.t. φ is a nonempty set of atoms satisfying the following conditions:

1. for each $p \in AP$ and $\Psi, \Psi' \in S$, $p \in \Psi$ iff $p \in \Psi'$,
2. for each $Qx.\psi \in cl(\varphi)$ and $\Psi, \Psi' \in S$, $Qx.\psi \in \Psi$ iff $Qx.\psi \in \Psi'$, where $Q \in \{\exists, \forall\}$,
3. for each $\exists x.\psi \in cl(\varphi)$ and $\Psi \in S$, $\exists x.\psi \in \Psi$ iff $\psi \in \Psi'$ for some $\Psi' \in S$.

▶ Remark. In the above definition, all atoms Ψ in S agree on the satisfaction of global atomic propositions (item 1) and sentences, i.e., formulae containing no free variables (item 2).

Let $\mathcal{S}$ denote the set of all macro states w.r.t. φ.

▶ **Definition 11** (Successor). We have the following definitions:

- Assume two atoms $\Psi, \Psi' \subseteq cl(\varphi)$. Then Ψ' is a *successor* of Ψ, denoted by $\Psi \to \Psi'$, if for each $X\psi \in cl(\varphi)$, $X\psi \in \Psi$ iff $\psi \in \Psi'$.
- Assume two macro states S and S' w.r.t. φ. Then S' is a successor of S if there is a *total and surjective* relation $\eta \subseteq S \times S'$ such that for each $(\Psi, \Psi') \in \eta$, Ψ' is a successor of Ψ. [Recall that $\eta \subseteq S \times S'$ is total (resp. surjective) iff for each $\Psi \in S$ (resp. $\Psi' \in S'$), there is $\Psi' \in S'$ (resp. $\Psi \in S$) such that $(\Psi, \Psi') \in \eta$.]

  For any two macro states S and S', we write $S \overset{\eta}{\rightarrowtail} S'$ to highlight the relation η associated with the transition.

The pairs $(\mathcal{A}, \to)$ and $(\mathcal{S}, \rightarrowtail)$ constitute a directed graph and multigraph respectively. Let's first give some intuition of our decision procedure. One may easily observe: Let $trc = (\alpha, I, (\beta_i)_{i\in I})$ be a computation trace satisfying φ. For each $j \in \mathbb{N}$ and $i \in I$, define $\Phi_{j,i} = \{\psi \in cl(\varphi) \mid (trc, [x \to i], j) \vDash \psi\}$. Then for each $j \in \mathbb{N}$, the tuple of atoms $(\Phi_{j,i})_{i\in I}$ can be *abstracted* into a macro state S_j (which is a set of atoms), and trc is accordingly abstracted into an infinite path $S_0S_1\ldots$ in $(\mathcal{S}, \rightarrowtail)$. From this observation, a natural idea to decide the satisfiability of φ is to search for a path in $(\mathcal{S}, \rightarrowtail)$ which satisfies some constraints (as obviously not all such paths give a valid computation trace). However, it seems nontrivial to specify these constraints. We use the following example to illustrate the difficulties.

▶ **Example 12.** Suppose $\varphi = G(\exists x.(p(x) \wedge XG\neg p(x)))$ (for simplicity, let $\xi = p(x) \wedge XG\neg p(x)$). It is not hard to see that φ is *unsatisfiable* (an obvious "model" of φ requires *infinitely* many process identifiers). The closure of φ is

$$\begin{aligned} cl(\varphi) \quad = \quad & \{p(x), \neg p(x), G\neg p(x), Fp(x), XG\neg p(x), XFp(x),\ \xi,\ \overline{\xi}, \\ & \exists x.\ \xi,\ \forall x.\ \overline{\xi},\ G(\exists x.\ \xi), F(\forall x.\ \overline{\xi}), XG(\exists x.\ \xi), XF(\forall x.\ \overline{\xi})\}. \end{aligned}$$

Let $S = \{\Psi_1, \Psi_2, \Psi_3\}$, where

$$\begin{aligned} \Psi_1 \quad &= \quad \{p(x), Fp(x), XG\neg p(x), \xi,\ \exists x.\ \xi,\ G(\exists x.\ \xi), XG(\exists x.\ \xi)\}, \\ \Psi_2 \quad &= \quad \{p(x), Fp(x), XFp(x),\ \overline{\xi},\ \exists x.\ \xi, G(\exists x.\ \xi), XG(\exists x.\ \xi)\}, \\ \Psi_3 \quad &= \quad \{\neg p(x), G\neg p(x), XG\neg p(x),\ \overline{\xi},\ \exists x.\ \xi, G(\exists x.\ \xi), XG(\exists x.\ \xi)\}. \end{aligned}$$

It is a routine to check that S satisfies all the constraints in Definition 10 and is thus a macro state. In addition, let $\eta = \{(\Psi_1, \Psi_3), (\Psi_2, \Psi_1), (\Psi_2, \Psi_2), (\Psi_3, \Psi_3)\}$. It is easy to verify that each pair of atoms in η satisfies the successor relation between atoms (item 1 of Definition 11), moreover, η is total and surjective, and thus $S \overset{\eta}{\rightarrowtail} S$. Hence $\pi = S \overset{\eta}{\rightarrowtail} S \overset{\eta}{\rightarrowtail} S\ldots$ is an infinite path in $(\mathcal{S}, \rightarrowtail)$. Moreover, since both $Fp(x)$ and $p(x)$ occur in Ψ_1 and Ψ_2, and $Fp(x)$ does not occur in Ψ_3, it is not hard to observe that over every path of atoms in π, that is, every infinite path in Figure 2, all the occurrences of $Fp(x)$ on the path are fulfilled. Therefore, to decide the satisfiability it is far from enough to simply search for a lasso in $(\mathcal{S}, \rightarrowtail)$ where over every path of atoms, all the occurrences of the "until" formulae are fulfilled, as that would lead to the wrong conclusion that φ is satisfiable. ◀

The unsatisfiability of φ in Example 12 is due to the fact that the "satisfaction" of φ requires infinitely many process identifiers (recall that as a model, we require the set of process identifier to be *finite*). To rule out such cases, we introduce a concept called *potentially Eulerian*.

Suppose $G = (V, E)$ is a directed multigraph. An *Eulerian cycle* in G is a cycle in G that traverses each arc in E exactly once. A directed multigraph G is *Eulerian* if it has

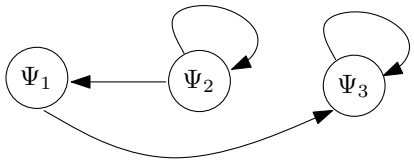


Figure 2 The directed graph (S, η) in Example 12.

an Eulerian cycle. For $v \in V$, let $\mathsf{indeg}(v)$ and $\mathsf{outdeg}(v)$ denote respectively the number of incoming arcs of v (i.e. the arcs with v as the head) and the number of outgoing arcs of v (i.e. the arcs with v as the tail) in G.

▶ **Proposition 13** ([3]). *Let $G = (V, E)$ be a directed multigraph. Then G is Eulerian iff G is connected and for each vertex $v \in V$, $\mathsf{indeg}(v) = \mathsf{outdeg}(v)$.*

▶ **Definition 14** (Potentially Eulerian)**.** A directed multigraph $G = (V, E)$ is said to be potentially Eulerian if G can become Eulerian by adding parallel arcs.

Note that in the above definition, only parallel arcs can be added. For instance, let $G = (\{v_1, v_2, v_3\}, \{(v_1, v_2), (v_2, v_1), (v_2, v_3), (v_3, v_1)\})$, then G is not Eulerian, but G is potentially Eulerian since adding a parallel arc (v_1, v_2) makes G Eulerian.

▶ **Proposition 15.** *Let $G = (V, E)$ be a directed multigraph. Then G is potentially Eulerian iff G is strongly connected.*

▶ **Example 16** (Example 12 continued)**.** Let G be the directed graph (S, η) in Example 12, that is, vertices are the atoms in S, and the arc relation is given by η (see Figure 2). Then it is easy to check that G is connected but not strongly connected, that is, G is *not* potentially Eulerian. ◀

Example 16 illustrates that the concept of "potentially Eulerian" may be used to deal with the situation that the satisfaction of φ requires infinitely many process identifiers, which is indeed the case in our construction, as we shall see.

Another difficulty is to formulate a proper constraint to guarantee all occurrences of "until" formulae are fulfilled somehow (which would be much easier for LTL). One natural candidate might be to require that over each path of atoms in the desired lasso of $(\mathcal{S}, \twoheadrightarrow)$, each "until" formula occurring in the path is fulfilled at least once. However, it turns out this would be too restrictive (see Example 19), and indeed we introduce a mechanism to relax this constraint; cf. $L(\cdot)$ function in Definition 17 and Definition 18, item 4.

In the following, we construct an arc-labelled directed graph G_φ from $(\mathcal{S}, \twoheadrightarrow)$ so that searching for a desired lasso in $(\mathcal{S}, \twoheadrightarrow)$ (for the satisfiability of φ) can be reduced to a reachability problem in G_φ.

▶ **Definition 17** (The graph G_φ)**.** The arc-labelled directed graph $G_\varphi = (V_\varphi, E_\varphi)$ is constructed from $(\mathcal{S}, \twoheadrightarrow)$ as follows:

- V_φ is the union of $\mathcal{S}$ and the set of tuples (S, S', G) such that $S, S' \in \mathcal{S}$, and $G = (S \cup S', E, L)$ is an arc-labelled directed graph such that $E \subseteq S \times S'$ and $L : E \to 2^{\mathcal{A} \times \mathcal{A}}$.
- E_φ is the union of
  - the set of tuples (S, η, S') such that $S \overset{\eta}{\twoheadrightarrow} S'$,
  - the set of tuples $(S, \eta, (S, S', G))$ such that $S \overset{\eta}{\twoheadrightarrow} S'$, and $G = (S \cup S', \eta, L)$, where for each $(\Psi, \Psi') \in \eta$, $L((\Psi, \Psi')) = \{(\Psi, \Psi')\}$,
  - the set of tuples $((S, S', G), \eta, (S, S'', G'))$ with

- $S' \overset{\eta}{\rightarrowtail} S''$,
- let $G = (S \cup S', E, L)$, then $G' = (S \cup S'', E \cdot \eta, L')$, where for each $(\Psi, \Psi'') \in E \cdot \eta$ (note that this implies that there exists some $\Psi' \in S'$ such that $(\Psi, \Psi') \in E$ and $(\Psi', \Psi'') \in \eta$)

$$L'((\Psi, \Psi'')) = \bigcup_{\Psi' \in S', (\Psi, \Psi') \in E, (\Psi', \Psi'') \in \eta} L((\Psi, \Psi')) \cup \{(\Psi', \Psi'')\}.$$

We explain the intuition of the arc labeling function $L(\cdot)$ in G as follows: Suppose π is a path from S to S' in $(\mathcal{S}, \rightarrowtail)$, and accordingly the vertex (S, S', G) with $G = (S \cup S', E, L)$ is reached from S in G_φ when going along π, then for each arc $(\Psi, \Psi') \in E$, $L((\Psi, \Psi'))$ is the set of all the possible arcs (Ψ'', Ψ''') on the paths from Ψ to Ψ' in the subgraph over the set of atoms induced by π.

Our goal is to reduce the satisfiability problem of φ to a reachability problem in G_φ, more specifically, to decide whether a vertex (S, S', G) satisfying some proper constraints in G_φ can be reached from a vertex S_0 that contains φ. In order to specify these constraints, we need another notation.

Given $G = (S \cup S', E, L)$ where $E \subseteq S \times S'$ and $L : E \to 2^{\mathcal{A} \times \mathcal{A}}$, and $(\Psi, \Psi') \in E$, the directed graph $G_{L((\Psi, \Psi'))}$ is defined by taking the set of atoms appearing in $L((\Psi, \Psi'))$ as the set of vertices and $L((\Psi, \Psi'))$ as the set of arcs (To put in a different way, $G_{L((\Psi, \Psi'))}$ is the graph corresponding to the relation given by $L((\Psi, \Psi')) \subseteq \mathcal{S} \times \mathcal{S}$). In addition, for a connected component C of G, a directed graph $G_{C,L}$ is defined as the *union* of the directed graphs $G_{L((\Psi, \Psi'))}$, where (Ψ, Ψ') is an arc in C.

▶ **Definition 18** (φ-witnessing path)**.** A path in G_φ is *φ-witnessing* if it is of the form

$$\underbrace{S_0 \overset{\eta_1}{\rightarrowtail} S_1 \cdots \overset{\eta_{m-1}}{\rightarrowtail} S_{m-1} \overset{\eta_m}{\rightarrowtail} S_m}_{\pi_1} \underbrace{\overset{\eta'_1}{\rightarrowtail} (S_m, S'_1, G_1) \overset{\eta'_2}{\rightarrowtail} \cdots \overset{\eta'_{n-1}}{\rightarrowtail} (S_m, S'_{n-1}, G_{n-1}) \overset{\eta'_n}{\rightarrowtail} (S_m, S'_n, G_n)}_{\pi_2}$$

and satisfies the following conditions. Let $G_n = (S_m, E_n, L_n)$.

1. There exists some $\Psi \in S_0$ such that $\varphi \in \Psi$,
2. $S_m = S'_n$,
3. each connected component C of G_n is potentially Eulerian (i.e. strongly connected),
4. each connected component C of G_n satisfies that for each $\psi_1 U \psi_2 \in cl(\varphi)$, if $\psi_1 U \psi_2$ occurs in some atom in G_{C,L_n}, then ψ_2 occurs in some atom in G_{C,L_n} as well.

We use the following example to illustrate the concept of φ-witnessing paths.

▶ **Example 19.** Consider the formula $\varphi = G(\exists x.(p'(x) \wedge XF\neg p'(x)))$ (For convenience, let $\xi = p'(x) \wedge XF\neg p'(x)$).

$$\begin{aligned} cl(\varphi) \quad = \quad & \{p'(x), \neg p'(x), F\neg p'(x), Gp'(x), XF\neg p'(x), XGp'(x), \xi, \ \overline{\xi}, \ \exists x.\xi, \forall x.\overline{\xi}, \\ & G(\exists x.\ \xi), F(\forall x.\ \overline{\xi}), XG(\exists x.\ \xi), XF(\forall x.\ \overline{\xi})\}. \end{aligned}$$

Let

$$\begin{aligned} \Psi_1 \quad = \quad & \{p'(x), F\neg p'(x), XF\neg p'(x), \xi, \exists x.\xi, G(\exists x.\ \xi), XG(\exists x.\xi)\}, \\ \Psi_2 \quad = \quad & \{\neg p'(x), F\neg p'(x), XF\neg p'(x), \overline{\xi}, \exists x.\xi, G(\exists x.\ \xi), XG(\exists x.\xi)\}, \end{aligned}$$

and $S_1 = \{\Psi_1, \Psi_2\}$, $S_2 = \{\Psi_1\}$, $\eta_1 = \{(\Psi_1, \Psi_1), (\Psi_2, \Psi_1)\}$, $\eta_2 = \{(\Psi_1, \Psi_1), (\Psi_1, \Psi_2)\}$. Suppose $\pi = S_1 \overset{\eta_1}{\rightarrowtail} (S_1, S_2, G_1) \overset{\eta_2}{\rightarrowtail} (S_1, S_1, G_2)$, where $G_1 = (\{\Psi_1, \Psi_2\}, \eta_1, L_1)$, $G_2 = (\{\Psi_1, \Psi_2\}, \eta_1 \cdot \eta_2, L_2)$,

- $L_1((\Psi_1, \Psi_1)) = \{(\Psi_1, \Psi_1)\}$ and $L_1((\Psi_2, \Psi_1)) = \{(\Psi_2, \Psi_1)\}$,
- $L_2((\Psi_1, \Psi_1)) = \{(\Psi_1, \Psi_1)\}$, $L_2((\Psi_2, \Psi_2)) = \{(\Psi_2, \Psi_1), (\Psi_1, \Psi_2)\}$, moreover, $L_2(e) = \{(\Psi_1, \Psi_1), e\}$ for $e = (\Psi_1, \Psi_2), (\Psi_2, \Psi_1)$.

Then π is a path in G_φ. Moreover, we notice that G_2 satisfies the following conditions: 1) G_2 is strongly connected; 2) let C be the unique connected component of G_2 (that is, G_2 itself), then $F\neg p'(x)$ occurs in G_{C,L_2}, and $\neg p'(x)$ occurs in G_{C,L_2} as well. Therefore, π is a φ-witnessing path in G_φ.

On the other hand, π does *not* satisfy the constraint that over every path of atoms in π, $F\neg p'(x)$ is fulfilled at least once: $\Psi_1\Psi_1\Psi_1$ is a path of atoms in π, $F\neg p'(x)$ occurs everywhere on the path, but $\neg p'(x)$ never appears. This justifies to some extent the use of the labelling function L in our construction, which facilitates a less restrictive constraint than requiring that over every path of atoms, each "until" formula occurring in the path is fulfilled at least once. ◀

The following lemma paves the way to the complexity upper bound and is crucial.

▶ **Lemma 20.** *φ is satisfiable iff there is a φ-witnessing path in G_φ.*

▶ **Theorem 21.** *The satisfiability problem of* $\mathsf{NN-ILTL}$, $\mathsf{NN-ILTL}(X, F, G)$, $\mathsf{NN-ILTL}\backslash X$, *and* $\mathsf{NN-ILTL}(F, G)$ *is EXPSPACE-complete.*

By Lemma 20, the satisfiability of $\mathsf{NN-ILTL}$ can be reduced to the reachability problem in G_φ. Then the EXPSPACE upper bound in Theorem 21 follows from the fact that G_φ is a directed graph of doubly exponential size in $|\varphi|$ and the Savitch's theorem [2]. For the lower bound, we strengthen the lower bound in [16, 20] by providing a reduction from the exponential size corridor tiling problem to the satisfiability problem of $\mathsf{NN-ILTL}(F, G)$, with the help of Lemma 4.

5 Conclusion

In this paper, we have drawn a relatively complete picture on the decidability and complexity of the satisfiability of various fragments of ILTL. To the best of our knowledge, this is the first systematic work on the satisfiability of indexed temporal logics. We believe that these results will deepen the understanding of the meta-properties of this class of logics, and will be instrumental for, e.g., parameterised model checking.

Future work includes investigating satisfiability for indexed branching-time temporal logics like indexed CTL and CTL^*, and other meta-property including expressive power. It is also interesting to see whether some techniques can be applied to the extensions of temporal logics with data variable quantifications.

References

1 B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI'14*, p. 262–281. 2014.

2 S. Arora and B. Barak, Computational Complexity: A Modern Approach. CUP, 2009.

3 J. Bang-Jensen and G. Z. Gutin. *Digraphs: Theory, Algorithms and Applications*. 2008.

4 P. van Emde Boas. The convenience of tilings. In *Complexity, Logic, and Recursion Theory*, p. 331–363, 1997.

5 M. C. Browne, E. M. Clarke, and O. Grumberg. Reasoning about networks with many identical finite state processes. *Information and Computation*, 81(1):13–31, 1989.

6 E. Clarke, M. Talupur, T. Touili, H. Veith. Verification by network decomposition. In *CONCUR 2004*. p. 276–291, 2004.

7 N. Decker, P. Habermehl, M. Leucker, and D. Thoma. Ordered navigation on multi-attributed data words. In *CONCUR'14*, p. 497–511, 2014.

8 S. Demri, D. Figueira, and M. Praveen. Reasoning about data repetitions with counter systems. In *LICS'13*, p. 33–42, 2013.

9 S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. *ACM Transations on Computational Logic*, 10(3), 2009.

10 Orna Grumberg, Orna Kupferman and Sarai Sheinvald. A Game-Theoretic Approach to Simulation of Data-Parameterized Systems. In *ATVA'14*, p. 348–363, 2014.

11 E. A. Emerson and V. Kahlon. Reducing model checking of the many to the few. In *CADE'00*, p. 236–254, 2000.

12 E. A. Emerson and K. S. Namjoshi. On reasoning about rings. *International Journal of Foundations of Computer Science*, 14(4):527–550, 2003.

13 E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2):105–131, 1996.

14 E. A. Emerson and J. Srinivasan. A decidable temporal logic to reason about many processes. In *PODC'90*, p. 233–246, 1990.

15 D. Figueira. Alternating register automata on finite words and trees. *Logical Methods in Computer Science*, 8(1), 2012.

16 S. M. German and A. P. Sistla. Reasoning about systems with many processes. *Journal of the ACM*, 39(3):675–735, 1992.

17 O. Grumberg, O. Kupferman, and S. Sheinvald. Model checking systems and specifications with parameterized atomic propositions. In *ATVA'12*, p. 122–136, 2012.

18 O. Grumberg, O. Kupferman, and S. Sheinvald. An automata-theoretic approach to reasoning about parameterized systems and specifications. In *ATVA'13*, p. 397–411, 2013.

19 J. Y. Halpern and M. Y. Vardi. The complexity of reasoning about knowledge and time. In *STOC'86*, p. 304–315, 1986.

20 J. Y. Halpern and M. Y. Vardi. The complexity of reasoning about knowledge and time. I. lower bounds. *Journal Computer and System Sciences*, 38(1):195–237, 1989.

21 A. Kara, T. Schwentick, and T. Zeume. Temporal logics on words with multiple data values. In *FSTTCS'10*, p. 481–492, 2010.

22 J. H. Reif and A. P. Sistla. A multiprocess network logic with temporal and spatial modalities. In *ICALP'83*, p. 629–639, 1983.

23 A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

24 A. P. Sistla and L. D. Zuck. Reasoning in a restricted temporal logic. *Inf. Comput.*, 102(2):167–195, 1993.

25 F. Song and Z. Wu. Extending temporal logics with data variable quantifications. In *FSTTCS'14*, p. 253–265, 2014.

26 M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. Comput. Syst. Sci.*, 32(2):183–221, 1986.

27 K. Y. Rozier and M. Y. Vardi. A Multi-encoding Approach for LTL Symbolic Satisfiability Checking. In*FM'11*, p. 417–431, 2011.

28 J. Li, L. Zhang, G. Pu, M. Y. Vardi and J. He. LTL Satisfiability Checking Revisited. In*TIME'13*, p. 91–98, 2013.

29 J. E. Hopcroft and J. D Ullman. Introduchon to Automata Theory, Languages, and Computation (Second edition). Addison-Wesley, Mass, 2000.

Expressiveness and Complexity Results for Strategic Reasoning

Julian Gutierrez, Paul Harrenstein, and Michael Wooldridge

Department of Computer Science, University of Oxford
Wolfson building, Parks road, Oxford, United Kingdom

Abstract

This paper presents a range of expressiveness and complexity results for the specification, computation, and verification of Nash equilibria in multi-player non-zero-sum concurrent games in which players have goals expressed as temporal logic formulae. Our results are based on a novel approach to the characterisation of equilibria in such games: a semantic characterisation based on *winning strategies* and *memoryful reasoning*. This characterisation allows us to obtain a number of other results relating to the analysis of equilibrium properties in temporal logic.

We show that, up to bisimilarity, reasoning about Nash equilibria in multi-player non-zero-sum concurrent games can be done in ATL* and that constructing equilibrium strategy profiles in such games can be done in 2EXPTIME using finite-memory strategies. We also study two simpler cases, two-player games and sequential games, and show that the specification of equilibria in the latter setting can be obtained in a temporal logic that is weaker than ATL*. Based on these results, we settle a few open problems, put forward new logical characterisations of equilibria, and provide improved answers and alternative solutions to a number of questions.

1998 ACM Subject Classification F.4.1 Mathematical logic – Temporal logic, F.3.1 Specifying and Verifying and Reasoning about Programs, I.2.11 Distributed Artificial Intelligence – Multiagent systems

Keywords and phrases Temporal logic, Nash equilibrium, Game models, Formal Verification

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.268

1 Introduction

In the last decade, there has been an increasing interest in game-theoretic models of concurrent and distributed systems as well as in temporal logic formalisms to reason about the complex behaviour of such systems. These logical formalisms are used to reason about linear-time or branching-time properties of the systems, and usually extend logics such as LTL or CTL. The alternating-time temporal logic ATL* of Alur, Henzinger, and Kupferman [1] is an extension of CTL* that is intended to support, in addition, reasoning about strategic behaviour. ATL* replaces the path quantifiers of CTL* with *agent/strategy* modalities that can be used to express properties of coalitions of agents in game-like concurrent and multi-agent systems. ATL* is probably the most popular and widely used temporal logic for *strategic reasoning* in computer science, artificial intelligence, and multi-agent systems research [8, 16].

The theory of ATL* is well understood. In particular, it is known that satisfiability and model checking for ATL* are 2EXPTIME-complete [1, 15] – theoretically better than for many of its extensions, where these problems are non-elementary, or even undecidable [3, 10, 11]. ATL* is a natural logic to reason about winning strategies in game-like systems, which makes it sufficiently powerful for many computational purposes. For instance, several design and verification problems can be reduced to questions about the existence and computation of

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 268–282

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

winning strategies [1]. In fact, from a game-theoretic point of view, these problems are usually reduced to a rather simple setting: two-player zero-sum sequential games [17].

However, when considering multi-player non-zero-sum concurrent games, the notion of a winning strategy does not seem to be the most appropriate analytical concept [2]. In such settings, we need richer game-theoretic solution concepts, such as Nash equilibrium [14]. However, at first sight, ATL* may not seem to be appropriate to reason about solution concepts such as Nash equilibrium, and in fact, it seems to be widely assumed that winning strategies cannot be used to study equilibria in multi-player non-zero-sum games.

Indeed, in the quest for a formalism expressive enough to capture game-theoretic equilibria, *many* temporal logics have been developed, including: Strategy Logic (SL [3, 11]); an extension of ATL with strategy contexts (ATL$_{sc}$ [10]); Quantified CTL* (QCTL* [9]); and Coordination Logic (CL [4]). All these logical formalisms, however, have undecidable satisfiability and non-elementary model checking problems in the general case. These undesirable properties present substantial challenges from a computational point of view when applying these temporal logics for synthesis or verification purposes. In fact, only very few practical model checking tools support logical specifications with formalisms more powerful than ATL*.

For some of the reasons given above, a number of studies have been conducted, partly because better results can be obtained in restricted scenarios, for instance, when considering sequential (also called turn-based) games or two-player games. For example, it is known that with respect to two-player sequential games, Nash equilibria can be expressed in a fragment of SL that is as expressive as ATL*; see [3] for further details. As a consequence, reasoning about Nash equilibria in this class of games can be done in 2EXPTIME. With respect to multi-player sequential games, Nash equilibria can be expressed in both ATL$^*_{sc}$ and Quantified CTL* while keeping the logics decidable [9]. Model checking is still non-elementary. In addition, until recently, the simplest fragment of SL known to be powerful enough to express Nash equilibria was the Boolean goal fragment [12], which may require, so-called, non-behavioural strategies [12] to realise a profile of strategies.[1] Such a fragment is not known to have a decidable satisfiability problem [11], but can be used to model check Nash equilibrium properties in 2EXPTIME given the characterisation of Nash equilibria in SL. These are all very interesting results, which suggest that further work should be done.

In this paper, we will show that under a suitable set of necessary and sufficient conditions, *ATL* can be used to reason about the existence of various kinds of equilibria* in multi-player non-zero-sum concurrent games. In particular, we will be interested in how to:

1. verify if a given game has a Nash equilibrium (NE model checking);
2. check if there is a game where a desired Nash equilibrium exists (NE satisfiability);
3. synthesise a profile of strategies that realises a particular Nash equilibrium;
4. express Nash equilibria in a number of types of games.

Let us now explain how ATL* can be used to address the above questions. We say that a *property* can be *expressed* in a given logic L if there is a formula ϕ of L such that ϕ is satisfied in all and only the models having such a property. Moreover, we say that a *problem* can be *characterised* in a given logic L if every instance of the problem can be specified in L. Letting P be the property of having a Nash equilibrium (formally defined later on), we show that using ATL* one can both express P in multi-player sequential games and characterise the NE model checking and NE satisfiability problems for these games. In fact, we can do so

[1] It was recently discovered how to express Nash equilibria in a fragment of SL that is simpler than the Boolean goal fragment and which can be given an interpretation in terms of behavioural strategies.

using a logic that is strictly weaker than ATL*. Moreover, we show that using ATL* one can characterise the NE satisfiability problem for multi-player concurrent games and the NE model checking problem for a subclass of games that we call Moore with deterministic past.

An interesting aspect of our results with respect to ATL* is that the class of Moore games with deterministic past is complete for the whole family of multi-player concurrent games in the following sense: every formula ϕ of ATL* is satisfiable if and only if it is satisfied by some model representing a game that is Moore with deterministic past. Then, Moore games with deterministic past can be seen as canonical within the class of multi-player concurrent games and therefore be used as an underlying model with respect to which define games and synthesise profiles of strategies. Using these results we can, in the end, perform a computational analysis of Nash equilibrium properties for various types of games.

Our results build on a novel semantic characterisation of Nash equilibria which combines the concepts of *winning strategies* and *memoryful reasoning*. Informally, in this paper we use techniques from repeated game theory (more specifically, punishment strategies as used in the literature about the Folk Nash theorems for repeated games [5, 14]) and combine them with recent developments in the study of strategic reasoning in logic and computer science: a memoryful extension of ATL* called mATL* [13]. Our results also indicate why an ATL* specification that allows one to reason about the existence of Nash equilibrium may be so hard to find: it can be obtained via a non-elementary reduction from mATL*, which was recently shown to be as expressive as ATL*, a result that partly builds on quite sophisticated automata-theoretic techniques developed for a memoryful extension of CTL* [7].

Based on what is currently known about some of the computational properties of mATL*, (for instance, 2EXPTIME satisfiability, synthesis, and model checking problems as well as a non-elementary reduction to ATL*), we are able to obtain a number of subsequent results. We also studied the two important special cases of two-player games and sequential games. In the latter setting, we show that the existence of Nash equilibria can be expressed in m$^-$ATL* [13], a (memoryful) temporal logic that is known to be strictly weaker than ATL*, which improves previous results in the literature [3, 9] and shows that concurrent behaviour, even in the two-player case, appears to complicate matters considerably.

Moreover, given the translation from mATL* to ATL*, we show that Nash equilibrium can be realised with respect to finite-memory strategies [15], which in turn also answers a question about the nature of Nash equilibrium strategy profiles and improves other results in the literature [12]. Finally, we extend all of our main findings to strong Nash equilibrium, a game-theoretic solution concept designed to deal with coalitions of players in a game – a very natural situation in concurrent and multi-agent scenarios. Interestingly, the results for this more complex game-theoretic setting are obtained using virtually the same techniques as in the case for Nash equilibria, where only single-player deviations are considered.

Finally, the table below summarises some of the results mentioned before, *i.e.*, the logics that can be used to study NE satisfiability and NE model checking for each type of games.

	2P-SG	2P-CG	MP-SG	MP-CG
NE satisfiability	m$^-$ATL*	ATL*	m$^-$ATL*	ATL*
NE model checking	m$^-$ATL*	SL	m$^-$ATL*	SL

Notation: 2P (2-player), MP (multi-player), SG (sequential game), CG (concurrent game).

Structure of the paper. Section 2 introduces the models, logics, and games that we will consider throughout the paper. Section 3 presents the semantic characterisation of equilibria

in terms of punishment strategies, and Section 4 defines Moore games with deterministic past. Section 5 shows how to construct an mATL* specification of the semantic characterisation of equilibria given before in order to solve NE satisfiability, and Section 6 studies the special cases of sequential games and two-player games. Section 8 concludes the paper.

2 Preliminaries

Sets. Given any set $S = \{s, q, r, \ldots\}$, we use S^*, S^ω, and S^+ for, respectively, the sets of finite, infinite, and non-empty finite sequences of elements in S. If $w_1 = s's'' \ldots s^k \in S^*$ and w_2 is any other (finite or infinite) sequence, we write $w_1 w_2$ for their concatenation $s's'' \ldots s^k w_2$. For $Q \subseteq S$, we write S_{-Q} for $S \setminus Q$ and S_{-i} if $Q = \{i\}$. We extend this notation to tuples $u = (s_1, \ldots, s_k, \ldots, s_n)$ in $S_1 \times \ldots \times S_n$, and write u_{-k} for $(s_1, \ldots, s_{k-1}, s_{k+1}, \ldots, s_n)$, and similarly for sets of elements, that is, by u_{-Q} we mean u without each s_k, for $k \in Q$. Given a sequence w, we write $w[t]$ for the element in position $t+1$ in the sequence; for instance, $w[0]$ is the first element of w. We also write $w[l \ldots m]$ for the sequence $w[l] \ldots w[m]$, and $w[l \ldots m)$ for $w[l] \ldots w[m-1]$; if $m = 0$, we let $w[l \ldots m)$ be the empty sequence, denoted ϵ.

Games. Let $\mathrm{Ag} = \{1, \ldots, n\}$ be a set of *players* and St a set of *states*. For each player $i \in \mathrm{Ag}$ we have a set of actions Ac_i and with every state s and player i we associate a set $\mathrm{Ac}_i(s) \subseteq \mathrm{Ac}_i$ of *actions* that i can perform at s. We write Ac for $\bigcup_{i \in \mathrm{Ag}} \mathrm{Ac}_i$ and assume that $\bigcup_{i \in \mathrm{Ag}} \mathrm{Ac}_i$ is a partition of Ac. We call a profile of actions $(a_1, \ldots, a_n) \in \mathrm{Ac}_1 \times \cdots \times \mathrm{Ac}_n$ a *direction*, and denote it by d. We let D be the set of directions – also called *decisions* – with respect to Ac, and write d_i for the a_i of d that is in Ac_i. Furthermore, we have a (deterministic) transition function $\delta \colon \mathrm{St} \times \mathrm{Ac}_1 \times \cdots \times \mathrm{Ac}_n \to \mathrm{St}$, which indicates the system transitions when $d = (a_1, \ldots, a_n)$ is performed at a state s. A state s' is *accessible* from another state s whenever there is some $d = (a_1, \ldots, a_n)$ such that $\delta(s, a_1, \ldots, a_n) = s'$. A *run* is an infinite sequence $\rho = s_0 s_1 s_2 \ldots$ such that for every $t \geq 0$ we have that s_{t+1} is accessible from s_t. The set of runs is denoted by R. By a *(finite) history* we mean a finite sequence $\pi = s_0 s_1 s_2 \ldots s_k$ of accessible states. By $\mathit{prefix}(\rho)$ we denote the set of finite prefixes of ρ, *i.e.*, $\mathit{prefix}(\rho) = \{\pi \in \mathrm{St}^* : \rho = \pi\rho' \text{ for some } \rho' \in \mathrm{St}^\omega\}$. Given $\pi \in \mathrm{St}^+$, by $\mathit{last}(\pi)$ we denote the last state in π, *i.e.*, if $\pi = \pi' s$, then $\mathit{last}(\pi) = s$. By s^0 we denote the *initial state*.

A *strategy* for a player i is a function $f_i \colon \mathrm{St}^+ \to \mathrm{Ac}_i$ such that $f_i(\pi s) \in \mathrm{Ac}_i(s)$ for every $\pi \in \mathrm{St}^*$ and $s \in \mathrm{St}$. That is, a strategy for a player i specifies for every finite history π an action available to i in $\mathit{last}(\pi)$. The set of strategies for player i is denoted by F_i. A *strategy profile* is a tuple $\vec{f} = (f_1, \ldots, f_n)$ in $F_1 \times \cdots \times F_n$. Observe that given a state s and a transition function $\delta \colon \mathrm{St} \times \mathrm{Ac}_1 \times \cdots \times \mathrm{Ac}_n \to \mathrm{St}$, each strategy profile $\vec{f}$ defines a unique run ρ where $\rho[0] = s$ and $\rho[t+1] = \delta(\rho[t], f_1(\rho[0 \ldots t]), \ldots, f_n(\rho[0 \ldots t]))$, for all $t \geq 0$. We write $\rho(\vec{f}(s))$ for such a run, and simply $\rho(\vec{f})$ if $s = s^0$. Furthermore, each player i is assumed to have an associated dichotomous preference relation over runs, which is modelled as a subset Γ_i of the set of runs R. Intuitively, a player i strictly prefers all runs in Γ_i to those that are not in Γ_i and is indifferent otherwise. Thus, Γ_i represents the *goal* or *objective* of player i. We also write $\rho \succsim_i \rho'$ to indicate that player i weakly prefers run ρ to ρ' and $\rho \succ_i \rho'$ for player i strictly preferring ρ to ρ', *i.e.*, if $\rho \succsim_i \rho'$ but not $\rho' \succsim_i \rho$.

Now we are in a position to define the concept of a *Nash equilibrium strategy profile* as a strategy profile $\vec{f} = (f_1, \ldots, f_n)$ such that for all players i and all strategies g_i in F_i,

$$\rho(\vec{f}) \succsim_i \rho(\vec{f}_{-i}, g_i).$$

We say that a run ρ is *sustained by a Nash equilibrium strategy profile* if there is some Nash

equilibrium strategy profile $\vec{f}$ with $\rho = \rho(\vec{f})$. Then, we know that

$$\rho(\vec{f}_{-i}, g_i) \in \Gamma_i \text{ implies } \rho(\vec{f}) \in \Gamma_i.$$

A game is played by each player i selecting a strategy f_i with the aim that the induced run $\rho(\vec{f})$ belongs to its goal/objective set Γ_i. If $\rho(\vec{f}) \in \Gamma_i$ we say that i has its goal satisfied or achieves its objective. Otherwise, we say that i does not have its goal satisfied.

Memoryful Alternating Temporal Logics

Memoryful ATL* (mATL* [13]) is an extension of ATL* that allows for memoryful reasoning. At the syntactic level, mATL* simply adds a propositional variable *present* to the standard ATL* language. More specifically, given a set of atomic propositions AP and a set of agents Ag, the language of mATL* formulae is given by the following grammar:

$$\phi \quad ::= \quad \mathit{present} \mid p \mid \neg\phi \mid \phi \vee \phi \mid \mathbf{X}\,\phi \mid \phi\,\mathbf{U}\,\phi \mid \langle\langle C\rangle\rangle\,\phi$$

such that $p \in \mathrm{AP}$ and $C \subseteq \mathrm{Ag}$, and the formulae *present*, $\mathbf{X}$, and $\mathbf{U}$ are in the scope of $\langle\langle C\rangle\rangle$. We use the following abbreviations: we write $\top$ for $p \vee \neg p$, $\bot$ for $\neg\top$, $\mathbf{F}\,\phi$ for $\top\,\mathbf{U}\,\phi$, $\mathbf{G}\,\phi$ for $\neg\,\mathbf{F}\,\neg\phi$, $\mathbf{E}\,\phi$ for $\langle\langle \mathrm{Ag}\rangle\rangle\,\phi$, $\mathbf{A}\,\phi$ for $\langle\langle\emptyset\rangle\rangle\,\phi$, and $[\![C]\!]\,\phi$ for $\neg\,\langle\langle C\rangle\rangle\,\neg\phi$; we also use the conventional abbreviations for the other propositional logic operators. We write $\phi \in \mathcal{L}(\mathrm{AP}, \mathrm{Ag})$ if ϕ is an mATL* formula in this language. When either AP or Ag, or both, are known, we may omit them. With $\mathrm{AP}' \subseteq \mathrm{AP}$, we may write $\phi|_{\mathrm{AP}'}$ if $\phi \in \mathcal{L}(\mathrm{AP}', \mathrm{Ag})$ for some set of agents Ag.

The denotation of mATL* formulae is given by *concurrent game structures*. We let a tuple $M = (\mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, s^0, \lambda, \delta)$ be a concurrent game structure (CGS), where $\lambda : \mathrm{St} \to 2^{\mathrm{AP}}$ is a labelling function, and all other components of M are as defined before. The *size of* M is defined to be $|\mathrm{St}| \times |\mathrm{Ac}|^{|\mathrm{Ag}|}$. Moreover, in particular, we write R^*_s and R^ω_s for, respectively, the finite and infinite runs of M that start at state s. We simply write R^* and R^ω if $s = s^0$. Moreover, we will write $\vec{f}_C$ for $(f_i, \ldots, f_k)$, with $C = \{i, \ldots, k\} \subseteq \mathrm{Ag}$, that is, a joint strategy for the players in C. Similarly, we will write $\vec{g}_{-C}$ for a joint strategy for the players in Ag_{-C}. For simplicity, we will assume that all strategies are defined on all finite runs of M, and hence at all states. We define the set of *$\vec{f}_C$-runs from state s* to be $\{\rho' \in R^\omega_s : \rho' = \rho(\vec{f}_C(s), \vec{g}_{-C}(s))$ for some joint strategy $\vec{g}_{-C}$ for $\mathrm{Ag}_{-C}\}$.

We can now define the semantics of mATL* formulae based on the rules given below. Let M be a concurrent game structure, $\rho \in R^\omega$ be an infinite run, $\pi \in R^*$ be a finite run, and $t \in \mathbb{N}$ be a temporal index. The semantics of mATL* is defined by the following rules:

$\rho, t, \pi \models_M \mathit{present}$	iff	$\rho[0 \ldots t] = \pi$.
$\rho, t, \pi \models_M p$	iff	$p \in \lambda(\rho[t])$.
$\rho, t, \pi \models_M \neg\phi$	iff	$\rho, t, \pi \models_M \phi$ does not hold.
$\rho, t, \pi \models_M \phi \vee \phi'$	iff	$\rho, t, \pi \models_M \phi$ or $\rho, t, \pi \models_M \phi'$.
$\rho, t, \pi \models_M \mathbf{X}\,\phi$	iff	$\rho, t+1, \pi \models_M \phi$.
$\rho, t, \pi \models_M \phi\,\mathbf{U}\,\phi'$	iff	$\rho, t', \pi \models_M \phi'$, for some $t' \geq t$, and
		$\rho, k, \pi \models_M \phi$, for all $t \leq k < t'$.
$\rho, t, \pi \models_M \langle\langle C\rangle\rangle\,\phi$	iff	there is some $\vec{f}_C$ such that for all $\vec{f}_C$-runs ρ' from state $\rho[t]$,
		it is the case that $\rho[0 \ldots t)\rho', 0, \rho[0 \ldots t] \models_M \phi$ holds.

We say that M is a *model* of ϕ (denoted $M \models \phi$) if $\rho, 0, s^0 \models_M \phi$ for some $\rho \in R^\omega$. We also say that ϕ is *satisfiable* if $M \models \phi$ for some CGS M. Moreover, we say that ϕ is *equivalent* to ϕ' if $M \models \phi \iff M \models \phi'$ for all M. Finally, the *size of* ϕ is its number of subformulae.

The interpretation of the tense and boolean operators is, essentially, as for LTL. The difference is with respect to the *present* proposition and the *agent/strategy quantifier* $\langle\langle C\rangle\rangle$.

On the one hand, the atomic proposition *present* holds whenever the history of the current run under consideration (*i.e.*, the finite run $\rho[0 \dots t]$) is the same as the history of the run leading to the state where *present* last was true – that is, π. On the other hand, the agent quantifier does three important things: firstly, it considers the runs ρ' that are consistent with $\vec{f}_C$, but adds to them the history of play seen so far – thus, obtaining the run $\rho[0 \dots t)\rho'$; secondly, it resets the evaluation of temporal formulae by letting the temporal index t be 0; and, thirdly, it resets the history with respect to which the atomic proposition *present* will now hold – then, letting π be $\rho[0 \dots t]$. A few examples are in order (also see, *e.g.*, [13]).

Examples. Because the quantifiers of mATL* reset the temporal index t, CTL* formulae, such as $\mathbf{E}\,\mathbf{F}(p \Longrightarrow \mathbf{A}\,\mathbf{G}\,q)$, cannot be interpreted in the same way in mATL*: the subformula $\mathbf{G}\,q$ will be evaluated from $t = 0$ regardless of the value of t when p holds. There is a standard way to remedy this problem. Any subformula starting with a path quantifier can be evaluated "in the usual CTL* sense" in the following way: $M \models_{\text{CTL}^*} \mathbf{A}\,\phi$ if and only if $M \models \mathbf{A}\,\mathbf{F}(\mathit{present} \wedge \phi)$; and similarly for $\mathbf{E}$. Therefore, in order to be interpreted as a CTL* expression, the formula above can be rewritten as $\mathbf{E}\,\mathbf{F}(p \Longrightarrow \mathbf{A}\,\mathbf{F}(\mathit{present} \wedge \mathbf{G}\,q))$.

But, of course, mATL* is interesting because it can express, in a relatively simple way, properties that refer both to the past and to the future within a strategic environment. For instance, suppose an agent i has two objectives: satisfy LTL goal formula γ_1 and if i sees proposition p satisfied, then try to satisfy LTL goal formula γ_2 too. What is important is that both γ_1 and γ_2 are "goals" that were set from the beginning of the computation, not from the point where, potentially, proposition p is encountered. Expressing that i can ensure that its goals are satisfied can be done in the following way: $\langle\!\langle i \rangle\!\rangle(\gamma_1 \wedge \mathbf{G}(p \Longrightarrow \langle\!\langle i \rangle\!\rangle\, \gamma_2))$. Then, player i must be more careful when playing the game. It has to play in such a way that if p is seen while trying to satisfy γ_1, then it is in a position to satisfy γ_2 too. Seen as a formula in ATL* instead, the situation is rather different: while γ_1 has to hold from the beginning of the computation, γ_2 has to hold from the moment proposition p is seen.

Logical and Computational Properties. Using the abbreviations $\mathbf{E}$ and $\mathbf{A}$ and the atomic proposition *present* as explained in the first example, it can be shown that CTL* is a sublogic of mATL*. Thus, CTL and LTL are also sublogics of mATL*. Moreover, none of these logics is as powerful as mATL*, as might be expected. What is surprising is that, in fact, ATL* is as expressive as mATL*. However, the translation from mATL* to ATL* formulae is non-elementary [13]. An interesting case is the fragment of mATL* without the *present* proposition. Such a fragment, called m$^-$ATL* is not as expressive as ATL*, and incomparable with CTL/CTL*. The proof is simple and has to do with the fact that *present* is needed for mATL* to be able to capture "memoryless" properties such as those given by CTL* (cf. the first example above and [13]). Letting $L_1 < L_2$ mean that logic L_1 is strictly less expressive than logic L_2, and $L_1 \equiv L_2$ that L_1 is as expressive as L_2, we have the following:

$$\text{m}^-\text{ATL} < \text{mATL}^* \equiv \text{ATL}^* < \{\text{SL}, \text{CL}, \text{ATL}_{sc}, \text{QCTL}^*\}.$$

With respect to the complexity of mATL*, it is known that it has model checking, satisfiability and synthesis problems in 2EXPTIME, thus no harder than ATL*. These results sharply contrast with those for other logics such as Strategy Logic, ATL with strategy contexts, or Coordination Logic, in which satisfiability and synthesis are *undecidable* and model checking is *non-elementary* [4, 9, 11]. The complexity properties of logics for strategic reasoning present a substantial challenge for practical purposes. In fact, a very simple fragment of Strategy Logic – where Nash equilibrium can be specified – may require non-behavioural strategies as models [12], which is not the case in ATL* and mATL*.

Games with Temporal Logic Goals

Finally, we are in a position to define a game with temporal logic goals. Given a concurrent game structure $M = (\mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, s^0, \lambda, \delta)$ and a set of temporal logic formulae $(\gamma_i)_{i\in\mathrm{Ag}}$, a tuple $G = (M, (\gamma_i)_{i\in\mathrm{Ag}})$ is a *game with temporal logic goals*. We say that G is a game with LTL goals if each $\gamma_i \in \mathcal{L}(\mathrm{AP})$ is an LTL formula, and simply call G a game. The idea is that each $\gamma_i \in \mathcal{L}(\mathrm{AP})$ is a formula that succinctly represents the goal of player i, that is, γ_i is a logical representation of Γ_i. Then, we have that a run ρ satisfies goal γ_i if and only if $\rho \in \Gamma_i$.

3 From Winning Strategies to Nash Equilibria

In order to characterise the existence of Nash equilibrium in terms of winning strategies, we will introduce the concept of *punishment* strategies. These are a type of winning strategies that a group of players may want to use against a player j who wants to refrain from following a given sequence of actions – a situation that is commonly known as a deviation by j.

A finite history π is a *deviation* from a run ρ if $\pi = \pi' s$ for some state $s \in \mathrm{St}$ such that $\pi' \in \mathit{prefix}(\rho)$ and $\pi \notin \mathit{prefix}(\rho)$. We also use $\mathit{dev}(\rho)$ to refer to the set of deviations from ρ. A finite history π is a *deviation from a set* R if $\pi \in \mathit{dev}(\rho)$ for some $\rho \in R$ and $\pi \in \mathit{prefix}(\rho')$ for no $\rho' \in R$. The set of deviations from R is denoted by $\mathit{dev}(R)$.

If a run ρ is intended to be sustained by a Nash equilibrium strategy profile $\vec{f}$, punishment may be successful deterring agents from adopting deviating strategies – strategies that do not comply with ρ. However, if a deviation is observed, it is generally not necessary to punish all players but rather only those that the deviation can be attributed to in order to prevent such undesirable behaviour. We thus introduce the following formal concept of *attributability*.

▶ **Definition 1** (Attributability)**.** Let ρ be a run and $\pi \in \mathit{dev}(\rho)$ a deviation from ρ. Then, the deviation π is *attributable to a set C of players* if there are profiles $\vec{f} = (f_1, \ldots, f_n)$ and $\vec{g} = (g_1, \ldots, g_n)$ such that $\rho(\vec{f}) = \rho$ and $\pi \in \mathit{prefix}(\rho(\vec{f}_{-C}, \vec{g}_C))$. We also say that π is attributable to j for π being attributable to $C = \{j\}$. A deviation π is said to be *individual* if π is attributable to some player j. Moreover, a deviation π from ρ is said to be *uniquely attributable to C* if C is the only set that the deviation π can be attributed to. A deviation that is uniquely attributable to a player j (a set C) is called a *j-deviation* (a *C-deviation*).

The necessity of the concept of attributability is rather clear: in order to define punishable players and punishment strategies the least that one needs to know is who should be punished once a deviation arises. Some equally important information in order to define punishment strategies is the *history* of play so far, since players' goals are defined with respect to the beginning of the game. Then, at every stage of the game one must be able to "remember" the history of play so far – that is, we need some *memoryful* power within the game. In order to formalise the above notions and, in particular, to make precise how we will deal with memoryful reasoning later on, let us introduce some notations first.

Given a finite history π, each strategy profile $\vec{f}$ induces a unique run $\rho(\vec{f}, \pi) = s^0 s^1 s^2 \ldots$ defined inductively as follows, with $t \geq 0$,

$$s^t = \begin{cases} s^0 & \text{if } t = 0 \text{ and } \pi = \epsilon, \\ \delta(\mathit{last}(\pi), f_1(\pi), \ldots, f_n(\pi)) & \text{if } t = 0 \text{ and } \pi \neq \epsilon, \\ \delta(s^{t-1}, f_1(s^0 \ldots s^{t-1}), \ldots, f_n(s^0 \ldots s^{t-1})) & \text{otherwise.} \end{cases}$$

We omit ϵ in $\rho(\vec{f}, \epsilon)$ and simply write $\rho(\vec{f})$. Then, π can be seen as the history of play with which a strategy or a profile of strategies can be made to agree, so that it is taken into

account when playing beyond π. We are now in a position to formulate the central concept of this section, namely that of a player, or a set of players, being *punishable*.

▶ **Definition 2** (Punishable players, punishment strategies)**.** A set C of players is *punishable at run* ρ if there is a profile $\vec{f}^C = (f_1^C, \ldots, f_n^C)$ – also called a profile of *punishment strategies against* C – such that for every profile of strategies $\vec{g}_C$ of C and every deviation π from ρ that is uniquely attributable to C we have, for some $j \in C$,

$$\pi\rho((\vec{f}^C_{-C}, \vec{g}_C), \pi) \notin \Gamma_j.$$

One thing to observe in this definition is that a profile of punishment strategies may punish different players at different C-deviations. Moreover, as we will see, for a set C of players to be punishable, it is sufficient to punish only one player in C at each C-deviation. More importantly, Definition 2 shows the strong link between two important components: firstly, the need to consider/remember π and, secondly, the existence of a winning strategy $\vec{f}^C_{-C}$ for the set of players not in C against the goal of at least one of the players in C.

Thus, a careful analysis of the definition of punishability we proposed shows the following facts. That if one is interested in that every player j whose goal is not satisfied by ρ must be punishable, then one has to guarantee, in particular, that at *every* stage of the game those in the set of players in Ag_{-j} (i) must have a winning strategy against the goal of player j, and such a winning strategy (ii) must agree with ρ up to the time point when j deviates. These two conditions for a player j to be punishable are key to answering the next question:

When can a player j be guaranteed to be punishable
so that a desired Nash equilibrium can be rationally sustained?

Formally, punishability with respect to a desired run ρ can be fully characterised in terms of a number of winning strategies against the goal of j, as shown by the following lemma.

▶ **Lemma 3** (Punishability – semantic characterisation)**.** *Player j is* punishable at run ρ *if and only if for every deviation π from ρ that is uniquely attributable to j there is a profile $\vec{f}^\pi = (f_1^\pi, \ldots, f_n^\pi)$ such that for every strategy g_j of j we have that*

$$\pi\,\rho((\vec{f}^\pi_{-j}, g_j), \pi) \notin \Gamma_j.$$

The above lemma supports the claim that punishability can be understood as a simple combination of *winning strategies* (given by $\vec{f}^\pi_{-j}$ to achieve $\neg\gamma_j$) with *memory* (given by π). Because the concept of punishability will be central to the characterisation of the existence of Nash equilibrium strategy profiles, the above claim about winning strategies and memoryful reasoning can be naturally extended to the more complex concept of Nash equilibria.

Indeed, the manipulation of the three main concepts introduced in this section, namely *deviations*, *attributability*, and *punishment*, can be used to provide a semantic characterisation of the fundamental concept of Nash equilibrium. More importantly, they can be used to show that Nash equilibria can be characterised by the existence of punishment strategies, which, by definition, are winning strategies against players to which deviations can be attributed. Formally, the following result can be shown.

▶ **Lemma 4** (Nash equilibrium – semantic characterisation)**.** *Let ρ be a run and assume that all individual deviations from ρ are uniquely attributable. Then, ρ is sustained by a Nash equilibrium strategy profile if and only if all players j with $\rho \notin \Gamma_j$ are punishable at ρ.*

Lemma 4 shows that, provided that every j-deviation can be uniquely attributable to player j, the following condition is both necessary and sufficient for the existence of a Nash equilibrium ρ: that *every player j whose goal is not satisfied by ρ must be punishable.*

The *necessary* and *sufficient* conditions for the existence of Nash equilibria are both formulated in terms of our notion of punishability (Lemma 4). To be able to punish a deviating player, however, a coalition also has to keep track of the run that leads to the deviation. This idea could informally be summarised as follows:

$$\textbf{Nash equilibrium} = \textbf{Punishability} + \textbf{Memory}$$

The concept of punishability is closely related to that of a winning strategy. Thus, the above results prepare the ground for the characterisation of Nash equilibrium in temporal logics like mATL* that can reason about both *winning strategies* and *memoryfulness*.

4 Model Transformations

In order to logically describe the semantic characterisation of Nash equilibria presented in Section 3, we first define two structure-preserving transformations on CGSs, which can be used to map any CGS into another CGS in a desired canonical form. The first map is used to transform a CGS into a CGS *with deterministic past*. The second map is used to transform a CGS M into a CGS M' where all information about players' choices in M is explicitly given via labels in the states of M'. This transformation is similar to the well known translation used to obtain Moore from Mealy machines. For this reason, we call such a translation a *Moore transformation*, and the resulting CGS M' a *Moore* CGS.

Let us now formally define the class of CGS with deterministic past. A CGS M is said to be a CGS with deterministic past if whenever $\delta(s', d') = s$ and $\delta(s'', d'') = s$ then $d' = d''$. The first map above mentioned, namely $\mathbf{detp} : \mathcal{M} \to \mathcal{M}$, called a *deterministic-past map*, transforms a CGS M into a CGS M' which can be shown to be a CGS *with deterministic past*. The map **detp** is defined as follows. Given $M \in \mathcal{M}$, where $M = (\mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, s^0, \lambda, \delta)$, the structure $\mathbf{detp}(M)$ is defined to be the tuple $(\mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}', q^0, \lambda', \delta')$ such that

- $\mathrm{St}' = \{s_d \ : \ s \in \mathrm{St} \ \&\ d \in \mathrm{D}\}$, and $q^0 = s^0_d$ for any $d \in \mathrm{D}$;
- $\lambda'(s_d) = \lambda(s)$, for all $d \in \mathrm{D}$; and $\delta'(s_d, d') = s'_{d'}$ if and only if $\delta(s, d') = s'$, for all $d \in \mathrm{D}$;

where s_d is an abbreviation for the pair (s, d) and D is the set of directions/decisions.

▶ **Lemma 5** (Deterministic past). *Let M be a CGS and* **detp** *be a deterministic-past map. Then* $\mathbf{detp}(M)$ *is a CGS with deterministic past of size $\mathcal{O}(|\mathrm{D}| \times |M|)$.*

The second map, namely $\mathbf{moore} : \mathcal{M} \to \mathcal{M}$, for consistency called a *Moore* map, is defined as follows. Given $M \in \mathcal{M}$, with $M = (\mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, s^0, \lambda, \delta)$, the structure $\mathbf{moore}(M)$, called a *Moore* CGS, is defined to be the tuple $(\mathrm{AP}', \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, s^0, \lambda', \delta)$ such that

- $\mathrm{AP}' = \mathrm{AP} \uplus \mathrm{APi} \uplus \mathrm{APo}$, where
  $\mathrm{APi} = \{\check{a}^x_i : i \in \mathrm{Ag} \ \&\ x \in \mathrm{Ac}_i\}$ and $\mathrm{APo} = \{\hat{a}^x_i : i \in \mathrm{Ag} \ \&\ x \in \mathrm{Ac}_i\}$;
- if $\delta(s, d) = s'$, with $d = (x_1, \ldots, y_n)$, then there are propositions $\check{a}^x_1, \ldots, \check{a}^y_n \in \mathrm{APi}$ and $\hat{a}^x_1, \ldots, \hat{a}^y_n \in \mathrm{APo}$, such that $\{\check{a}^x_1, \ldots, \check{a}^y_n\} \subseteq \lambda'(s') \cap \mathrm{APi}$ and $\{\hat{a}^x_1, \ldots, \hat{a}^y_n\} \subseteq \lambda'(s) \cap \mathrm{APo}$.

▶ **Lemma 6** (Moore CGS). *Let M be a CGS and* **moore** *be a Moore map. Then* $\mathbf{moore}(M)$ *is a Moore CGS of size $\mathcal{O}(|M|)$.*

Composing these two maps one can obtain, from a CGS $M = (\mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, s^0, \lambda, \delta)$, a CGS $M'' = (\mathrm{AP} \uplus \mathrm{APi} \uplus \mathrm{APo}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}', q^0, \lambda'', \delta')$ of size polynomial in $|M|$ which preserves all mATL* properties that can be defined by formulae in $\mathcal{L}(\mathrm{AP})$, provided that both $\mathrm{AP} \cap \mathrm{APi} = \emptyset$ and $\mathrm{AP} \cap \mathrm{APo} = \emptyset$ – which is, by definition, ensured by both maps.

▶ **Lemma 7.** *Let* $M = (\mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, s^0, \lambda, \delta)$ *be a CGS and* $M' = \mathbf{moore}(\mathbf{detp}(M))$. *Then,* M' *is a Moore CGS with deterministic past of size* $\mathcal{O}(|\mathrm{D}| \times |M|)$ *which is such that* $M \models \phi$ *if and only if* $M'|_{\mathrm{AP}} \models \phi$, *for all formulae* $\phi \in \mathcal{L}(\mathrm{AP})$.

Informally, Lemma 7 relies on the fact that, because of the definition of δ' in **detp**, the branching structure of every s of M is preserved in each $s_d \in \{s_d \ : \ d \in \mathrm{D}\}$ of $M'|_{\mathrm{AP}}$. In particular, note that, for any given $M \in \mathcal{M}$, the map $\mathbf{detp}(M)$ simply makes up to $|\mathrm{D}|$ (bisimilar) copies of each state of M where all players' choices are preserved in each s_d. Moreover, for any given $M \in \mathcal{M}$, the structures M and $\mathbf{moore}(M)|_{\mathrm{AP}}$ are isomorphic. Also, since both **detp** and **moore** are total maps on CGSs, it can easily be shown from the above lemmas the following result with respect to the properties that can be expressed in mATL*.

▶ **Corollary 8** (Model completeness). *For every* $\phi \in \mathcal{L}(\mathrm{AP})$, *it is the case that* ϕ *is satisfiable if and only if there is a Moore CGS with deterministic past model* M *such that* $M \models \phi$.

5 Nash Equilibria in Memoryful Logical Form

We now show that, up to bisimilarity, the concepts introduced in Section 3 can be given a logical characterisation in mATL*. Without much further introduction, we first present an mATL* formula that can be used to reason about Nash equilibria, and then describe how each part of the specification relates to the conditions given in Section 3. Some notation first.

Given a set of players $W \subseteq \mathrm{Ag}$, a set of actions $\mathrm{Ac} = \bigcup_{i \in \mathrm{Ag}} \mathrm{Ac}_i$ for such players, and a set of atomic propositions AP, we write L for $\mathrm{Ag} \setminus W$, and $\mathrm{APi} = \bigcup_{i \in \mathrm{Ag}} \mathrm{APi}_i$ and $\mathrm{APo} = \bigcup_{i \in \mathrm{Ag}} \mathrm{APo}_i$ for two sets of atomic propositions constructed based on Ac as follows:

$$x \in \mathrm{Ac}_i \iff \check{a}_i^x \in \mathrm{APi}_i \iff \hat{a}_i^x \in \mathrm{APo}_i,$$

such that $\mathrm{AP} \uplus \mathrm{APi} \uplus \mathrm{APo}$ is a set we denote by AP'. Then, the formula $\mathsf{NormForm}$, defined below, logically describes the labelling pattern found in the class of CGSs that satisfy the conditions to be a Moore CGS with deterministic past, just as we studied them in Section 4:

$$\mathsf{NormForm} = \mathbf{A}\,\mathbf{G} \bigwedge_{i \in \mathrm{Ag}} \Big(\big(\bigvee_{\check{a}_i^x \in \mathrm{APi}} \check{a}_i^x \big) \wedge \bigwedge_{\check{a}_i^x \in \mathrm{APi}_i} \big(\check{a}_i^x \implies \big(\bigwedge_{\check{a}_i^y \in \mathrm{APi}_i \setminus \{\check{a}_i^x\}} \neg \check{a}_i^y \big) \big) \wedge \bigwedge_{\hat{a}_i^x \in \mathrm{APo}_i} \big(\hat{a}_i^x \iff \mathbf{E}\,\mathbf{F}(\mathit{present} \wedge \mathbf{X}\,\check{a}_i^x) \big) \Big).$$

Then, the mATL* formula ϕ_{NE} below is satisfiable iff there is a Moore with deterministic past CGS $M = (\mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, s^0, \lambda, \delta)$ such that $G = (M, (\gamma_i)_{i \in \mathrm{Ag}})$ has a Nash equilibrium:

$$\phi_{\mathrm{NE}} = \Big(\mathbf{E} \bigvee_{W \subseteq \mathrm{Ag}} \big(\mathsf{NEGoal}(W) \wedge \mathsf{AllPunishable}(L) \big) \Big) \wedge \mathsf{NormForm}$$

such that

$$\mathsf{NEGoal}(W) = \bigwedge_{i \in W} \gamma_i \wedge \bigwedge_{j \in L} \neg \gamma_j$$

and

$$\mathsf{AllPunishable}(L) = \mathbf{G} \bigwedge_{d \in \mathrm{Ac}_1 \times \ldots \times \mathrm{Ac}_n} \Big(\mathsf{InRun}(d) \implies \bigwedge_{j \in L} \mathbf{A}\,\mathsf{Punishable}(j, d) \Big)$$

where

$$\mathsf{InRun}(d) = \bigwedge_{i \in \mathrm{Ag}} \mathbf{X}\,\check{a}_i^{d_i}$$

and

$$\mathsf{Punishable}(j,d) = \mathbf{F}\left(\mathit{present} \wedge \mathbf{X}\left(\mathsf{AttDev}(j,d) \Longrightarrow \mathsf{Punish(j)}\right)\right)$$

with

$$\mathsf{AttDev}(j,d) = \big(\bigwedge_{i\in \mathrm{Ag}_{-j}} \check{a}_i^{d_i}\big) \wedge \neg \check{a}_j^{d_j} \qquad \text{and} \qquad \mathsf{Punish}(j) = \langle\!\langle \mathrm{Ag}_{-j} \rangle\!\rangle \neg\gamma_j$$

Formula ϕ_{NE} simply describes the conditions given by Lemmas 3 and 4 to characterise the existence of Nash equilibria within the canonical class of game models given by the family of Moore CGSs with deterministic past. Let us now describe each formula in more detail.

The first line of NormForm indicates that every state must be labelled by one, and only one, profile of players' choices in APi (deterministic past); the second line, indicates that every player's choice available at some state is recorded in such a state (Moore). Note that because in mATL* path quantifiers reset the present, the CTL formula $\mathbf{E}\,\mathbf{X}\,\check{a}_i^x$ must be written as $\mathbf{E}\,\mathbf{F}(\mathit{present} \wedge \mathbf{X}\,\check{a}_i^x)$ – a standard mechanism to interpret formulae in the present.

The rest of the specification deals with Lemmas 3 and 4. It first indicates that for some set of players W (the "winners") there is a run that satisfies their goals, while the players not in W (the "losers") do not. That the goals are satisfied or not is checked by NEGoal. The formula also indicates that all "losers" must be punishable, which is specified by AllPunishable. This captures Lemma 4 provided that all individual deviations are uniquely attributable.

To check that all players in L are punishable in the same run and at every time point, we first check the profile of players' choices currently used ("d"). Because NormForm guarantees deterministic past, formula InRun will be necessarily satisfied by exactly one d, which is used to check, for every player in L, if it can be punished should it deviate. Since to consider a potential deviation one has to use a universal path quantifier, then the semantics of the logic dictates that the present is reset. This is the reason why the *present* formula must be used in Punishable. Finally, we need to check two conditions. On the one hand, that an individual deviation is uniquely attributable to j: formula AttDev does precisely that. On the other hand, that if this was the case, then player j can be effectively punished: formula Punish does exactly that. Note that memory is used in a different way for the first time; formula $\langle\!\langle \mathrm{Ag}_{-j} \rangle\!\rangle$ resets the present again, but this time we do not use the *present* proposition. Instead, we consider (the negation of) player j's goal from the beginning – *i.e.*, it is evaluated from the first time point, not from the present. Then, with respect to Lemma 3, formula $\langle\!\langle \mathrm{Ag}_{-j} \rangle\!\rangle$ represents $\vec{f}^{\,\pi}_{-j}$, while π is being naturally captured by the semantics of the logic.

Formally, appealing to Lemmas 3 and 4, the following result can be shown:

▶ **Theorem 9.** *Let $(\gamma_i)_{i\in\mathrm{Ag}}$, with each $\gamma_i \in \mathcal{L}(\mathrm{AP})$, be the goals of a set of players* Ag *with action set* $\mathrm{Ac} = \biguplus_{i\in\mathrm{Ag}} \mathrm{Ac}_i$. *Then, the formula ϕ_{NE} constructed from $(\mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, (\gamma_i)_{i\in\mathrm{Ag}})$ is satisfiable iff there is a Moore CGS with deterministic past $M = (\mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, s^0, \lambda, \delta)$ such that the game $G = (M, (\gamma_i)_{i\in\mathrm{Ag}})$ has a Nash equilibrium.*

Because ATL* is as expressive as mATL* – there is a non-elementary translation from mATL* to ATL* – the following result easily follows from Theorem 9.

▶ **Corollary 10.** *NE satisfiability in multi-player non-zero-sum perfect-information concurrent games can be characterised in ATL*.*

Some complexity results follow from Theorem 9. Since satisfiability and model checking for mATL* are 2EXPTIME, it follows that, given a tuple $(\mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, (\gamma_i)_{i\in\mathrm{Ag}})$, deciding whether there is a CGS $M = (\mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, s^0, \lambda, \delta)$ such that the game $G = (M, (\gamma_i)_{i\in\mathrm{Ag}})$ has a Nash equilibrium – the problem we call *NE satisfiability* – is a 2EXPTIME problem.

The other important decision problem, namely checking the existence of a Nash equilibrium over a given CGS M – the problem we call *NE model checking* – is already known to be 2EXPTIME and solved, for instance, using the Boolean goal fragment of SL. Then, we focus on NE satisfiability instead. Formally, the following complexity result can be shown.

▶ **Corollary 11.** *NE satisfiability is 2EXPTIME-complete.*

Regarding NE model checking, we would like to note that because ϕ_{NE} is an ATL* formula, it cannot distinguish between bisimilar models. Then, in order to do NE model checking with ϕ_{NE}, one has to necessarily ensure that the game being analysed already has deterministic past – for instance, such as the games studied in [5] – so that the map **detp**, which introduces bisimilar states to the model to be checked, need not be used.

In order to improve, or simplify, the results obtained in this section, in the next section we will study special cases where the problems under consideration may be less complex. In particular, we will consider two interesting, and frequently found, scenarios in the games literature: sequential games (sometimes also called turn-based games) and two-player games.

6 Special Cases

Sequential Games

The fact that in a sequential (turn-based) game only one player makes a non-trivial move greatly simplifies the way to reason about Nash equilibria using mATL*. In fact, we can express the existence of a Nash equilibrium in the logic m$^-$ATL* (mATL* without *present*), which is *strictly weaker* than mATL*. The formula to express Nash equilibria in m$^-$ATL* is:

$$\phi_{\mathsf{NE}}^{\mathsf{SG}} = \mathbf{E} \bigwedge_{j \in \text{Ag}} \left(\neg\gamma_j \Longrightarrow \mathbf{G} \langle\!\langle \text{Ag}_{-j} \rangle\!\rangle \neg\gamma_j \right)$$

from which, again using Lemmas 3 and 4, the next expressivity result can be shown.

▶ **Theorem 12.** *Let $(\gamma_i)_{i\in\text{Ag}}$, with each $\gamma_i \in \mathcal{L}(\text{AP})$, be the goals of a set of players* Ag*. Then, the m$^-$ATL* formula $\phi_{\mathsf{NE}}^{\mathsf{SG}}$ constructed from $(\text{AP}, \text{Ag}, (\gamma_i)_{i\in\text{Ag}})$ is satisfiable in a sequential game if and only if there is some CGS $M = (\text{AP}, \text{Ag}, \text{Ac}, \text{St}, s^0, \lambda, \delta)$ such that the multi-player non-zero-sum sequential game $G = (M, (\gamma_i)_{i\in\text{Ag}})$ has a Nash equilibrium.*

In this case, one can see that the formula $\mathbf{G} \langle\!\langle \text{Ag}_{-j} \rangle\!\rangle \neg\gamma_j$ is almost an explicit logical description of $\pi\, \rho((\vec{f}^{\pi}_{-j}, g_j), \pi) \notin \Gamma_j$, from Lemma 3, as the expression must hold for every π. There are two reasons why the expression for sequential games is so much simpler than the general case: firstly, only one player can deviate at each state – thus any deviation is uniquely attributable – and, secondly, winning (punishment) strategies can be defined directly on the Nash equilibrium run being followed, rather than on paths taken by deviations.

Two-player Games

The complexity of both checking the existence (NE satisfiability) and verifying the satisfaction (NE model checking) of Nash equilibria cannot be improved in the two-player case. However, an explicit and conceptually simple algorithm for NE satisfiability can be defined in the two-player case, which can be solved using CTL* synthesis, as usual, by associating the action set $\text{Ac} = \text{Ac}_1 \uplus \text{Ac}_2$ with the set of atomic propositions AP. We write $\text{SAT}(\psi)$ for LTL satisfiability and $\text{SYN}(\psi, \mathit{In}, \mathit{Out})$ for CTL* synthesis, where player 1 plays first trying to realise ψ, controls In, and with In and Out the sets of input and output variables; cf. [5].

TWO-NE-SATISFIABILITY($\mathrm{AP}_1, \mathrm{AP}_2, \gamma_1, \gamma_2$)
1. if SAT($\gamma_1 \wedge \gamma_2$) return "yes"
2. if SYN(γ_1, AP_1, AP_2) or SYN(γ_2, AP_2, AP_1) return "yes"
3. if SYN($\neg\gamma_2$, AP_1, AP_2) and SYN($\neg\gamma_1$, AP_2, AP_1) return "yes"
4. if SYN($\mathbf{E}\,\gamma_1 \wedge \mathbf{A}\,\neg\gamma_2$, AP_1, AP_2) or SYN($\mathbf{E}\,\gamma_2 \wedge \mathbf{A}\,\neg\gamma_1$, AP_2, AP_1) return "yes"
5. return "no"

The above algorithm, inspired by the results obtained in [6], suggests a simple and intuitive mATL* characterisation of the NE satisfiability problem in two-player games:

$$\begin{aligned}\phi_{\mathsf{NE}}^{\mathsf{2G}} \;=\; & \Big(\mathbf{E}(\gamma_1 \wedge \gamma_2)\Big) \vee \Big(\langle\!\langle 1\rangle\!\rangle\, \gamma_1 \vee \langle\!\langle 2\rangle\!\rangle\, \gamma_2\Big) \vee \Big(\langle\!\langle 1\rangle\!\rangle\, \neg\gamma_2 \wedge \langle\!\langle 2\rangle\!\rangle\, \neg\gamma_1\Big) \vee \Big(\mathsf{NormForm} \wedge \big(\\ & \mathbf{E}\left(\gamma_1 \wedge \neg\gamma_2 \wedge \mathbf{G} \textstyle\bigwedge_{p\in \mathrm{AP}_1} (\mathbf{X}\, \check{a}_1^p \Longrightarrow [\![2]\!]\, \mathbf{F}(\mathit{present} \wedge \mathbf{X}(\check{a}_1^p \Longrightarrow \langle\!\langle 1\rangle\!\rangle\, \neg\gamma_2)))\right) \vee \\ & \mathbf{E}\left(\gamma_2 \wedge \neg\gamma_1 \wedge \mathbf{G} \textstyle\bigwedge_{p\in \mathrm{AP}_2} (\mathbf{X}\, \check{a}_2^p \Longrightarrow [\![1]\!]\, \mathbf{F}(\mathit{present} \wedge \mathbf{X}(\check{a}_2^p \Longrightarrow \langle\!\langle 2\rangle\!\rangle\, \neg\gamma_1)))\right) \big)\Big).\end{aligned}$$

That NE satisfiability for two-player turn-based games can be characterised in ATL* was known [3]. Our result is instead in the concurrent setting. The specification requires *present* and we do not see how to avoid it, which suggests that *concurrency makes things harder*.

▶ **Theorem 13.** *NE satisfiability for two-player non-zero-sum concurrent games can be solved using CTL* synthesis and characterised in ATL* using a translation from formula $\phi_{\mathsf{NE}}^{\mathsf{2G}}$.*

7 Further Implications

Strong Nash Equilibrium

The solution concept we have focused so far, namely Nash equilibrium, assumes that deviations are only possible by single players. In the concurrent and multi-agent world, however, it is natural to assume that players may deviate in groups in order to achieve a common goal. Such kind of behaviour is captured by the notion of *strong Nash equilibrium* where, informally, a strategy profile is a strong Nash equilibrium whenever no coalition/group of players who do not get their goals achieved can deviate in a beneficial way for all deviating players. Formally, a *strong Nash equilibrium strategy profile* is a strategy profile $\vec{f} = (f_1, \ldots, f_n)$ such that for all groups C of players and all strategies g_C in F_C, for all $j \in C$ we have $\rho(\vec{f}) \succsim_j \rho(\vec{f}_{-C}, g_C)$. We say that a run ρ can be *sustained by a strong Nash equilibrium strategy profile* if there is some strong Nash equilibrium strategy profile $\vec{f}$ such that $\rho = \rho(\vec{f})$.

Our study of punishability in Section 3 can be extended to deviations by groups of players, and therefore to strong Nash equilibrium. Similarly, the logical specification given in Section 5 can also be easily modified to capture the new setting, in particular, by letting $\bigwedge_{C\subseteq L} \mathbf{A}\,\mathsf{Punishable}(C, d)$ in $\mathsf{AllPunishable}$, and re-defining AttDev and Punish as follows:

$$\mathsf{AttDev}(C,d) = \Big(\bigwedge_{i\in \mathrm{Ag}_{-C}} \check{a}_i^{d_i}\Big) \wedge \Big(\bigwedge_{j\in C} \neg\check{a}_j^{d_j}\Big) \qquad \text{and} \qquad \mathsf{Punish}(C) = \langle\!\langle \mathrm{Ag}_{-C}\rangle\!\rangle \bigvee_{j\in C} \neg\gamma_j\ .$$

We write ϕ_{SNE} for the characterisation in mATL* of satisfiability for strong Nash equilibrium.

▶ **Theorem 14.** *Let $(\gamma_i)_{i\in \mathrm{Ag}}$, with each $\gamma_i \in \mathcal{L}(\mathrm{AP})$, be the goals of a set of players* Ag *with action set* $\mathrm{Ac} = \biguplus_{i\in \mathrm{Ag}} \mathrm{Ac}_i$. *Then, the formula ϕ_{SNE} constructed from $(\mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, (\gamma_i)_{i\in \mathrm{Ag}})$ is satisfiable iff there is a Moore CGS with deterministic past $M = (\mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, s^0, \lambda, \delta)$ such that the game $G = (M, (\gamma_i)_{i\in \mathrm{Ag}})$ has a strong Nash equilibrium.*

We would like to note that even though ϕ_{SNE} is more complex than ϕ_{NE}, the latter formula can still characterise the satisfiability problem for strong Nash equilibrium in the two simpler settings we studied in the previous section. Firstly, note that in the sequential case, trivially, the same specification works for both Nash equilibrium and strong Nash equilibrium since only one player can deviate at a time. Likewise, with respect to two-player games the same specification also works for both solution concepts since the only deviation for a group of players greater than one, is to a run that satisfies all players' goals, which, by definition, is a Nash equilibrium. Formally, we have the following expressivity result.

▶ **Theorem 15.** *Formula ϕ_{NE} characterises the satisfiability problems for both Nash and strong Nash equilibria in two-player concurrent games and multi-player sequential games.*

Synthesis

Finally, we study the nature – memoryless, finite-memory, etc. – of the strategies in the equilibrium strategy profiles discussed so far, should one intend to synthesise them. This has been an open question for some time: because the logics known to be able to express equilibria in multi-player games have an undecidable synthesis problem (and a non-elementary model checking problem), strategy synthesis is not fully understood in the most general case. On the contrary, in our setting the problem can be reduced to ATL* synthesis, which is known to be solvable using (one-bit) finite-memory strategies [15]. Formally, we have:

▶ **Corollary 16.** *Every (strong) Nash equilibrium can be realised using a (strong) Nash equilibrium profile $\vec{f} = (f_1, \ldots, f_n)$, where each f_i is a finite-memory strategy.*

8 Analysis

We have presented various expressivity and complexity results for the specification, computation, and verification of (strong) Nash equilibria, as well as particular results for two special scenarios, namely two-player games and sequential games. Our results build on a novel approach: a semantic characterisation of equilibria based on winning strategies and memoryful reasoning, which is amenable to a further temporal logic characterisation.

Our results suggest that a logic for strategic reasoning where memoryful reasoning and attributability can be explicitly captured may ease the logical description of the existence of Nash equilibria. Otherwise, having a model of games where deviations can be more easily recognised should help when reasoning about Nash equilibria within a logical framework.

Acknowledgements. We are extremely grateful to the anonymous reviewers for their detailed comments. We thank the financial support of the ERC grant 291528 ("RACE") at Oxford.

References

1 Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.
2 Krishnendu Chatterjee and Thomas A. Henzinger. A survey of stochastic ω-regular games. *J. Comput. Syst. Sci.*, 78(2):394–413, 2012.
3 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy logic. *Inf. Comput.*, 208(6):677–693, 2010.
4 Bernd Finkbeiner and Sven Schewe. Coordination logic. In *CSL*, volume 6247 of *LNCS*, pages 305–319. Springer, 2010.

5 Julian Gutierrez, Paul Harrenstein, and Michael Wooldridge. Iterated Boolean games. *Inf. Comput.*, 242:53–79, 2015.

6 Julian Gutierrez and Michael Wooldridge. Equilibria of concurrent games on event structures. In *CSL-LICS*. ACM, 2014.

7 Orna Kupferman and Moshe Y. Vardi. Memoryful branching-time logic. In *LICS*, pages 265–274. IEEE Computer Society, 2006.

8 François Laroussinie. Temporal logics for games. *Bulletin of the EATCS*, 100:79–98, 2010.

9 François Laroussinie and Nicolas Markey. Quantified CTL: expressiveness and complexity. *LMCS*, 10(4), 2014.

10 François Laroussinie and Nicolas Markey. Augmenting ATL with strategy contexts. *Inf. Comput.*, 2015. To appear.

11 Fabio Mogavero, Aniello Murano, Giuseppe Perelli, and Moshe Y. Vardi. Reasoning about strategies: On the model-checking problem. *ACM Trans. Comput. Log.*, 15(4):34, 2014.

12 Fabio Mogavero, Aniello Murano, and Luigi Sauro. On the boundary of behavioral strategies. In *LICS*, pages 263–272. IEEE Computer Society, 2013.

13 Fabio Mogavero, Aniello Murano, and Moshe Y. Vardi. Relentful strategic reasoning in alternating-time temporal logic. *J. Log. Comput.*, 2015. To appear.

14 Martin Osborne and Ariel Rubinstein. *A Course in Game Theory*. The MIT Press, 1994.

15 Sven Schewe. ATL* satisfiability is 2EXPTIME-complete. In *ICALP*, volume 5126 of *LNCS*, pages 373–385. Springer, 2008.

16 Wiebe van der Hoek and Michael Wooldridge. Logics for multiagent systems. *AI Magazine*, 33(3):92–105, 2012.

17 Igor Walukiewicz. A landscape with games in the background. In *LICS*, pages 356–366. IEEE Computer Society, 2004.

Meeting Deadlines Together

Laura Bocchi[1], Julien Lange[2], and Nobuko Yoshida[2]

1 **University of Kent, UK**
2 **Imperial College London, UK**

Abstract

This paper studies safety, progress, and non-zeno properties of Communicating Timed Automata (CTAs), which are timed automata (TA) extended with unbounded communication channels, and presents a procedure to *build* timed global specifications from systems of CTAs. We define safety and progress properties for CTAs by extending properties studied in communicating finite-state machines to the timed setting. We then study non-zenoness for CTAs; our aim is to prevent scenarios in which the participants have to execute an infinite number of actions in a finite amount of time. We propose sound and decidable conditions for these properties, and demonstrate the practicality of our approach with an implementation and experimental evaluations of our theory.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Program

Keywords and phrases timed automata, multiparty session types, global specification

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.283

1 Introduction

Meeting deadlines is part of our everyday life; this is also the case for distributed software systems that have real-time constraints, such as e-business and financial systems, where exchanges of agreements and data transmissions need to be completed within specified time-frames. Guaranteeing that a single entity will finish its assigned task within an upcoming deadline is a crucial requirement that is generally difficult to attain. It is even harder to ensure that several, distributed, and interdependent entities will work *together* in a timely fashion to meet each other's deadlines. To model such real-time distributed behaviours, communicating timed automata (CTAs) [17] have been introduced as an extension of communicating finite-state machines (CFSMs) [9] with time constraints. A system of CTAs consists of several automata that exchange messages through unbounded FIFO channels and must comply with time constraints on emission/reception of messages. These two features (unbounded channels and time) make CTAs difficult to verify, e.g., reachability is undecidable in general [12].

This paper tackles the following two shortcomings of the current state-of-the-art of CTAs. First, to the best of our knowledge, safety and progress properties, such as absence of deadlocks and unspecified reception (type) errors, which are standard in the literature on CFSMs [10], and essential for distributed systems, have not been studied in the context of CTAs. Moreover, customary properties for TAs such as time-divergence [2] and non-zenoness [21, 7] (preventing that some participant's only possible way forward is by firing actions at increasingly short intervals of time) have not been investigated for CTAs.

Second, while global specifications such as message sequent charts (MSC) and choreographies [8, 16] are useful to model protocols from a global viewpoint, there has not been any work to *build* global specifications from CTAs. The top-down approach [6] alone, which requires a preexisting global specification, is not satisfactory in agile development life-cycles [23], in refinement and reverse-engineering of existing systems, or to compose real-time distributed components, possibly dynamically (see [18, 19, 14]).

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 283–296

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

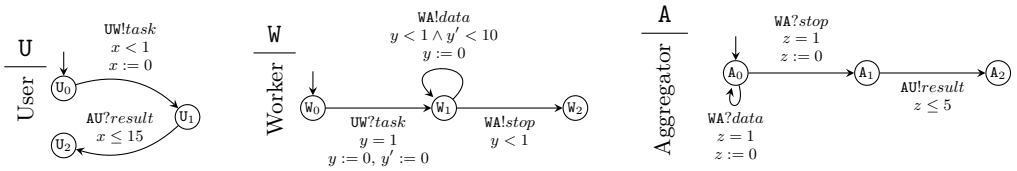


Figure 1 Scheduled Task Protocol (System S_{ST}).

This work introduces classical properties of CFSMs and TAs to the world of CTAs, and investigates the interplay between asynchronous communications through unbounded channels and time constraints. We define the classes of CTAs that enjoy four properties – *safety*, *progress*, *non-zenoness*, and *eventual reception* – and give a sound decision procedure for checking whether a system of CTAs belongs to these classes. This procedure does not rely on any other information than the CTAs themselves. Interestingly, a property of CFSMs called multiparty compatibility (MC) [14], which characterises a sound and complete correspondence with multiparty session types in the untimed setting [16], soundly characterises safe CTAs and offers a basis for decidable decision procedures for progress and non-zenoness in the timed setting. We give: (i) a sound characterisation for progress by checking the satisfiability of first order logic formulae (thus verifiable by generic SMT solvers), and (ii) a sound characterisation of non-zenoness by using a synchronous execution of CTAs. Eventual reception follows from (i) and (ii). In addition, we present an algorithm to build a timed global type [6] from CTAs, whose traces are equivalent to the original system. Thus, if a system validates some of the properties discussed above, then the CTAs obtained by projecting its timed global type onto its participants will preserve these properties.

The system S_{ST} in Fig. 1 (Scheduled Task Protocol) will be used to illustrate our approach throughout the paper. S_{ST} consists of three participants (or machines): a user `U`, a worker `W`, and an aggregator `A`, who exchange messages through *unbounded* FIFO buffers. Each machine is equipped with one or more clocks, initially set to 0 and possibly reset during the protocol. Time elapses at the same pace for all clocks, which is a standard assumption [17]. The protocol is as follows: `U` sends a task to `W`, `W` progressively sends intermediary data to `A`, and finally `A` sends the aggregated result to `U`. The time constraints are:

- `U` must send a task to `W` within one time unit, reset its clock x, and expects to receive the result within 15 time units.
- `W` must consume `U`'s *task* message at time 1, reset its clocks y and y', and repeatedly send *data* to `A`, waiting less than 1 time unit between each emission (modelled by the constraint and reset on y). The overall iteration cannot last more than 10 time units (modelled by the constraint on y', which is not reset in the loop). When `W` has finished, it must send a notification *stop* to `A`.
- `A` must read intermediary data every 1 time unit, reset each time its clock z, and send the overall result to `U` within 5 time units after receiving *stop*.

This example, albeit small, models a complex interaction where each machine has its own, interdependent, deadlines; e.g., `U` relies on the other machines' deadlines to receive the final result within 15 time units. Note that the channel between `W` and `A` is *unbounded*: `W` can send to `A` an arbitrary number of messages before `A` receives them, cf. $\texttt{WA!}data(y < 1 \wedge y' < 10, y := 0)$.

Contribution and synopsis. In the rest of the paper, we give several conditions that guarantee that no participant misses its deadlines, that every message sent is eventually

received *on time*, and that no participant is forced to perform actions infinitely fast, i.e., forced into a zeno behaviour. In § 2 we recall basic definitions on CTAs. In § 3 we extend the standard safety properties of CFSMs to the timed setting, and show that multiparty compatibility (MC) is a sound condition for safety (Theorem 6). MC CTAs still allow undesirable scenarios when, e.g., (1) the system gets stuck because of unmeetable deadlines, (2) the system's only possibility to meet its deadlines is through zeno behaviours, or (3) sent messages are never received. We give sound and decidable conditions to rule out (1) in § 4 (Theorem 13) and (2-3) in § 5 (Theorem 17 and Theorem 19). In § 6, we discuss the applications of our theory and its implementation. The work in [6] studies a correspondence between timed local types (projected from timed global types) and CTAs, focusing on type-checking timed π-calculus processes. The present work studies CTAs directly, i.e., without relying on a priori global knowledge of the system, and gives more general conditions for safety, progress, and non-zenoness. None of the previous works [18, 19, 14] on building global specifications from local ones caters for time constraints. Unlike existing work on the properties of CTAs (e.g., reachability) our results do not set limitations to channel size or to network topologies [17, 12]. We discuss related work further in § 7. The proofs, additional material, and the implementation are available online [3].

2 Communicating Timed Automata

We introduce communicating timed automata (CTA) following definitions from [14, 17]. Fix a finite set $\mathcal{P}$ of *participants* (ranged over by $\mathtt{p}$, $\mathtt{q}$, $\mathtt{r}$, $\mathtt{s}$, etc.). Let $\mathbb{A}$ be a finite alphabet of messages ranged over by a, b, etc. The set of finite words on $\mathbb{A}$ is denoted by $\mathbb{A}^*$, ww' is the concatenation of w and w', and ε is the empty word (overloaded on any alphabet). The set of *channels* is $C \stackrel{\text{def}}{=} \{\mathtt{pq} \mid \mathtt{p},\mathtt{q} \in \mathcal{P} \text{ and } \mathtt{p} \neq \mathtt{q}\}$. Given a (finite) set of *clocks* $\mathcal{X}$ (ranged over by x, y, etc.), the set of *actions* (ranged over by ℓ) is $Act_{\mathcal{X}} \stackrel{\text{def}}{=} C \times \{!, ?\} \times \mathbb{A} \times \Phi(\mathcal{X}) \times 2^{\mathcal{X}}$, and the set of *guards* (ranged over by g) $\Phi(\mathcal{X})$ is

$$g ::= \mathsf{true} \mid x \leq c \mid c \leq x \mid \neg g \mid g_1 \wedge g_2$$

where c ranges over constants in $\mathbb{Q}_{\geq 0}$, and from which we derive the usual abbreviations. We write $\mathsf{fc}(g)$ for the set of clocks in g and $\mathtt{sr}!a(g,\lambda)$ or $\mathtt{sr}?a(g,\lambda)$ for an element of $Act_{\mathcal{X}}$. Action $\mathtt{sr}!a(g,\lambda)$ says that $\mathtt{s}$ sends a message a to $\mathtt{r}$, provided that guard g is satisfied, and resets the clocks in $\lambda \subseteq \mathcal{X}$; the dual receiving action is $\mathtt{sr}?a(g,\lambda)$. Given $\ell = \mathtt{sr}!a(g,\lambda)$ or $\ell = \mathtt{sr}?a(g,\lambda)$, we define: $\mathsf{msg}(\ell) = a$, $\mathsf{guard}(\ell) = g$, and $\mathsf{reset}(\ell) = \lambda$. We define the subject of an action: $\mathsf{subj}(\mathtt{pr}!a(g,\lambda)) = \mathsf{subj}(\mathtt{sp}?a(g,\lambda)) \stackrel{\text{def}}{=} \mathtt{p}$.

A *communicating timed automaton*, or *machine*, is a finite transition system given by a tuple $M = (Q, q_0, \mathcal{X}, \delta)$ where Q is a finite set of *states*, $q_0 \in Q$ is the initial state, $\mathcal{X}$ is a set of clocks, and $\delta \subseteq Q \times Act_{\mathcal{X}} \times Q$ is a set of *transitions*. We write $q \xrightarrow{\ell} q'$ when $(q, \ell, q') \in \delta$.

A machine $M = (Q, q_0, \mathcal{X}, \delta)$ is *deterministic* if for all states $q \in Q$ and all actions $\ell, \ell' \in Act_{\mathcal{X}}$, if $(q,\ell,q'), (q,\ell',q'') \in \delta$ and $\mathsf{msg}(\ell) = \mathsf{msg}(\ell')$, then $q' = q''$ and $\ell = \ell'$. A state $q \in Q$ is: *final* if it has no outgoing transitions; *sending* (resp. *receiving*) if it is not final and each of its outgoing transitions is of the form $\mathtt{sr}!a(g,\lambda)$ (resp. $\mathtt{sr}?a(g,\lambda)$); and *mixed* if it is neither final, sending, nor receiving. We say that q is *directed* if it contains only sending/receiving actions to/from the same participant. Hereafter, we only consider deterministic machines, whose states are directed and not mixed. These assumptions, adapted from [14], ensure that a machine corresponds to a syntactic local session type [16]. We discuss how to lift some of these restrictions in § 7.

A *timed communicating system* consists of a finite set of machines and a set of queues (one for each channel) used for asynchronous message passing. Given a valuation $\nu : \mathcal{X} \to \mathbb{R}_{\geq 0}$ of

the clocks in $\mathcal{X}$, $\nu \models g$ denotes that the guard g is satisfied by ν and $\lambda(\nu)$ denotes a valuation where all clocks in λ are set to 0 (reset) and clocks not in λ keep their values in ν.

▶ **Definition 1** (Timed communicating system)**.** A timed communicating system (or *system*), is a tuple $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ where each $M_{\mathtt{p}} = (Q_{\mathtt{p}}, q_{0\mathtt{p}}, \mathcal{X}_{\mathtt{p}}, \delta_{\mathtt{p}})$ is a CTA and for all $\mathtt{p}{\neq}\mathtt{q} \in \mathcal{P}:\ \mathcal{X}_{\mathtt{p}} \cap \mathcal{X}_{\mathtt{q}} = \emptyset$. A *configuration* of S is a triple $s = (\vec{q}; \vec{w}; \nu)$ where: $\vec{q} = (q_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ is the *control state* and $q_{\mathtt{p}} \in Q_{\mathtt{p}}$ is the *local state* of machine $M_{\mathtt{p}}$; $\vec{w} = (w_{\mathtt{pq}})_{\mathtt{pq}\in C}$ with $w_{\mathtt{pq}} \in \mathbb{A}^*$ is a vector of queues; $\nu : \bigcup_{\mathtt{p}\in\mathcal{P}} \mathcal{X}_{\mathtt{p}} \to \mathbb{R}_{\geq 0}$ is a clock valuation. The *initial configuration* of S is $s_0 = (\vec{q_0}; \vec{\varepsilon}; \nu_0)$ with $\vec{q_0} = (q_{0\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$, $\vec{\varepsilon}$ being the vector of empty queues, and $\nu_0(x) = 0$ for each clock $x \in \bigcup_{\mathtt{p}\in\mathcal{P}} \mathcal{X}_{\mathtt{p}}$. ◀

Hereafter, we fix a machine $M_{\mathtt{p}} = (Q_{\mathtt{p}}, q_{0\mathtt{p}}, \mathcal{X}_{\mathtt{p}}, \delta_{\mathtt{p}})$ for each participant $\mathtt{p} \in \mathcal{P}$ (assuming that $\forall \mathtt{p} \in \mathcal{P}:\ (q, \ell, q') \in \delta_{\mathtt{p}} \implies \mathsf{subj}(\ell) = \mathtt{p}$), and let $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ be the corresponding system. We write $\mathcal{X}$ for $\bigcup_{\mathtt{p}\in\mathcal{P}} \mathcal{X}_{\mathtt{p}}$ and $\nu + t$ for the valuation mapping each $x \in \mathcal{X}$ to $\nu(x) + t$. The definition below is from [17, Definition 1], omitting internal transitions.

▶ **Definition 2** (Reachable configuration)**.** Configuration $s' = (\vec{q'}; \vec{w'}; \nu')$ is *reachable* from configuration $s = (\vec{q}; \vec{w}; \nu)$ by *firing the transition* α, written $s \xrightarrow{\alpha} s'$ (or $s \to s'$ when the label is immaterial), if either:

1. $(q_{\mathtt{s}}, \mathtt{sr}!a(g, \lambda), q'_{\mathtt{s}}) \in \delta_{\mathtt{s}}$ and (a) $q'_{\mathtt{p}} = q_{\mathtt{p}}$ for all $\mathtt{p}{\neq}\mathtt{s}$; (b) $w'_{\mathtt{sr}} = w_{\mathtt{sr}}a$ and $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ for all $\mathtt{pq}{\neq}\mathtt{sr}$; (c) $\nu' = \lambda(\nu)$; (d) $\alpha = \mathtt{sr}!a(g, \lambda)$, and $\nu \models g$;
2. $(q_{\mathtt{r}}, \mathtt{sr}?a(g, \lambda), q'_{\mathtt{r}}) \in \delta_{\mathtt{r}}$ and (a) $q'_{\mathtt{p}} = q_{\mathtt{p}}$ for all $\mathtt{p}{\neq}\mathtt{r}$; (b) $w_{\mathtt{sr}} = aw'_{\mathtt{sr}}$ and $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ for all $\mathtt{pq}{\neq}\mathtt{sr}$; (c) $\nu' = \lambda(\nu)$; (d) $\alpha = \mathtt{sr}?a(g, \lambda)$ and $\nu \models g$; or
3. $\alpha = t \in \mathbb{R}_{\geq 0}$, $\nu' = \nu + t$, $w'_{\mathtt{pq}} = w_{\mathtt{pq}}$ for all $\mathtt{pq} \in C$, and $q'_{\mathtt{p}} = q_{\mathtt{p}}$ for all $\mathtt{p} \in \mathcal{P}$.

We let ρ range over sequences of labels $\alpha_1 \cdots \alpha_k$ and write $\to^*$ for the reflexive transitive closure of $\to$. The *reachability set* of S is $RS(S) \stackrel{\text{def}}{=} \{s \mid s_0 \to^* s\}$. ◀

Condition (1) allows a machine $\mathtt{s}$ to put a message a on queue $\mathtt{sr}$, if the time constraints in g are satisfied by ν; dually, (2) allows $\mathtt{r}$ to consume a message from the queue, if g is satisfied; and (3) models the elapsing of time (or a delay).

3 Safety in CTAs

This section defines safe CTAs and gives a sufficient condition for safety, called multiparty compatibility (MC) [14], in the timed setting. Here, we present a new approach based on *synchronous transition systems* (*STS*); the *STS* is also useful for defining progress and non-zeno properties in § 4.

Let n range over vectors of local states; and e range over events, which are elements of the set $C \times \mathbb{A} \times \Phi(\mathcal{X}) \times 2^{\mathcal{X}} \times \Phi(\mathcal{X}) \times 2^{\mathcal{X}}$, and write $(\mathtt{s} \to \mathtt{r} : a;\ g_{\mathtt{s}}, \lambda_{\mathtt{s}};\ g_{\mathtt{r}}, \lambda_{\mathtt{r}})$ for the event in which $\mathtt{s}$ sends message a to $\mathtt{r}$, with $\mathtt{s}$ (resp. $\mathtt{r}$) having guard $g_{\mathtt{s}}$ (resp. $g_{\mathtt{r}}$) and resets $\lambda_{\mathtt{s}}$ (resp. $\lambda_{\mathtt{r}}$). We introduce the synchronous transition system of S, following [19].

▶ **Definition 3** (Synchronous transition system)**.** The *synchronous transition system* of S, written $STS(S)$, is a tuple $(N, n_0, \hookrightarrow, E)$ such that:

- $\hookrightarrow$ is the relation defined as $n \stackrel{e}{\hookrightarrow} n'$ with $e = (\mathtt{s} \to \mathtt{r} : a;\ g_{\mathtt{s}}, \lambda_{\mathtt{s}};\ g_{\mathtt{r}}, \lambda_{\mathtt{r}})$ iff $n = \vec{q}, n' = \vec{q}', q_{\mathtt{s}} \xrightarrow{\mathtt{sr}!a(g_{\mathtt{s}}, \lambda_{\mathtt{s}})} q'_{\mathtt{s}}, q_{\mathtt{r}} \xrightarrow{\mathtt{sr}?a(g_{\mathtt{r}}, \lambda_{\mathtt{r}})} q'_{\mathtt{r}}$, and $\forall \mathtt{p} \in \mathcal{P} \setminus \{\mathtt{s}, \mathtt{r}\} :\ q_{\mathtt{p}} = q'_{\mathtt{p}}$ (write $\hookrightarrow$ when e is unimportant and $\hookrightarrow^*$ for the reflexive and transitive closure of $\hookrightarrow$);
- $n_0 = \vec{q_0}$ is the initial node; $N = \{n \mid n_0 \hookrightarrow^* n\}$ is the (finite) set of nodes; and $E = \{e \mid \exists n, n' \in N \text{ and } n \stackrel{e}{\hookrightarrow} n'\}$ is the set of events.

We write $n_1 \xrightarrow{e_1 \cdots e_k} n_{k+1}$, when, for some $n_2, \ldots, n_k \in N$, $n_1 \overset{e_1}{\hookrightarrow} n_2 \cdots n_k \overset{e_k}{\hookrightarrow} n_{k+1}$. Let φ range over (possibly empty) sequences of events $e_1 \cdots e_k$, and ε denote the empty sequence. ◀

The *STS* of the Scheduled Task Protocol (S_{ST}) is given in Fig. 2; essentially, it models all the synchronous executions of S_{ST}. In the following, we fix $STS(S) = (N, n_0, \hookrightarrow, E)$.

Given $e = (\mathtt{s} \to \mathtt{r} : a;\ g_{\mathtt{s}}, \lambda_{\mathtt{s}};\ g_{\mathtt{r}}, \lambda_{\mathtt{r}})$, we define $\mathsf{sid}(e) \stackrel{\text{def}}{=} \mathtt{s}$, $\mathsf{rid}(e) \stackrel{\text{def}}{=} \mathtt{r}$, and $\mathsf{id}(e) \stackrel{\text{def}}{=} \{\mathtt{s}, \mathtt{r}\}$. The projection of e on $\mathtt{p}$ (written $e\!\upharpoonright_{\mathtt{p}}$) is given by: $(\mathtt{s} \to \mathtt{r} : a;\ g_{\mathtt{s}}, \lambda_{\mathtt{s}};\ g_{\mathtt{r}}, \lambda_{\mathtt{r}})\!\upharpoonright_{\mathtt{s}} = \mathtt{sr}!a(g_{\mathtt{s}}, \lambda_{\mathtt{s}})$; $(\mathtt{s} \to \mathtt{r} : a;\ g_{\mathtt{s}}, \lambda_{\mathtt{s}};\ g_{\mathtt{r}}, \lambda_{\mathtt{r}})\!\upharpoonright_{\mathtt{r}} = \mathtt{sr}?a(g_{\mathtt{r}}, \lambda_{\mathtt{r}})$; and $(\mathtt{s} \to \mathtt{r} : a;\ g_{\mathtt{s}}, \lambda_{\mathtt{s}};\ g_{\mathtt{r}}, \lambda_{\mathtt{r}})\!\upharpoonright_{\mathtt{p}} = \varepsilon$, if $\mathtt{p} \notin \{\mathtt{s}, \mathtt{r}\}$. We extend $\varphi\!\upharpoonright_{\mathtt{p}}$ to sequences of events and, given $n \in N$, define $\mathsf{ids}(n) \stackrel{\text{def}}{=} \bigcup\{\mathsf{id}(e) \mid n \hookrightarrow^* \overset{e}{\hookrightarrow}\}$.

▶ **Definition 4** (Multiparty compatibility (MC)). System S is *multiparty compatible* if for all $\mathtt{p} \in \mathcal{P}$, for all $q \in Q_{\mathtt{p}}$, and for all $n = \vec{q} \in N$, if $q_{\mathtt{p}} = q$, then

1. if q is a sending state, then $\forall (q, \ell, q') \in \delta_{\mathtt{p}} : \exists \varphi, \exists e \in E : n \overset{\varphi \cdot e}{\hookrightarrow} \wedge\, e\!\upharpoonright_{\mathtt{p}} = \ell \wedge \varphi\!\upharpoonright_{\mathtt{p}} = \varepsilon$;
2. if q is a receiving state, then $\exists (q, \ell, q') \in \delta_{\mathtt{p}} : \exists \varphi, \exists e \in E : n \overset{\varphi \cdot e}{\hookrightarrow} \wedge\, e\!\upharpoonright_{\mathtt{p}} = \ell \wedge \varphi\!\upharpoonright_{\mathtt{p}} = \varepsilon$. ◀

Intuitively, condition (1) ensures that for every sending state, all messages that can be sent can also be received, while (2) guarantees that, for every receiving state, at least one transition will be eventually fireable, i.e., an *expected* message will eventually be received. System S_{ST}, in Fig. 1, is multiparty compatible.

Observe that $STS(S)$ and MC do not address time constraints. In fact, $STS(S)$ might include interactions forbidden by time constraints. These can be ruled out at a later stage when analysing time properties in § 4. We deliberately kept communication and time properties separated, so that we can provide simpler and modular definitions in § 7. Crucially, MC guarantees that any *asynchronous* execution can be mapped to a path in $STS(S)$, i.e., it can be simulated by $STS(S)$.

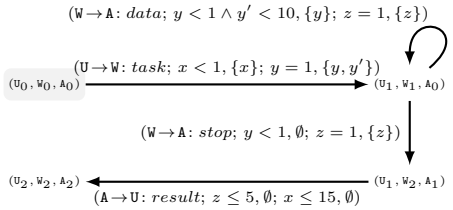


Figure 2 *STS* for Scheduled Task, cf. Fig. 1.

We recall two types of errors from the CFSM model, which are ruled out by MC also in the timed setting. Let $s = (\vec{q}; \vec{w}; \nu)$ be a configuration of a system S; s is a ***deadlock configuration*** [10, Def. 12] if $\vec{w} = \vec{\varepsilon}$, there is $\mathtt{r} \in \mathcal{P}$ such that $q_{\mathtt{r}}$ is a receiving state, and for every $\mathtt{p} \in \mathcal{P}$, $q_{\mathtt{p}}$ is a receiving or final state, i.e., all machines are blocked waiting for messages; and s is an ***orphan message configuration*** if all $q_{\mathtt{p}} \in \vec{q}$ are final but $\vec{w} \neq \vec{\varepsilon}$, i.e., there is at least a non-empty buffer and all the machines are in a final state.

▶ **Definition 5** (Safe system). S is *safe* if for all $s \in RS(S)$, s is not a deadlock, nor an orphan message configuration. ◀

▶ **Theorem 6** (Safety). *If S is multiparty compatible, then it is safe.*

The proof follows from the fact that (*i*) MC guarantees safety in CFSMs [14] and (*ii*) time constraints imply that a subset of the configurations reachable in the untimed setting are reachable in the timed setting (modulo clock valuations). Thus, if there is a deadlock or an orphan message configuration in the timed setting, there is one in the untimed setting, which contradicts the results in [14].

The projection $STS(S)\!\upharpoonright_{\mathtt{p}}$ of a synchronous transition system $STS(S)$ on a machine $\mathtt{p}$ is given by substituting each event $e \in E$ with its projection $e\!\upharpoonright_{\mathtt{p}}$, then minimising the automaton w.r.t. language equivalence. For example, the projections of $STS(S)$ onto $\mathtt{U}$, $\mathtt{W}$,

and $\mathtt{A}$ are isomorphic to the system S_{ST} in Fig. 1. Below $\sim$ denotes the standard timed bisimulation [15].

▶ **Theorem 7** (Equivalence). *If $S = (M_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$ is MC then $S \sim (STS(S)\!\upharpoonright_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$.*

Theorem 7 says that the behaviour of the original system is preserved by $STS(S)$, this result is crucial to be able to construct a global specification that is equivalent to a system of CTAs. It follows from the fact that, (i) if the system is MC, then all the machine's behaviour is preserved except for the receive actions that are never executed; and (ii) since we assume that the machines are deterministic w.r.t. messages, the projections of $STS(S)$ also preserve all required transitions.

4 Progress with Time Constraints

This section introduces a *progress* property for CTAs, ensuring that no communication mismatch prevents the progress of the overall system (cf. § 4.1). In § 4.2, we give a sufficient condition to guarantee progress in CTAs (cf. Theorem 13).

4.1 Progress Properties

We identify several types of errors, inspired by their counterparts in the (untimed) CFSM model, which may arise in timed communicating systems. Let $s = (\vec{q}; \vec{w}; \nu) \in RS(S)$; s is an ***unsuccessful reception configuration*** if there exists $\mathtt{r} \in \mathcal{P}$ such that $q_{\mathtt{r}}$ is a receiving state, and for all $(q_{\mathtt{r}}, \mathtt{sr}?a(g,\lambda), q'_{\mathtt{r}}) \in \delta_{\mathtt{r}}$ either (i) $w_{\mathtt{sr}} \neq \varepsilon$ and $w_{\mathtt{sr}} \notin a\mathbb{A}^*$ or (ii) $\forall t \in \mathbb{R}_{\geq 0} : \nu + t \not\models g$ (i.e., $\mathtt{r}$ cannot receive messages from any of its queues, as they either contain an unexpected message or none of the transition guards will ever be satisfied); and s is an ***unfeasible configuration*** if there exists $\mathtt{s} \in \mathcal{P}$ such that $q_{\mathtt{s}}$ is a sending state, and $(q_{\mathtt{s}}, \mathtt{sr}!a(g,\lambda), q'_{\mathtt{s}}) \in \delta_{\mathtt{s}}$ implies that $\forall t \in \mathbb{R}_{\geq 0} : \nu + t \not\models g$ (i.e., $\mathtt{s}$ is unable to send a message because none of its guards will ever be satisfied).

▶ **Definition 8** (Progress). S satisfies the *progress* property if for all $s \in RS(S)$, s is not a deadlock, an orphan message, an unsuccessful reception, nor an unfeasible configuration. ◀

Observe that the original semantics of CTAs in [17] and in Def. 2 do not allow us to identify unsuccessful reception or unfeasible configurations. From Def. 2, a system may take a time transition which *permanently* prevents a machine from firing further actions. Below, we adjust the semantics of CTAs and give examples of "undesirable" scenarios it prevents.

▶ **Definition 9** (Reachable configuration (2)). $s \xrightarrow{\alpha} s'$ is defined as Def. 2, replacing (3) with:

3. $\alpha = t \in \mathbb{R}_{>0}$, $\nu' = \nu + t$, $\forall\, \mathtt{pq} \in C :\ w'_{\mathtt{pq}} = w_{\mathtt{pq}}$, and $\forall\, \mathtt{p} \in \mathcal{P} :\ q'_{\mathtt{p}} = q_{\mathtt{p}}$ and
   - **a.** $q_{\mathtt{p}}$ sending $\implies \exists (q_{\mathtt{p}}, \ell, q''_{\mathtt{p}}) \in \delta_{\mathtt{p}} :\ \exists t' \in \mathbb{R}_{\geq 0} :\ \nu' + t' \models \mathsf{guard}(\ell)$
   - **b.** $\forall (q_{\mathtt{p}}, \mathtt{sp}?a(g,\lambda), q''_{\mathtt{p}}) \in \delta_{\mathtt{p}} :\ (w_{\mathtt{sp}} \in a\mathbb{A}^* \implies \exists t' \in \mathbb{R}_{\geq 0} : \nu' + t' \models g)$

Unless stated otherwise, we only consider this semantics hereafter. ◀

Condition (3a) handles the case of machines waiting to perform send actions, and (3b) handles receive transitions, as illustrated by the examples below:

Consider configuration $((q_0, q_2); \vec{\varepsilon}; \nu_0)$ in which $\mathtt{s}$ must send a message within 3 time units. Condition (3a) prevents a time transition with delay $t = 3$. Indeed, with a clock valuation

ν_0+3, none of the action of $\mathtt{s}$ from q_0 can be fired. Consider now configuration $((q_1,q_2);\vec{w};\nu)$ with $w_{\mathtt{sr}}=a$ and $\nu(x)=\nu(y)=3.5$. Condition (3b) rules out a time transition with $t=1$. Indeed, even if $\mathtt{r}$ has a transition whose guard will be enabled after time $\nu(y)+1=4.5$, i.e., $(q_2,\mathtt{sr}?b(y=5),q_3)$, this transition cannot be fired due to the content of queue $w_{\mathtt{sr}}\notin b\mathbb{A}^*$; on the other hand transition $(q_2,\mathtt{sr}?a(y=4),q_3)$ is no longer fireable, due to its time constraint.

4.2 A Sound Characterisation of Progress

Roadmap. We give a sound condition that guarantees progress in the presence of time constraints. The main property, *interaction-enabling* (IE) in Def. 12, essentially checks that future actions are possible. IE guarantees that: (1) whatever the past, each machine that is in a sending state is eventually able to fire one of its transitions and (2) for every message that is sent, there exists a (future) time where this message can be received. IE relies on checking whether an action ℓ is *progress enabling* (Def. 11) which ensures that, for all possible past clock valuations, there exists a future time where the guard of ℓ is satisfied.

In the rest of this section, we give (i) a procedure for understanding the past of a configuration, based on a graph modelling the causal dependencies between previously executed actions; and (ii) a procedure to check that, for any reachable configuration, there is always a future time where an available action can be fired.

Understanding the past. We check that S has progress by analysing paths, i.e., sequences of events, in $STS(S)$. Since $STS(S)$ gives an over-approximation of the causal dependencies between actions, we will construct a graph of the actual dependencies of the underlying actions of a path. We compute the underlying actions of a path via the function:

$$\mathsf{nodes}(e_1\cdots e_k)\stackrel{\text{def}}{=} e_1{\downarrow_{\mathsf{sid}(e_1)}}\cdot e_1{\downarrow_{\mathsf{rid}(e_1)}}\cdots e_k{\downarrow_{\mathsf{sid}(e_k)}}\cdot e_k{\downarrow_{\mathsf{rid}(e_k)}}\qquad(k\geq 0)$$

Remarkably, given a path φ and two actions ℓ_i and ℓ_j in $\mathsf{nodes}(\varphi)$, $i<j$ does not imply that there is a causal dependency between ℓ_i and ℓ_j. For instance, in

$$\mathsf{nodes}(\varphi)=\mathtt{sr}!a(x<10,\emptyset)\cdot\mathtt{sr}?a(10\leq y,\emptyset)\cdot\mathtt{sp}!a(x<10,\emptyset)\cdot\mathtt{sp}?a(10\leq z,\emptyset)$$

the two receive actions $\mathtt{sr}?a(10\leq y,\emptyset)$ and $\mathtt{sp}?a(10\leq z,\emptyset)$ may not always be executed in that order, since they are executed by two different participants.

The graph of dependencies of an action ℓ_k in a sequence of actions $\ell_1\cdots\ell_k$ (Def. 10 below) gives an abstraction of all actions on which ℓ_k depends. This is done by taking into account two kinds of dependencies: output/input dependencies between matching send and receive actions, and local dependencies within a single machine.

▶ **Definition 10** (Graph of Dependencies). Let $\mathsf{dep}(\varepsilon;\ell)\stackrel{\text{def}}{=}\emptyset$ and

$$\mathsf{dep}(\rho\cdot\ell_1;\ell_2)\stackrel{\text{def}}{=}\begin{cases}\{(\ell_1,\ell_2)\}\cup\mathsf{dep}(\rho;\ell_i)_{i=1,2} & \text{if } \ell_1=\mathtt{sr}!a(g_1,\lambda_1),\ \ell_2=\mathtt{sr}?a(g_2,\lambda_2)\\ \{(\ell_1,\ell_2)\}\cup\mathsf{dep}(\rho;\ell_1) & \text{if } \mathsf{subj}(\ell_1)=\mathsf{subj}(\ell_2)\\ \mathsf{dep}(\rho;\ell_2) & \text{otherwise}\end{cases}$$

The graph of dependencies of $\rho=\ell_1\cdots\ell_k$ $(k>0)$, written $\mathsf{DG}(\rho)$, is the graph (D,A) s.t. $A=\mathsf{dep}(\ell_1\cdots\ell_{k-1};\ell_k)\setminus\{(\ell_i,\ell_k)\,|\,1\leq i<k\}$ and $D=\{\ell_r{\not=}\ell_k\mid\exists(\ell_i,\ell_j)\in A\wedge r\in\{i,j\}\}$.[1] ◀

[1] For the sake of presentation, we write ℓ_i for the node (i,ℓ_i) in D where ℓ_i is an action in ρ and i is its position in ρ. This guarantees that each element in ρ is assigned a unique node in D.

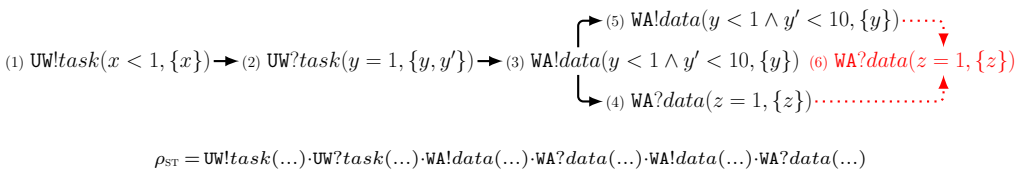


Figure 3 Graph of dependencies $\mathsf{DG}(\rho_{\text{ST}})$, in solid black, cf. Scheduled Task Protocol (Fig. 1).

$$\mathsf{idx}(\rho) \stackrel{\text{def}}{=} \{i \mid \ell_i \in D\} \qquad W_x^i(\rho) \stackrel{\text{def}}{=} \begin{cases} v_i - v_j & \text{if } 0 \leq j = max\{j < i \mid \ell_j \in D \wedge x \in \mathsf{reset}(\ell_j)\} \\ v_i & \text{otherwise} \end{cases}$$

$$\mathsf{allpast}(\rho) \stackrel{\text{def}}{=} \bigwedge_{i \in \mathsf{idx}(\rho)} \mathsf{absolute}_\rho(\ell_i)$$

$$\mathsf{elapse}(\rho) \stackrel{\text{def}}{=} \bigwedge_{(\ell_i,\ell_j) \in A} v_i \leq v_j \qquad \mathsf{absolute}_\rho(\ell_i) \stackrel{\text{def}}{=} \mathsf{guard}(\ell_i)\ \{x \mapsto W_x^i(\rho)\}_{x \in \mathcal{X}}$$

Figure 4 Functions on graphs of dependencies, where $\mathsf{DG}(\rho) = (D, A)$.

$\mathsf{DG}(\ell_1 \cdots \ell_k)$ is a graph whose nodes form a subset of $\{\ell_1, \ldots, \ell_{k-1}\}$ and whose edges model causal dependencies between actions (computed backwards starting from ℓ_k). In Fig. 3 (in solid black), we give the graph of dependencies of $\texttt{WA}?data(z = 1, \{z\})$ in the sequence ρ_{ST}, corresponding to an execution of the Scheduled Task Protocol.

Given a graph of dependencies $\mathsf{DG}(\rho)$, we define several functions that allow us to construct predicates modelling the past. The definitions of these functions are given in Fig. 4, where we fix $\mathsf{DG}(\rho) = (D, A)$. Below, we illustrate how they behave using $\mathsf{DG}(\rho_{\text{ST}})$ in Fig. 3. First, we transform the guard of an action ℓ_i such that its solutions are the possible *absolute* times (i.e., from the initial configuration of the system) in which one may execute ℓ_i (taking into account the last reset of each clock in ρ). In our example, we have:

$$\mathsf{absolute}_{\rho_{\text{ST}}}(\ell_5) = v_5 - v_3 < 1 \wedge v_5 - v_2 < 10 \qquad \text{with} \quad \ell_5 = \texttt{WA}!data(y < 1 \wedge y' < 10, \{y\})$$

Observe that clock y (resp. y') is replaced by the difference between variable v_5 and variable v_3 (resp. v_2) corresponding to the *latest* step where y (resp. y') was reset. Unifying, e.g., y and y' into v_5 models the fact that time elapses at the same pace for all clocks. Next, we aggregate the information in $\mathsf{DG}(\rho)$, by (i) recording the indices of all the actions on which ℓ_k depends ($\mathsf{idx}(\rho)$); (ii) taking the conjunction of all constraints in absolute time ($\mathsf{allpast}(\rho)$); and (iii) recording the fact that time never decreases between two causally dependent actions ($\mathsf{elapse}(\rho)$). Taking the dependencies for ρ_{ST} in Fig. 3, we have:

$$\mathsf{allpast}(\rho_{\text{ST}}) = v_1 < 1 \wedge v_2 = 1 \wedge (v_3 - v_2 < 1 \wedge v_3 - v_2 < 10) \wedge v_4 = 1 \wedge (v_5 - v_3 < 1 \wedge v_5 - v_2 < 10)$$
$$\mathsf{elapse}(\rho_{\text{ST}}) = v_1 \leq v_2 \wedge v_2 \leq v_3 \wedge v_3 \leq v_4 \wedge v_3 \leq v_5 \qquad \mathsf{idx}(\rho_{\text{ST}}) = \{1, 2, 3, 4, 5\}$$

Predicting the future. We now give the main definition of this section, allowing to check whether the past implies that there exists a satisfiable future. We use the functions defined above to check whether a given event in $STS(S)$ can indeed meet its time constraints.

▶ **Definition 11** (Progress enabling (PE)). A pair (n, e) is *progress enabling* (PE) for $\mathsf{p} \in \mathsf{id}(e)$ if for *all* paths φ such that $n_0 \stackrel{\varphi}{\hookrightarrow} n$, letting:

$$\rho = \begin{cases} \mathsf{nodes}(\varphi \cdot e) & \text{if } \mathtt{p} = \mathsf{rid}(e) \\ \mathsf{nodes}(\varphi) \cdot e|_{\mathsf{sid}(e)} & \text{otherwise} \end{cases}$$

and $k = |\rho|$, $\ell_k = e|_{\mathtt{p}}$, $\vec{v} = \{v_i \mid i \in \mathsf{idx}(\rho)\}$; the following holds

$$\forall \vec{v}\ \exists v_k : \ \mathsf{allpast}(\rho) \wedge \mathsf{elapse}(\rho) \implies \mathsf{absolute}_\rho(\ell_k) \wedge \textstyle\bigwedge_{v_i \in \vec{v}} v_i \leq v_k$$

A pair (n, φ) is *recursively progress enabling* (RPE) for $P \subseteq \mathcal{P}$ if $\varphi = \varepsilon$ and $P = \emptyset$; or if (n, e) is PE for $\mathsf{sid}(e)$ and for $\mathsf{rid}(e)$ and (n', φ') is RPE for $P \setminus \mathsf{id}(e)$ with $\varphi = e \cdot \varphi'$ and $n \overset{e}{\hookrightarrow} n'$.

Given a node n and an event e in $STS(S)$, and a participant $\mathtt{p}$, the above definition ensures that for all possible past clock valuations, there exists a *future* time where participant $\mathtt{p}$ has the possibility to execute action $e\,|_{\mathtt{p}}$. For instance, the pair $((\mathtt{U}_1, \mathtt{W}_1, \mathtt{A}_0), (\mathtt{W} \to \mathtt{A} : \mathit{data};\ y < 1 \wedge y' < 10, \{y\};\ z = 1, \{z\}))$ is PE for $\mathtt{A}$, notably because the following holds:

$$\forall v_1 \ldots v_5\ \exists v_6 : \ \mathsf{allpast}(\rho_{\mathrm{ST}}) \wedge \mathsf{elapse}(\rho_{\mathrm{ST}}) \implies (v_6 - v_4) = 1 \wedge v_1 \leq v_6 \ldots v_5 \leq v_6$$

Below, Def. 11 is used in $STS(S)$ to ensure progress of the overall system.

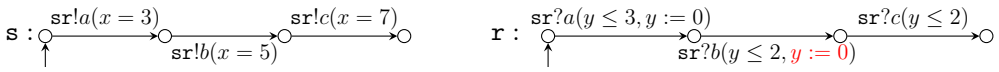
▶ **Definition 12** (Interaction enabling (IE)). A node $n \in N$ is interaction enabling (IE) if either (i) it is final or (ii) the following conditions hold:

1. There is $e \in E$ and φ such that $n \overset{e \cdot \varphi}{\hookrightarrow}$ and $(n, e \cdot \varphi)$ is RPE for $\mathsf{ids}(n)$;
2. For all $e \in E$ such that $n \overset{e}{\hookrightarrow} n'$, (n, e) is PE for $\mathsf{rid}(e)$, and n' is IE.

A system S is *interaction enabling* (IE) if n_0 is IE.

Def. 12 recursively checks the nodes of $STS(S)$ (starting from n_0) and for each ensures that: (1) there is at least one path, involving all the participants still active at node n, that is RPE, i.e., where each guard along that path is satisfied for any past; (2) each receive action is PE and its successor is IE (note that a send action is always a dependency of its receive action). Condition (1) ensures that no sender will be left in a configuration where it cannot send any message, due to time constraints being unsatisfiable; condition (2) ensures that a receive action is always feasible given that its corresponding send action was executed.

Examples. **(1)** The first example shows how resets affect the satisfiability of guards.

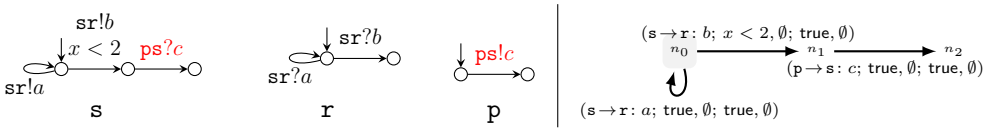


The system above is IE, notably, because the following holds:

$$\forall v_1 v_2 v_3 v_4 v_5\ \exists v_6 : \ v_1 = 3 \wedge v_2 \leq 3 \wedge v_3 = 5 \wedge v_4 - v_2 \leq 2 \wedge v_5 = 7 \wedge v_1 \leq \ldots \leq v_5 \implies v_6 - v_4 \leq 2 \wedge v_1 \leq v_6 \ldots v_5 \leq v_6$$

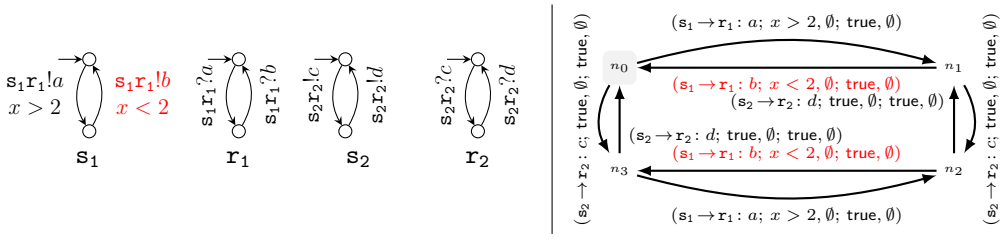
Notice that the resets of clock y (recorded by subtracting v_2 and v_4 in the formula above) allow $\mathtt{r}$ to receive message c before *absolute* time 7. If we modified the example by removing the second reset of y in machine $\mathtt{r}$, then the system would not be IE because message c would be expected before *absolute* time 5, while c can only be sent at time 7. In fact, the RHS of the implication above would become: $v_6 - v_2 \leq 2 \wedge v_1 \leq v_6 \ldots v_5 \leq v_6$.

(2) The second example shows a system of three machines, which violates IE (Def. 12).



If participant $\mathtt{s}$ does not send b before time 2, then message c (sent by $\mathtt{p}$), will never be received. This system is *not* IE because there is no path from n_0 that is RPE for $\{\mathtt{s}, \mathtt{r}, \mathtt{p}\}$. The only transition that is PE from n_0 is the loop on n_0 (which does not involve $\mathtt{p}$).

(3) The third example shows that IE captures a "global" notion of progress (i.e., *all* participants must able to proceed). Consider the system of four machines below:



this system is *not* IE. Indeed, although there is one RPE path outgoing node n_1 (machines $\mathtt{s}_2$ and $\mathtt{r}_2$ can continue interacting), there is no path that is RPE for *all* participants $\{\mathtt{s}_1, \mathtt{r}_1, \mathtt{s}_2, \mathtt{r}_2\}$. Observe that $\mathtt{s}_1$ is stuck in n_1, as the transition with label $\mathtt{s}_1\mathtt{r}_1!b(x < 2, \{x\})$ can never be fired by $\mathtt{s}_1$, i.e., $\forall v_0 \exists v_1 : v_0 > 2 \implies v_0 \leq v_1 < 2$ does not hold.

▶ **Theorem 13** (Progress). *Suppose S is multiparty compatible (Def. 4) and interaction enabling (Def. 12). (1) Then S satisfies the progress property. (2) For all $s = (\vec{q}; \vec{w}; \nu) \in RS(S)$, if there is $\mathtt{p} \in \mathcal{P}$ such that $q_{\mathtt{p}}$ is not final, then there is s' such that $s \to s'$.*

▶ **Theorem 14** (Decidability). *Interaction enabling (Def. 12) is decidable.*

The decidability of Def. 12 relies on the fact that the logic used in Def. 11 forms a subset of the Presburger arithmetic, which is decidable; and that it is enough to check *finite* paths in $STS(S)$. The complexity of the decision procedure is mostly affected by the enumeration of paths in $STS(S)$ (which can be reduced via partial order reduction techniques) and the satisfiability of Presburger formulae (which can be relegated to an SMT solver).

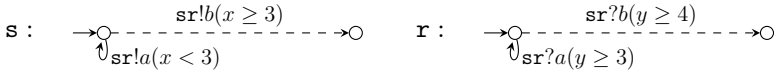
5 Non-Zenoness and Eventual Reception in CTAs

In the presence of time constraints, one needs to make sure that some participant's only possible way forward is not by firing actions at increasingly short intervals of time, i.e., by zeno behaviours. This is a common requirement in real-time systems [2], and it is justified by the assumption that *"any physical process, no matter how fast, cannot be infinitely fast"* [21].

In order to identify zeno behaviours in our systems, we assume without loss of generality that there is a special clock $\hat{x} \in \mathcal{X}$ which is *never* reset, i.e., for all $\mathtt{p} \in \mathcal{P}$ and all $(q, \ell, q') \in \delta_{\mathtt{p}} : \hat{x} \notin \mathsf{reset}(\ell)$. Hence, $\hat{x}$ keeps the absolute time since the beginning of the interactions. Let $s = (\vec{q}; \vec{w}; \nu)$ be a configuration of a system S, s is a ***zeno configuration*** if there exists $t \in \mathbb{R}_{\geq 0}$ such that for all $s' = (\vec{q'}; \vec{w'}; \nu')$, $s \to^* s'$ implies $\nu'(\hat{x}) < t$ and $s' \not\to s''$, for some s''.

▶ **Definition 15** (Non-zeno system). S is *non-zeno* (NZ) if $\forall s \in RS(S)$, s is not a zeno configuration.

The following example shows that a zeno configuration may still occur in systems that are multiparty compatible and interaction enabling.



The system above (*ignoring the dashed transitions*) satisfies MC and IE, e.g., $\forall v_0 \exists v_1 : v_0 <$

$3 \implies v_1 \geq 3 \wedge v_0 \leq v_1$, but is *not* NZ. Because of the upper bound $x < 3$ and the fact that x is not reset in the loop, machine $\mathtt{s}$ has to produce an *infinite* number of (send) actions in a *finite* amount of time (3 time units). A dramatic consequence of this zeno behaviour is that machine $\mathtt{r}$ will never be able to consume any message a due to the fact that constraint $y \geq 3$ will never be satisfied (cf. Def. 9). This system violates *eventual reception*, a property which guarantees that every message that is sent is eventually received. Formally, a system S satisfies ***eventual reception*** (ER) if for all $s = (\vec{q}; \vec{w}; \nu) \in RS(S)$, if $w_{\mathtt{sr}} \in a\mathbb{A}^*$, then $s \rightarrow^* \xrightarrow{\mathtt{sr}?a(g,\lambda)}$.

The system above (*considering the dashed transitions*) is NZ and satisfies ER: the dashed transitions offer an 'escape' from zeno-only behaviours where time can elapse and thus allow machine $\mathtt{r}$ to consume any messages that were sent. Observe that in general NZ alone is not sufficient to guarantee ER. However, ER is guaranteed for systems which validate all the condition presented in this paper, see Theorem 19 below.

The example also shows a fundamental difference between CTAs and models with synchronous communications, such as Networks of Timed Automata (NTAs) [2]. The work in [7] shows that it is sufficient that one machine in each loop of an NTA satisfies non-zenoness for the whole system to be non-zeno. This is not generally true for CTAs. In the example above (*ignoring the dashed transitions*), time cannot diverge despite the machine on the right being non-zeno.

Checking non-zenoness. Now we give a condition on $STS(S)$ that, together with MC, guarantees non-zenoness. A walk in $STS(S)$ is an alternating sequence $n_1 \cdot e_1 \cdot n_2 \cdots e_{k-1} \cdot n_k$ such that $n_i \overset{e_i}{\hookrightarrow} n_{i+1}$ for all $1 \leq i < k$. We let ω range over walks in $STS(S)$. A walk is *elementary* if $(n_i \cdot e_i) \neq (n_j \cdot e_j)$ for all $1 \leq i \neq j < k$. A (elementary) cycle in $STS(S)$ is a (elementary) walk $n_1 \cdot e_1 \cdot n_2 \cdots e_{k-1} \cdot n_k$ such that $n_1 = n_k$.

Given guard g and clock x, we say that g *is an upper bound* for x, written g is *UB* for x, if there is a sub-term $x \leq c$ in g (not under a negation) or a sub-term $c \leq x$ under a negation. We say that g *is strictly positive*, written g is *SP*, if for all clocks $x \in \mathsf{fc}(g)$ and for all sub-terms in g of the form $x \leq c$ (not under negation) or $c \leq x$ (under negation), $c \in \mathbb{Q}_{>0}$.

▶ **Definition 16** (Cycle enabling (CE)). System S is cycle enabling (CE) if for each elementary cycle ω in $STS(S)$, and for each clock x such that there is $(\mathtt{s} \to \mathtt{r} : a; g_{\mathtt{s}}, \lambda_{\mathtt{s}}; g_{\mathtt{r}}, \lambda_{\mathtt{r}})$ in ω and $g_{\mathtt{s}}$ is *UB* for x, the following holds, either

1. there are (i) $(\mathtt{p} \to \mathtt{q} : b; g_{\mathtt{p}}, \lambda_{\mathtt{p}}; g_{\mathtt{q}}, \lambda_{\mathtt{q}})$ in ω s.t. $x \in \lambda_{\mathtt{p}} \cup \lambda_{\mathtt{q}}$, and (ii) $(\mathtt{p}' \to \mathtt{q}' : b'; g_{\mathtt{p}'}, \lambda_{\mathtt{p}'}; g_{\mathtt{q}'}, \lambda_{\mathtt{q}'})$ in ω s.t. $g_{\mathtt{p}'}$ is *SP*; or
2. for each $(n_i \cdot e \cdot n_{i+1})$ in ω, there is $n' \neq n_i \in N$ and $e' \neq e \in E$ such that $\mathsf{id}(e) = \mathsf{id}(e')$, $n_i \overset{e'}{\hookrightarrow} n'$, and (n_i, e') is PE for $\mathsf{sid}(e')$ ◀

Condition (1) adapts *structural non-zenoness* from [22] to CTAs by requiring that: (i) each x is reset in ω, and (ii) it is possible to let some time elapse at each iteration. Condition (2) requires that the "escape" event e', leading to a different node n', can *always* be taken. Our running example satisfies CE (Def. 16); $STS(S_{\mathrm{ST}})$ has one (elementary) cycle in which two clocks have an upper bound: clock y satisfies (1) since it is reset and the guards have upper bounds strictly greater than 0 in the cycle; clock y' satisfies (2) since there is an escape event, $e' = (\mathtt{W} \to \mathtt{A} : \mathtt{end}; y < 1, \emptyset; z = 1, \{z\})$, which is PE for $\mathtt{W}$.

▶ **Theorem 17** (Non-zenoness). *If S is MC and CE, then S is non-zeno.*

▶ **Theorem 18** (Decidability). *Cycle enabling (Def. 16) is decidable.*

▶ **Theorem 19** (Eventual reception). *If S is MC, IE, and CE, then S satisfies ER.*

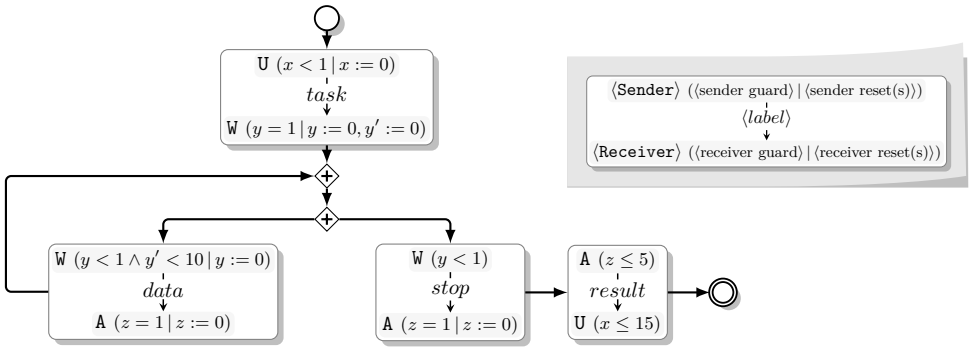


Figure 5 Timed choreography for the Scheduled Task Protocol (S_{ST}).

6 Applications and Implementation

Constructing global specifications. Our theory can be easily applied and integrated with other works, to construct sound (i.e., satisfying safety, progress, and non-zenoness) timed global specifications, such as (syntactic) multiparty session types [16, 6], or graphical choreographies [19, 13, 8]. Thanks to Theorem 7, we can build on the algorithm in [14] to construct (syntactic) timed global types from CTAs. In Appendix [3], we give the formal definitions of the adaptation of the algorithm in [14]. Given an MC system S our algorithm generates a timed global type [6] equivalent to the original system S (i.e., its projections are timed bisimilar to those of S). This implies that if S is IE (resp. CE) then the constructed timed global type will also enjoy progress (resp. non-zenoness). Similarly, building on the algorithm in [19], we obtain a *graphical* representation reminiscent of BPMN Choreographies, see [8, 19]. When applied to the Scheduled Task Protocol, the algorithm adapted from [19] produces the choreography in Fig. 5; giving a much clearer specification for S_{ST}.

Implementation. To assess the applicability and cost of our theory, we have integrated our theory into the tool first introduced in [19], which builds graphical choreographies from CFSMs. Our tool [3] (implemented in Haskell and using Z3) takes as input a textual representation of CTAs on which each condition (MC, IE, and CE) is checked for, and produces an equivalent choreography (such as the one in Fig. 5). The results of our experiments (executed on a Intel i7 computer, with 16GB of RAM) are below; where $|\mathcal{P}|$ is the number of machines, and $|N|$ (resp. $|\hookrightarrow|$) is the number of nodes (resp. transitions) in $STS(S)$.

S	$\|\mathcal{P}\|$	$\|N\|$	$\|\hookrightarrow\|$	MC	IE	CE	s		$\|\mathcal{P}\|$	$\|N\|$	$\|\hookrightarrow\|$	MC	IE	CE	s
Running Example	3	4	4	✓	✓	✓	0.41	×4	12	256	1024	✓	✓	✓	28.49
Bargain	3	5	5	✓	✓	✓	0.44	×2	6	25	50	✓	✓	✓	12.30
Temp. calculation [6]	3	6	6	✓	✓	✓	0.45	×2	6	36	72	✓	✓	✓	9.24
Word Count [20]	3	6	6	✓	✓	✓	0.41	×2	6	36	72	✓	✓	✓	8.63
ATM (Template) [11]	3	9	8	✓	✓	✓	0.36	×3	9	729	1944	✓	✓	✓	94.01
ATM (Instance) [11]	3	9	8	✓	✓	✓	0.53	×3	9	729	1944	✓	✓	✓	96.09
Consumer-Producer [11]	2	1	1	✓	✓	✓	0.16	×5	10	1	5	✓	✓	✓	43.19
Fischers Mutual Excl. [5]	2	4	3	✓	✓	✓	0.21	×4	8	256	768	✓	✓	✓	3.19

Most of the protocols are taken from the literature and all are checked within a minute on average. For the sake of space, we have used small examples throughout the paper, however our benchmarks include bigger protocols (up-to 12 machines), which have comparable size with those we encountered through our collaboration with Cognizant [23, 19]. Since the size of the *STS* is the most critical parameter for scalability, we have tested systems consisting of the parallel composition of several instances of a protocol. For instance, *Running Example* ×4 is the parallel composition of four instances of S_{ST}, cf. Fig. 1.

7 Conclusions and Related Work

Our results are summarised in the table below. Multiparty compatibility (MC) gives (*i*) an equivalence between an MC system and a system consisting of the projections of its *STS*; and (*ii*) a sufficient condition for *safety*. MC and interaction enabling (IE) form a sufficient condition for *progress*; while MC and cycle enabling (CE) form a sufficient condition for *non-zenoness* (NZ). Together, MC, IE, and CE ensure safety, progress, NZ, and *eventual reception* (ER).

Property	$S \sim (STS(S)\upharpoonright_{\mathtt{p}})_{\mathtt{p}\in\mathcal{P}}$	Safety	Progress	Non-Zeno	ER
MC (Def. 4)	✓	✓	✗	✗	✗
MC+IE (Def. 12)	✓	✓	✓	✗	✗
MC+CE (Def. 16)	✓	✓	✗	✓	✗
MC+IE+CE	✓	✓	✓	✓	✓

Multiparty session types. The work in [6] studies a typing system for a timed π-calculus using timed global types. A class of CTAs which are safe and have progress is given in [6] via projection of (well-formed) timed global types onto timed local types (which correspond to deterministic, non-mixed, and directed CTAs). Well-formedness yields conditions on CTAs that are more restrictive than the ones given in this paper. For instance, the system in Fig. 1, which is safe and enjoys progress, is ruled out by the conditions in [6]. In addition, this paper gives sufficient conditions for CTAs to belong to the class of safe CTAs with progress, which was left as an open problem in [6]. The construction of timed global types from either local types or CTAs is not addressed in [6]. Recently, [4] introduced a compliance and sub-typing relation for *binary* timed session types *without queues* (synchronous communication semantics). The existing works for constructing global specifications from local specifications [18, 14, 19] only apply to *untimed* models. Our conditions (IE and CE) are given independently of the definition of MC. The use of a more general notion of MC, as the one given in [19], would allow us to lift the assumptions that the machines are directed and have no mixed states (cf. § 2). Hence, we could capture more general timed choreographies.

Reachability and decidability. When extending NTAs [2] with unbounded channels, reachability is no longer decidable in general [17]. Existing work tackles undecidability by restricting the network topologies [12, 17] or the channel size [1]. We give general (w.r.t. topology and channel size) decidable conditions ensuring that a configuration violating safety, progress, or NZ will not be reached. Observe that the scenario in Fig. 1 would be ruled out in [17] (its topology is not a polyforest) and in [1] ($w_{\mathtt{WA}}$ is unbounded). Our conditions are based, instead, on the conversation structures, which also enable the construction of global specifications.

Non-zeno conditions. In § 5 we set the conditions for time divergence, by ruling out specifications in which the only way forward is a zeno behaviour. This condition is called time progress in [2] and it is built-in in the definition of runs of a TA. Several conditions have been proposed to ensure absence of non-zeno behaviours in TAs: some, e.g., [21], do not allow any zeno execution, and some, e.g., [7], and this work (cf. Def. 15), ensure that there is always a non-zeno way forward. The condition in [7] can be checked with a simple form of reachability analysis which introduced the notion of 'escape' from a zeno loop, which we also use. [21, 7] consider Networks of TAs (NTAs), which do not feature asynchrony nor unbounded channels.

Acknowledgements. We would like to thank the ZDLC team at Cognizant for their stimulating conversations and Dominic Orchard for some (very useful) Haskell tips. This

work is partially supported by UK EPSRC projects EP/K034413/1, EP/K011715/1, and EP/L00058X/1; and by EU 7FP project under grant agreement 612985 (UPSCALE).

References

1 S. Akshay, Paul Gastin, Madhavan Mukund, and K. Narayan Kumar. Model checking time-constrained scenario-based specifications. In *FSTTCS*, volume 8 of *LIPIcs*, pages 204–215, 2010.

2 Rajeev Alur and David L. Dill. A theory of timed automata. *TCS*, 126:183–235, 1994.

3 Webpage of this paper, 2015. `http://www.doc.ic.ac.uk/~jlange/cta/`.

4 Massimo Bartoletti, Tiziana Cimoli, Maurizio Murgia, Alessandro Sebastian Podda, and Livio Pompianu. Compliance and subtyping in timed session types. In *FORTE*, volume 9039 of *LNCS*, pages 161–177. Springer, 2015.

5 Johan Bengtsson et al. Uppaal - a tool suite for automatic verification of real-time systems. In *Hybrid Systems III*, volume 1066 of *LNCS*, pages 232–243. Springer, 1996.

6 Laura Bocchi, Weizhen Yang, and Nobuko Yoshida. Timed multiparty session types. In *CONCUR*, volume 8704 of *LNCS*, pages 419–434. Springer, 2014.

7 Howard Bowman and Rodolfo Gómez. How to stop time stopping. *FAC*, 18(4):459–493, 2006.

8 BPMN 2.0 Choreography, 2012. `http://en.bpmn-community.org/tutorials/34/`.

9 Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *JACM*, 30(2):323–342, 1983.

10 Gérard Cécé and Alain Finkel. Verification of programs with half-duplex communication. *I&C*, 202(2):166–190, 2005.

11 Prakash Chandrasekaran and Madhavan Mukund. Matching scenarios with timing constraints. In *FORMATS*, volume 4202 of *LNCS*, pages 98–112. Springer, 2006.

12 Lorenzo Clemente, Frédéric Herbreteau, Amelie Stainer, and Grégoire Sutre. Reachability of communicating timed processes. In *FOSSACS*, volume 7794 of *LNCS*, pages 81–96. Springer, 2013.

13 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In *ESOP*, volume 7211 of *LNCS*, pages 194–213. Springer, 2012.

14 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP (2)*, volume 7966 of *LNCS*, pages 174–186. Springer, 2013.

15 Uno Holmer, Kim Guldstrand Larsen, and Wang Yi. Deciding properties of regular real time processes. In *CAV*, volume 575 of *LNCS*, pages 443–453. Springer, 1991.

16 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *POPL*, pages 273–284. ACM, 2008.

17 Pavel Krcál and Wang Yi. Communicating timed automata: The more synchronous, the more difficult to verify. In *CAV*, volume 4144 of *LNCS*, pages 249–262, 2006.

18 Julien Lange and Emilio Tuosto. Synthesising Choreographies from Local Session Types. In *CONCUR*, volume 7454 of *LNCS*, pages 225–239. Springer, 2012.

19 Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From communicating machines to graphical choreographies. In *POPL*, pages 221–232, 2015.

20 Rumyana Neykova, Laura Bocchi, and Nobuko Yoshida. Timed runtime monitoring for multiparty conversations. In *BEAT*, volume 162 of *EPTCS*, pages 19–26, 2014.

21 Stavros Tripakis. Verifying progress in timed systems. In *Formal Methods for Real-Time and Probabilistic Systems*, volume 1601 of *LNCS*, pages 299–314. Springer, 1999.

22 Stavros Tripakis, Sergio Yovine, and Ahmed Bouajjani. Checking timed büchi automata emptiness efficiently. *Formal Methods in System Design*, 26(3):267–292, 2005.

23 Zero Deviation Lifecycle. `http://www.zdlc.co`.

To Reach or not to Reach? Efficient Algorithms for Total-Payoff Games*

Thomas Brihaye[1], Gilles Geeraerts[2], Axel Haddad[1], and Benjamin Monmege[2]

1 **Université de Mons, Belgium,** thomas.brihaye,axel.haddad@umons.ac.be
2 **Université libre de Bruxelles, Belgium,** gigeerae,benjamin.monmege@ulb.ac.be

Abstract

Quantitative games are two-player zero-sum games played on directed weighted graphs. Total-payoff games – that can be seen as a refinement of the well-studied mean-payoff games – are the variant where the payoff of a play is computed as the sum of the weights. Our aim is to describe the first pseudo-polynomial time algorithm for total-payoff games in the presence of arbitrary weights. It consists of a non-trivial application of the value iteration paradigm. Indeed, it requires to study, as a milestone, a refinement of these games, called min-cost reachability games, where we add a reachability objective to one of the players. For these games, we give an efficient value iteration algorithm to compute the values and optimal strategies (when they exist), that runs in pseudo-polynomial time. We also propose heuristics to speed up the computations.

1998 ACM Subject Classification D.2.4 Software/Program Verification, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Games on graphs, Reachability, Quantitative games, Value iteration

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.297

1 Introduction

Games played on graphs are nowadays a well-studied and well-established model for the computer-aided design of computer systems, as they enable *automatic synthesis* of systems that are *correct-by-construction*. Of particular interest are *quantitative games*, that allow one to model precisely *quantitative* parameters of the system, such as energy consumption. In this setting, the game is played by two players on a directed weighted graph, where the edge weights model, for instance, a cost or a reward associated to the moves of the players. Each vertex of the graph belongs to one of the two players who compete by moving a token along the graph edges, thereby forming an infinite path called a *play*. With each play is associated a real-valued *payoff* computed from the sequence of edge weights along the play. The traditional payoffs that have been considered in the literature include total-payoff [10], mean-payoff [7] and discounted-payoff [17]. In this quantitative setting, one player aims at maximising the payoff while the other tries to minimise it. So one wants to compute, for each player, the best payoff that he can guarantee from each vertex, and the associated optimal strategies (i.e., that guarantee the optimal payoff no matter how the adversary is playing).

Such quantitative games have been extensively studied in the literature. Their associated decision problems (*is the value of a given vertex above a given threshold?*) are known to be

* The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under Grant Agreement no601148 (CASSTING).

in NP ∩ co-NP. Mean-payoff games have arguably been best studied from the algorithmic point of view. A landmark is Zwick and Paterson's pseudo-polynomial time (i.e., polynomial in the weighted graph when weights are encoded in unary) algorithm [17], using the *value iteration* paradigm that consists in computing a sequence of vectors of values that converges towards the optimal values of the vertices. After a fixed, pseudo-polynomial, number of steps, the computed values are precise enough to deduce the actual values of all vertices. Better pseudo-polynomial time algorithms have later been proposed, e.g., in [1, 4, 6], also achieving sub-exponential expected running time by means of randomisation.

In this paper, we focus on *total-payoff games*. Given an infinite play π, we denote by $\pi[k]$ the prefix of π of length k, and by $\mathbf{TP}(\pi[k])$ the (finite) sum of all edge weights along this prefix. The *total-payoff* of π, $\mathbf{TP}(\pi)$, is the inferior limit of all those sums, i.e., $\mathbf{TP}(\pi) = \liminf_{k\to\infty} \mathbf{TP}(\pi[k])$. Compared to mean-payoff (and discounted-payoff) games, the literature on total-payoff games is less extensive. Gimbert and Zielonka have shown [10] that optimal memoryless strategies always exist for both players and the best algorithm to compute the values runs in exponential time [9], and consists in iteratively improving strategies. Other related works include *energy games* where one player tries to optimise its energy consumption (computed again as a sum), keeping the energy level always above 0 (which makes difficult to apply techniques solving those games in the case of total-payoff); and a probabilistic variant of total-payoff games, where the weights are restricted to be non-negative [5]. Yet, we argue that the total-payoff objective is interesting as a *refinement* of the mean-payoff. Indeed, recall first that the total-payoff is finite if and only if the mean-payoff is null. Then, the computation of the total-payoff enables a finer, two-stage analysis of a game $\mathcal{G}$: (i) compute the mean payoff $\mathbf{MP}(\mathcal{G})$; (ii) subtract $\mathbf{MP}(\mathcal{G})$ from all edge weights, and scale the resulting weights if necessary to obtain integers. At that point, one has obtained a new game $\mathcal{G}'$ with null mean-payoff; (iii) compute $\mathbf{TP}(\mathcal{G}')$ to *quantify the amount of fluctuation around the mean-payoff* of the original game. Unfortunately, so far, no efficient (i.e., pseudo-polynomial time) algorithms for total-payoff games have been proposed, and straightforward adaptations of Zwick and Paterson's value iteration algorithm for mean-payoff do not work, as we demonstrate at the end of Section 2. In the present article, we fill in this gap by introducing the first pseudo-polynomial time algorithm for computing the values in total-payoff games.

Our solution is a non-trivial value iteration algorithm that proceeds through nested fixed points (see Algorithm 2). A play of a total-payoff game is infinite by essence. We transform the game so that one of the players (the minimiser) must ensure a *reachability objective*: we assume that the game ends once this reachability objective has been met. The intuition behind this transformation, that stems from the use of an inferior limit in the definition of the total-payoff, is as follows: in any play π whose total-payoff is *finite*, there is a position ℓ in the play after which all the partial sums $\mathbf{TP}(\pi[i])$ (with $i \geqslant \ell$) will be larger than or equal to the total-payoff $\mathbf{TP}(\pi)$ of π, and infinitely often both will be equal. For example, consider the game depicted in Figure 1(a), where the maximiser player (henceforth called Max) plays with the round vertices and the minimiser (Min) with the square vertices. For both players, the optimal value when playing from v_1 is 2, and the play $\pi = v_1v_2v_3\ v_4v_5\ v_4v_3\ (v_4v_5)^\omega$ reaches this value (i.e., $\mathbf{TP}(\pi) = 2$). Moreover, for all $k \geqslant 7$: $\mathbf{TP}(\pi[k]) \geqslant \mathbf{TP}(\pi)$, and infinitely many prefixes ($\pi[8]$, $\pi[10]$, $\pi[12]$, ...) have a total-payoff of 2, as shown in Figure 1(b).

Based on this observation, we transform a total-payoff game $\mathcal{G}$, into a new game that has *the same value as the original total-payoff game* but incorporates a reachability objective for Min. Intuitively, in this new game, we allow a new action for Min: after each play prefix $\pi[k]$, he can ask to *stop the game*, in which case the payoff of the play is the payoff $\mathbf{TP}(\pi[k])$

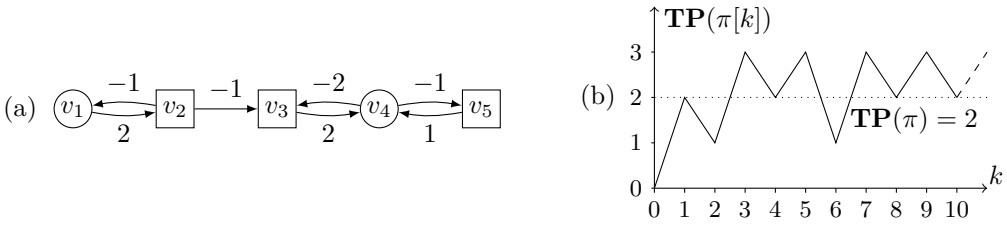


Figure 1 (a) A total-payoff game, and (b) the evolution of the partial sums in π.

of the prefix. However, allowing Min to stop the game at any moment would not allow to obtain the same value as in the original total-payoff game: for instance, in the example of Figure 1(a), Min could secure value 1 by asking to stop after $\pi[2]$, which is strictly smaller that the actual total-payoff (2) of the whole play π. So, we allow Max to *veto* to stop the game, in which case both must go on playing. Again, allowing Max to turn down all of Min's requests would be unfair, so we parametrise the game with a natural number K, which is the maximal number of vetoes that Max can play (and we denote by $\mathcal{G}^K$ the resulting game). For the *play* depicted in Figure 1(b), letting $K = 3$ is sufficient: trying to obtain a better payoff than the optimal, Min could request to stop after $\pi[0]$, $\pi[2]$ and $\pi[6]$, and Max can veto these three requests. After that, Max can safely accept the next request of Min, since the total payoff of all prefixes $\pi[k]$ with $k \geqslant 6$ are larger than or equal to $\mathbf{TP}(\pi) = 2$. Our key technical contribution is to show that *for all total-payoff games, there exists a finite, pseudo-polynomial, value of K such that the values in $\mathcal{G}^K$ and $\mathcal{G}$ coincide* (assuming all values are finite in $\mathcal{G}$: we treat the $+\infty$ and $-\infty$ values separately). Now, assume that, when Max accepts to stop the game (possibly because he has exhausted the maximal number K of vetoes), the game moves to a *target state*, and stops. By doing so, we effectively reduce the computation of the values in the total-payoff game $\mathcal{G}$ to the computation of the values in the total-payoff game $\mathcal{G}^K$ *with an additional reachability objective* (the target state) for Min.

In the following, such refined total-payoff games – where Min *must* reach a designated target vertex – will be called *min-cost reachability games*. Failing to reach the target vertices is the worst situation for Min, so the payoff of all plays that do not reach the target is $+\infty$, irrespective of the weights along the play. Otherwise, the payoff of a play is the sum of the weights up to the first occurrence of the target. As such, this problem nicely generalises the classical shortest path problem in a weighted graph. In the one-player setting (considering the point of view of Min for instance), this problem can be solved in polynomial time by Dijkstra's and Floyd-Warshall's algorithms when the weights are non-negative and arbitrary, respectively. In [11], Khachiyan *et al.* propose an extension of Dijkstra's algorithm to handle the two-player, non-negative weights case. However, in our more general setting (two players, arbitrary weights), this problem has, as far as we know, not been studied as such, except that the associated decision problem is known to be in NP ∩ co-NP [8]. A pseudo-polynomial time algorithm to solve a very close problem, called the *longest shortest path problem* has been introduced by Björklund and Vorobyov [1] to eventually solve mean-payoff games. However, because of this peculiar context of mean-payoff games, their definition of the length of a path differs from our definition of the payoff and their algorithm cannot be easily adapted to solve our min-cost reachability problem. Thus, as a second contribution, we show that a value iteration algorithm enables us to compute in pseudo-polynomial time the values of a min-cost reachability game. We believe that min-cost reachability games bear their own

potential theoretical and practical applications[1]. Those games are discussed in Section 3. In addition to the pseudo-polynomial time algorithm to compute the values, we show how to compute optimal strategies for both players and characterise them: there is always a memoryless strategy for the maximiser player, but we exhibit an example (see Figure 2(*a*)) where the minimiser player needs (finite) memory. Those results on min-cost reachability games are exploited in Section 4 where we introduce and prove correct our efficient algorithm for total-payoff games.

Finally, we briefly present our implementation in Section 5, using as a core the numerical model-checker PRISM. This allows us to describe some heuristics able to improve the practical performances of our algorithms for total-payoff games and min-cost reachability games on certain subclasses of graphs. More technical explanations and full proofs may be found in an extended version of this article [2].

2 Quantitative games with arbitrary weights

We denote by $\mathbb{Z}$ the set of integers, and $\mathbb{Z}_\infty = \mathbb{Z} \cup \{-\infty, +\infty\}$. The set of vectors indexed by V with values in S is denoted by S^V. We let $\preccurlyeq$ be the pointwise order over $\mathbb{Z}_\infty^V$, where $x \preccurlyeq y$ if and only if $x(v) \leqslant y(v)$ for all $v \in V$.

We consider two-player turn-based games on weighted graphs and denote the two *players* by Max and Min. A *weighted graph* is a tuple $\langle V, E, \omega \rangle$ where $V = V_{\mathsf{Max}} \uplus V_{\mathsf{Min}}$ is a finite set of vertices partitioned into the sets V_{Max} and V_{Min} of Max and Min respectively, $E \subseteq V \times V$ is a set of *directed edges*, $\omega \colon E \to \mathbb{Z}$ is the *weight function*, associating an integer weight with each edge. In our drawings, Max vertices are depicted by circles; Min vertices by boxes. For every vertex $v \in V$, the set of successors of v by E is denoted by $E(v) = \{v' \in V \mid (v, v') \in E\}$. Without loss of generality, we assume that every graph is deadlock-free, i.e., for all vertices v, $E(v) \neq \emptyset$. Finally, throughout this article, we let $W = \max_{(v,v') \in E} |\omega(v, v')|$ be the greatest edge weight (in absolute value) in the game graph. A *finite play* is a finite sequence of vertices $\pi = v_0 v_1 \cdots v_k$ such that for all $0 \leqslant i < k$, $(v_i, v_{i+1}) \in E$. A *play* is an infinite sequence of vertices $\pi = v_0 v_1 \cdots$ such that every finite prefix $v_0 \cdots v_k$, denoted by $\pi[k]$, is a finite play.

The total-payoff of a finite play $\pi = v_0 v_1 \cdots v_k$ is obtained by summing up the weights along π, i.e., $\mathbf{TP}(\pi) = \sum_{i=0}^{k-1} \omega(v_i, v_{i+1})$. In the following, we sometimes rely on the mean-payoff to obtain information about total-payoff objectives. The *mean-payoff* computes the average weight of π, i.e., if $k \geqslant 1$, $\mathbf{MP}(\pi) = \frac{1}{k} \sum_{i=0}^{k-1} \omega(v_i, v_{i+1})$, and $\mathbf{MP}(\pi) = 0$ when $k = 0$. These definitions are lifted to infinite plays as follows. The total-payoff of a play π is given by $\mathbf{TP}(\pi) = \liminf_{k \to \infty} \mathbf{TP}(\pi[k])$.[2] Similarly, the mean-payoff of a play π is given by $\mathbf{MP}(\pi) = \liminf_{k \to \infty} \mathbf{MP}(\pi[k])$. A weighted graph equipped with these payoffs is called a *total-payoff game* or a *mean-payoff game*, respectively.

A *strategy* for Max (respectively, Min) in a game $\mathcal{G} = \langle V, E, \omega, \mathbf{P} \rangle$ (with $\mathbf{P}$ one of the previous payoffs), is a mapping $\sigma \colon V^* V_{\mathsf{Max}} \to V$ (respectively, $\sigma \colon V^* V_{\mathsf{Min}} \to V$) such that for all sequences $\pi = v_0 \cdots v_k$ with $v_k \in V_{\mathsf{Max}}$ (respectively, $v_k \in V_{\mathsf{Min}}$), $(v_k, \sigma(\pi)) \in E$. A play or finite play $\pi = v_0 v_1 \cdots$ conforms to a strategy σ of Max (respectively, Min) if for all k such that $v_k \in V_{\mathsf{Max}}$ (respectively, $v_k \in V_{\mathsf{Min}}$), $v_{k+1} = \sigma(\pi[k])$. A strategy σ is *memoryless*

[1] An example of practical application would be to perform controller synthesis taking into account energy consumption. On the other hand, the problem of computing the values in certain classes of priced timed games has recently been reduced to computing the values in min-cost reachability games [3].

[2] Our results can easily be extended by substituting a lim sup for the lim inf. The lim inf is more natural since we adopt the point of view of the maximiser Max, hence the lim inf is the *worst* partial sum seen infinitely often.

if for all finite plays π, π', we have $\sigma(\pi v) = \sigma(\pi' v)$ for all v. A strategy σ is said to be *finite-memory* if it can be encoded in a deterministic Moore machine, $\langle M, m_0, \mathsf{up}, \mathsf{dec}\rangle$, where M is a finite set representing the memory of the strategy, with an initial memory content $m_0 \in M$, $\mathsf{up}\colon M \times V \to M$ is a memory-update function, and $\mathsf{dec}\colon M \times V \to V$ a decision function such that for every finite play π and vertex v, $\sigma(\pi v) = \mathsf{dec}(\mathsf{mem}(\pi v), v)$ where $\mathsf{mem}(\pi)$ is defined by induction on the length of the finite play π as follows: $\mathsf{mem}(v_0) = m_0$, and $\mathsf{mem}(\pi v) = \mathsf{up}(\mathsf{mem}(\pi), v)$. We say that $|M|$ is the *size* of the strategy.

For all strategies σ_{Max} and σ_{Min}, for all vertices v, we let $\mathsf{Play}(v, \sigma_{\mathsf{Max}}, \sigma_{\mathsf{Min}})$ be the outcome of σ_{Max} and σ_{Min}, defined as the unique play conforming to σ_{Max} and σ_{Min} and starting in v. Naturally, the objective of Max is to maximise its payoff. In this model of zero-sum game, Min then wants to minimise the payoff of Max. Formally, we let $\mathsf{Val}_{\mathcal{G}}(v, \sigma_{\mathsf{Max}})$ and $\mathsf{Val}_{\mathcal{G}}(v, \sigma_{\mathsf{Min}})$ be the respective values of the strategies, defined as (recall that $\mathbf{P}$ is either $\mathbf{TP}$ or $\mathbf{MP}$): $\mathsf{Val}_{\mathcal{G}}(v, \sigma_{\mathsf{Max}}) = \inf_{\sigma_{\mathsf{Min}}} \mathbf{P}(\mathsf{Play}(v, \sigma_{\mathsf{Max}}, \sigma_{\mathsf{Min}}))$ and $\mathsf{Val}_{\mathcal{G}}(v, \sigma_{\mathsf{Min}}) = \sup_{\sigma_{\mathsf{Max}}} \mathbf{P}(\mathsf{Play}(v, \sigma_{\mathsf{Max}}, \sigma_{\mathsf{Min}}))$. Finally, for all vertices v, we let $\underline{\mathsf{Val}}_{\mathcal{G}}(v) = \sup_{\sigma_{\mathsf{Max}}} \mathsf{Val}_{\mathcal{G}}(v, \sigma_{\mathsf{Max}})$ and $\overline{\mathsf{Val}}_{\mathcal{G}}(v) = \inf_{\sigma_{\mathsf{Min}}} \mathsf{Val}_{\mathcal{G}}(v, \sigma_{\mathsf{Min}})$ be the *lower* and *upper values* of v respectively. We may easily show that $\underline{\mathsf{Val}}_{\mathcal{G}} \preccurlyeq \overline{\mathsf{Val}}_{\mathcal{G}}$. We say that strategies σ^*_{Max} of Max and σ^*_{Min} of Min are optimal if, for all vertices v: $\mathsf{Val}_{\mathcal{G}}(v, \sigma^*_{\mathsf{Max}}) = \underline{\mathsf{Val}}_{\mathcal{G}}(v)$ and $\mathsf{Val}_{\mathcal{G}}(v, \sigma^*_{\mathsf{Min}}) = \overline{\mathsf{Val}}_{\mathcal{G}}(v)$ respectively. We say that a game $\mathcal{G}$ is *determined* if for all vertices v, its lower and upper values are equal. In that case, we write $\mathsf{Val}_{\mathcal{G}}(v) = \underline{\mathsf{Val}}_{\mathcal{G}}(v) = \overline{\mathsf{Val}}_{\mathcal{G}}(v)$, and refer to it as the *value* of v. If the game is clear from the context, we may drop the index $\mathcal{G}$ of all previous values. Mean-payoff and total-payoff games are known to be determined, with the existence of optimal memoryless strategies [17, 10].

Total-payoff games have been mainly considered as a refinement of mean-payoff games [10]. Indeed, if the mean-payoff value of a game is positive (respectively, negative), its total-payoff value is necessarily $+\infty$ (respectively, $-\infty$). When the mean-payoff value is 0 however, the total-payoff is necessarily different from $+\infty$ and $-\infty$, hence total-payoff games are particularly useful in this case. Deciding whether the total-payoff value of a vertex is positive can be achieved in NP $\cap$ co-NP. In [9], the complexity is refined to UP $\cap$ co-UP, and values are shown to be effectively computable solving nested fixed point equations with a strategy iteration algorithm working in exponential time in the worst case.

Our aim is to give a pseudo-polynomial algorithm solving total-payoff games. In many cases, (e.g., mean-payoff games), a successful way to obtain such an efficient algorithm is the *value iteration paradigm*. Intuitively, value iteration algorithms compute successive approximations $x_0, x_1, \ldots, x_i, \ldots$ of the game value by restricting the number of turns that the players are allowed to play: x_i is the vector of optimal values achievable when the players play at most i turns. The sequence of values is computed by means of an operator $\mathcal{F}$, letting $v_{i+1} = \mathcal{F}(v_i)$ for all i. Good properties (Scott-continuity and monotonicity) of $\mathcal{F}$ ensure convergence towards its smallest or greatest fixed point (depending on the value of x_0), which, in some cases, happens to be the value of the game. Let us briefly explain why such a simple approach fails with total-payoff games. In our case, the operator $\mathcal{F}$ is such that $\mathcal{F}(x)(v) = \max_{v' \in E(v)} \omega(v, v') + x(v')$ for all $v \in V_{\mathsf{Max}}$ and $\mathcal{F}(x)(v) = \min_{v' \in E(v)} \omega(v, v') + x(v')$ for all $v \in V_{\mathsf{Min}}$. This definition matches the intuition that x_i are optimal values after i turns.

Then, consider the example of Figure 1(a), limited to vertices $\{v_3, v_4, v_5\}$ for simplicity. Observe that there are two simple cycles with weight 0, hence the total-payoff value of this game is finite. Max has the choice between cycling into one of these two cycles. It is easy to check that Max's optimal choice is to enforce the cycle between v_4 and v_5, securing a payoff of -1 from v_4 (because of the lim inf definition of $\mathbf{TP}$). Hence, the values of x_3, x_4 and x_5 are respectively 1, -1 and 0. In this game, we have $\mathcal{F}(x_3, x_4, x_5) =$

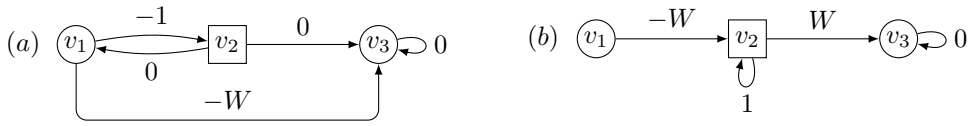


Figure 2 Two weighted graphs.

$\big(2+x_4, \max(-2+x_3, -1+x_5), 1+x_4\big)$, and the vector $(1,-1,0)$ is indeed a fixed point of $\mathcal{F}$. However, it is neither the greatest nor the smallest fixed point of $\mathcal{F}$, since *if* x is a fixed point of $\mathcal{F}$, *then* $x+(a,a,a)$ is also a fixed point, for all constant $a \in \mathbb{Z}$. If we try to initialise the value iteration algorithm with value $(0,0,0)$, which could seem a reasonable choice, the sequence of computed vectors is: $(0,0,0)$, $(2,-1,1)$, $(1,0,0)$, $(2,-1,1)$, $(1,0,0)$, ... that is not stationary, and does not even contain $(1,-1,0)$. Thus, it seems difficult to compute the actual game values with an iterative algorithm relying on the $\mathcal{F}$ operator, as in the case of mean-payoff games.[3] Notice that, in the previous example, the Zwick and Paterson's algorithm [17] to solve mean-payoff games would easily conclude from the sequence above, since the vectors of interest are then the one divided by the length of the current sequence, i.e., $(0,0,0)$, $(1,-0.5,0.5)$, $(0.33,0,0)$, $(0.5,-0.25,0.25)$, $(0.2,0,0)$, ... indeed converging towards $(0,0,0)$, the mean-payoff values of this game.

Instead, as explained in the introduction, we propose a different approach that consists in reducing total-payoff games to min-cost reachability games where Min must enforce a reachability objective on top of his optimisation objective. The aim of the next section is to study these games, and we reduce total-payoff games to them in Section 4.

3 Min-cost reachability games

In this section, we consider *min-cost reachability games* (MCR games for short), a variant of total-payoff games where one player has a reachability objective that he must fulfil first, before optimising his quantitative objective. Without loss of generality, we assign the reachability objective to player Min, as this will make our reduction from total-payoff games easier to explain. Hence, when the target is not reached along a path, its payoff shall be the worst possible for Min, i.e., $+\infty$. Formally, an MCR game is played on a weighted graph $\langle V, E, \omega\rangle$ equipped with a target set of vertices $T \subseteq V$. The payoff T-$\mathbf{MCR}(\pi)$ of a play $\pi = v_0 v_1 \dots$ is given by T-$\mathbf{MCR}(\pi) = +\infty$ if the play avoids T, i.e., if for all $k \geqslant 0$, $v_k \notin T$, and T-$\mathbf{MCR}(\pi) = \mathbf{TP}(\pi[k])$ if k is the least position in π such that $v_k \in T$. Lower and upper values are then defined as in Section 2. By an indirect consequence of Martin's theorem [12], we can show that MCR games are also determined. Optimal strategies may however not exist, as we will see later.

As an example, consider the MCR game played on the weighted graph of Figure 2(*a*), where W is a positive integer and v_3 is the target.

We claim that the values of vertices v_1 and v_2 are both $-W$. Indeed, consider the following strategy for Min: during each of the first W visits to v_2 (if any), go to v_1; else, go to v_3. Clearly, this strategy ensures that the target will eventually be reached, and that either (i) edge (v_1, v_3) (with weight $-W$) will eventually be traversed; or (ii) edge (v_1, v_2) (with weight -1) will be traversed at least W times. Hence, in all plays following this strategy, the

[3] In the context of stochastic models like Markov decision processes, Strauch [14] already noticed that in the presence of arbitrary weights, the value iteration algorithm does not necessarily converge towards the accurate value: see [13, Ex. 7.3.3] for a detailed explanation.

payoff will be at most $-W$. This strategy allows Min to secure $-W$, but he cannot ensure a lower payoff, since Max always has the opportunity to take the edge (v_1, v_3) (with weight $-W$) instead of cycling between v_1 and v_2. Hence, Max's optimal choice is to follow the edge (v_1, v_3) as soon as v_1 is reached, securing a payoff of $-W$. The Min strategy we have just given is optimal, and there is *no optimal memoryless strategy* for Min. Indeed, always playing (v_2, v_3) does not ensure a payoff $\leqslant -W$; and, always playing (v_2, v_1) does not guarantee to reach the target, and this strategy has thus value $+\infty$.

Let us note that Björklund and Vorobyov introduce in [1] the *longest shortest path problem* (LSP for short) and propose a pseudo-polynomial time algorithm to solve it. However, their definition has several subtle but important differences to ours, such as definition of the payoff of a play (equivalently, the length of a path). As an example, in the game of Figure 2(a), the play $\pi = (v_1 v_2)^\omega$ (that never reaches the target) has length $-\infty$ in their setting, while, in our setting, $\{v_3\}$-$\mathbf{MCR}(\pi) = +\infty$. Moreover, even if a pre-treatment would hypothetically allow one to use the LSP algorithm to solve MCR games, our solution is simpler to implement with the same worst-case complexity and heuristics only applicable to our value iteration solution. We now present our contributions for MCR games:

▶ **Theorem 1.** *Let $\mathcal{G} = \langle V, E, \omega, T\text{-}\mathbf{MCR}\rangle$ be an MCR game.*

1. *For $v \in V$, deciding whether $\mathsf{Val}(v) = +\infty$ can be done in polynomial time.*
2. *For $v \in V$, deciding whether $\mathsf{Val}(v) = -\infty$ is as hard as mean-payoff, in* NP ∩ co-NP *and can be achieved in pseudo-polynomial time.*
3. *If $\mathsf{Val}(v) \neq -\infty$ for all vertices $v \in V$, then both players have optimal strategies. Moreover,* Max *always has a memoryless optimal strategy, while* Min *may require finite (pseudo-polynomial) memory in his optimal strategy.*
4. *Computing all values $\mathsf{Val}(v)$ (for $v \in V$), as well as optimal strategies (if they exist) for both players, can be done in (pseudo-polynomial) time $O(|V|^2|E|W)$.*

To prove the first item it suffices to notice that vertices with value $+\infty$ are exactly those from which Min cannot reach the target. Therefore the problem reduces to deciding the winner in a classical reachability game, that can be solved in polynomial time [16], using the classical *attractor* construction: in vertices of value $+\infty$, Min may play indifferently, while Max has an optimal memoryless strategy consisting in avoiding the attractor.

To prove the second item, it suffices first to notice that vertices with value $-\infty$ are exactly those with a value < 0 in the mean-payoff game played on the same graph. On the other hand, we can show that any mean-payoff game can be transformed (in polynomial time) into an MCR game such that a vertex has value < 0 in the mean-payoff game if and only if the value of its corresponding vertex in the MCR game is $-\infty$. The rest of this section focuses on the proof of the third and fourth items. We start by explaining how to compute the values in pseudo-polynomial, and we discuss optimal strategies afterward.

Computing the values. From now on, we assume, without loss of generality, that there is exactly one target vertex denoted by $\mathtt{t}$, and the only outgoing edge from $\mathtt{t}$ is a self loop with weight 0: this is reflected by denoting $\mathbf{MCR}$ the payoff mapping $\{\mathtt{t}\}$-$\mathbf{MCR}$. Our value iteration algorithm for MCR games is given in Algorithm 1. To establish its correctness, we rely mainly on the operator $\mathcal{F}$, which denotes the function $\mathbb{Z}_\infty^V \to \mathbb{Z}_\infty^V$ mapping every vector $x \in \mathbb{Z}_\infty^V$ to $\mathcal{F}(x)$ defined by $\mathcal{F}(x)(\mathtt{t}) = 0$ and

$$\mathcal{F}(x)(v) = \begin{cases} \max\limits_{v' \in E(v)} \big(\omega(v, v') + x(v')\big) & \text{if } v \in V_{\mathsf{Max}} \setminus \{\mathtt{t}\} \\ \min\limits_{v' \in E(v)} \big(\omega(v, v') + x(v')\big) & \text{if } v \in V_{\mathsf{Min}} \setminus \{\mathtt{t}\} \end{cases}$$

Algorithm 1: Value iteration for min-cost reachability games

Input: MCR game $\langle V, E, \omega, \mathbf{MCR}\rangle$, W largest weight in absolute value

1 $\mathsf{X}(\mathtt{t}) := 0$
2 **foreach** $v \in V \setminus \{\mathtt{t}\}$ **do** $\mathsf{X}(v) := +\infty$
3 **repeat**
4 $\quad \mathsf{X}_{pre} := \mathsf{X}$
5 $\quad$ **foreach** $v \in V_{\mathsf{Max}} \setminus \{\mathtt{t}\}$ **do** $\mathsf{X}(v) := \max_{v' \in E(v)} \big(\omega(v,v') + \mathsf{X}_{pre}(v')\big)$
6 $\quad$ **foreach** $v \in V_{\mathsf{Min}} \setminus \{\mathtt{t}\}$ **do** $\mathsf{X}(v) := \min_{v' \in E(v)} \big(\omega(v,v') + \mathsf{X}_{pre}(v')\big)$
7 $\quad$ **foreach** $v \in V \setminus \{\mathtt{t}\}$ such that $\mathsf{X}(v) < -(|V|-1)W$ **do** $\mathsf{X}(v) := -\infty$
8 **until** $\mathsf{X} = \mathsf{X}_{pre}$
9 **return** X

More precisely, we are interested in the sequence of iterates $x_i = \mathcal{F}(x_{i-1})$ of $\mathcal{F}$ from the initial vector x_0 defined by $x_0(v) = +\infty$ for all $v \neq \mathtt{t}$, and $x_0(\mathtt{t}) = 0$. The intuition behind the sequence $(x_i)_{i \geqslant 0}$ is that x_i *is the value of the game if we impose that* Min *must reach the target within* i *steps* (and get a payoff of $+\infty$ if he fails to do so). Formally, for a play $\pi = v_0 v_1 \cdots v_i \cdots$, we let $\mathbf{MCR}^{\leqslant i}(\pi) = \mathbf{MCR}(\pi)$ if $v_k = \mathtt{t}$ for some $k \leqslant i$, and $\mathbf{MCR}^{\leqslant i}(\pi) = +\infty$ otherwise. We further let $\overline{\mathsf{Val}}^{\leqslant i}(v) = \inf_{\sigma_{\mathsf{Min}}} \sup_{\sigma_{\mathsf{Max}}} \mathbf{MCR}^{\leqslant i}(\mathsf{Play}(v, \sigma_{\mathsf{Max}}, \sigma_{\mathsf{Min}}))$ (where σ_{Max} and σ_{Min} are respectively strategies of Max and Min). We can show that the operator $\mathcal{F}$ allows one to compute the sequence $(\overline{\mathsf{Val}}^{\leqslant i})_{i \geqslant 0}$, i.e., for all $i \geqslant 0$: $x_i = \overline{\mathsf{Val}}^{\leqslant i}$.

Let us first show that the algorithm is *correct* when the values of all nodes are *finite*. Thanks to this characterisation, and by definition of $\overline{\mathsf{Val}}^{\leqslant i}$, it is easy to see that, for all $i \geqslant 0$: $x_i = \overline{\mathsf{Val}}^{\leqslant i} \succcurlyeq \overline{\mathsf{Val}} = \mathsf{Val}$. Moreover, $\mathcal{F}$ is a monotonic operator over the complete lattice $\mathbb{Z}_\infty^V$. By Knaster-Tarski's theorem, the fixed points of $\mathcal{F}$ form a complete lattice and $\mathcal{F}$ admits a greatest fixed point. By Kleene's fixed point theorem, using the Scott-continuity of $\mathcal{F}$, this greatest fixed point can be obtained as the limit of the non-increasing sequence of iterates $(\mathcal{F}^i(\overline{x}))_{i \geqslant 0}$ starting in the maximal vector $\overline{x}$ defined by $\overline{x}(v) = +\infty$ for all $v \in V$. As $x_0 = \mathcal{F}(\overline{x})$, the sequence $(x_i)_{i \geqslant 0}$ is also non-increasing (i.e., $x_i \succcurlyeq x_{i+1}$, for all $i \geqslant 0$) and converges towards the greatest fixed point of $\mathcal{F}$. We can further show that the value of the game Val is actually the greatest fixed point of $\mathcal{F}$. Moreover, we can bound the number of steps needed to reach that fixed point (when all values are finite – this is the point where this hypothesis is crucial), by carefully observing the possible vectors that can be computed by the algorithm: the sequence $(x_i)_{i \geqslant 0}$ is non-increasing, and stabilises after at most $(2|V|-1)W|V| + |V|$ steps on Val.

Thus, computing the sequence $(x_i)_{i \geqslant 0}$ up to stabilisation yields the values of all vertices in an MCR game *if all values are finite*. Were it not for line 7, Algorithm 1 would compute exactly this sequence. We claim that Algorithm 1 is correct even when vertices have values in $\{-\infty, +\infty\}$. Line 7 allows to cope with vertices whose value is $-\infty$: when the algorithm detects that Min can secure a value small enough from a vertex v, it sets v's value to $-\infty$. Intuitively, this is correct because if Min can guarantee a payoff smaller than $-(|V|-1) \times W$, he can force a negative cycle from which he can reach $\mathtt{t}$ with an arbitrarily small value. Hence, one can ensure that, after i iterations of the loop, $x_{i-1} \succcurlyeq \mathsf{X} \succcurlyeq \mathsf{Val}$, and the sequence still converges to Val, the greatest fixed point of $\mathcal{F}$. Finally, if some vertex v has value $+\infty$, one can check that $\mathsf{X}(v) = +\infty$ is an invariant of the loop. From that point, one can prove the correctness of the algorithm. Thus, the algorithm executes $O(|V|^2 W)$ iterations. Since each iteration can be performed in $O(|E|)$, the algorithm has a complexity of $O(|V|^2|E|W)$,

as announced in Theorem 1. As an example, consider the min-cost reachability game of Figure 2(a). The successive values for vertices (v_1, v_2) (value of the target v_3 is always 0) computed by the value iteration algorithm are the following: $(+\infty, +\infty)$, $(+\infty, 0)$, $(-1, 0)$, $(-1, -1)$, $(-2, -1)$, $(-2, -2), \ldots, (-W, -W+1)$, $(-W, -W)$. This requires $2W$ steps to converge (hence a pseudo-polynomial time).

Computing optimal strategies for both players. We now turn to the proof of the third item of Theorem 1, supposing that every vertex v of the game has a finite value $\mathsf{Val}(v) \in \mathbb{Z}$ (the case where $\mathsf{Val}(v) = +\infty$ is delt with the attractor construction).

Observe first that, Min may *need memory* to play optimally, as already shown by the example in Figure 2(a), where the target is v_3. Nevertheless, let us briefly explain why optimal strategies for Min always exist, with a memory of pseudo-polynomial size. We extract from the sequence $(x_i)_{i \geqslant 0}$ defined above (or equivalently, from the sequence of vectors X of Algorithm 1) the optimal strategy σ^*_{Min} as follows. Let k be the first index such that $x_{k+1} = x_k$. Then, for every play π ending in vertex $v \in V_{\mathsf{Min}}$, we let $\sigma^*_{\mathsf{Min}}(\pi) = \operatorname{argmin}_{v' \in E(v)} \big(\omega(v, v') + x_{k-|\pi|-1}(v')\big)$, if $|\pi| < k$, and $\sigma^*_{\mathsf{Min}}(\pi) = \operatorname{argmin}_{v' \in E(v)} \big(\omega(v, v') + x_0(v')\big)$ otherwise (those argmin may not be unique, but we can indifferently pick any of them). Since σ_{Min} only requires to know the last vertex and the length of the prefix up to k, and since $k \leqslant (2|V|-1)W|V|+|V|$ as explained above, σ^*_{Min} needs a memory of pseudo-polynomial size only. Moreover, it can be computed with the sequence of vectors X in Algorithm 1. It is not difficult to verify by induction that this strategy is optimal for Min. While optimal, this strategy might not be practical, for instance, in the framework of controller synthesis. Implementing it would require to store the full sequence $(x_i)_{i \geqslant 0}$ up to convergence step k (possibly pseudo-polynomial) in a table, and to query this large table each time the strategy is called. Instead, an alternative optimal strategy σ'_{Min} can be construct, that consists in playing successively two *memoryless strategies* σ^1_{Min} and σ^2_{Min} (σ^2_{Min} being given by the attractor construction). To determine when to switch from σ^1_{Min} to σ^2_{Min}, σ'_{Min} maintains a counter that is stored in a *polynomial* number of bits, thus the memory footprints of σ'_{Min} and σ^*_{Min} are comparable. However, σ'_{Min} is easier to implement, because σ^1_{Min} and σ^2_{Min} can be described by a pair of tables of linear size, and, apart from querying those tables, σ'_{Min} consists only in incrementing and testing the counter to determine when to switch. Moreover, this succession of two memoryless strategies allows us to also get some interesting strategy in case of vertices with values $-\infty$: indeed, we can still compute this pair of strategies, and simply modify the switching policy to run for a sufficiently long time to guarantee a value less than a given threshold. In the following, we call such a strategy a *switching strategy*.

Finally, we can show that, contrary to Min, Max always has a *memoryless optimal strategy* σ^*_{Max} defined by $\sigma^*_{\mathsf{Max}}(\pi) = \operatorname{argmax}_{v' \in E(v)} (\omega(v, v') + \mathsf{Val}(v'))$ for all finite plays π ending in $v \in V_{\mathsf{Max}}$. For example, in the game of Figure 2(a), $\sigma^*_{\mathsf{Max}}(\pi v_2) = v_3$ for all π, since $\mathsf{Val}(v_3) = 0$ and $\mathsf{Val}(v_1) = -W$. Moreover, the previously described optimal strategies can be computed along the execution of Algorithm 1. Finally, we can show that, for all vertices v, the pair of optimal strategies we have just defined yields a play $\mathsf{Play}(v, \sigma^*_{\mathsf{Max}}, \sigma^*_{\mathsf{Min}})$ which is *non-looping*, i.e., never visits the same vertex twice before reaching the target. For instance, still in the game of Figure 2(a), $\mathsf{Play}(v_1, \sigma^*_{\mathsf{Max}}, \sigma^*_{\mathsf{Min}}) = v_1 v_2 v_3^\omega$.

4 An efficient algorithm to solve total-payoff games

We now turn our attention back to total-payoff games (without reachability objective), and discuss our main contribution. Building on the results of the previous section, we introduce

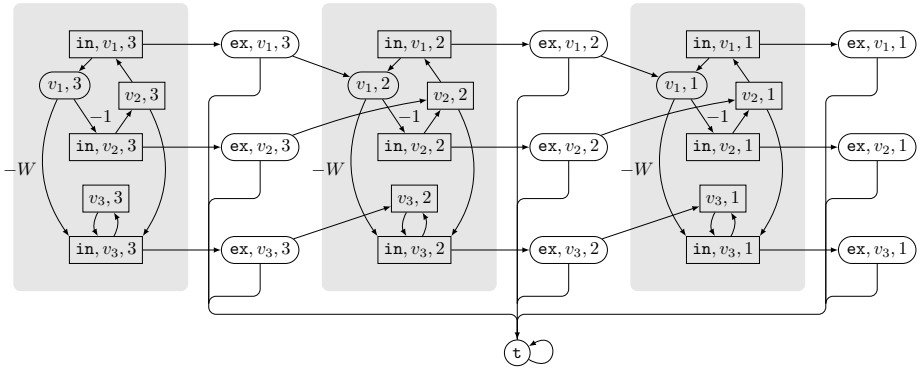


Figure 3 MCR game $\mathcal{G}^3$ associated with the total-payoff game of Figure 2(a).

the *first* (as far as we know) *pseudo-polynomial time algorithm* for solving those games in the presence of arbitrary weights, thanks to a reduction from total-payoff games to min-cost reachability games. The MCR game produced by the reduction has size pseudo-polynomial in the size of the original total-payoff game. Then, we show how to compute the values of the total-payoff game without building the entire MCR game, and explain how to deduce memoryless optimal strategies from the computation of our algorithm.

Reduction to min-cost reachability games. We provide a transformation from a total-payoff game $\mathcal{G} = \langle V, E, \omega, \mathbf{TP}\rangle$ to a min-cost reachability game $\mathcal{G}^K$ such that the values of $\mathcal{G}$ can be extracted from the values in $\mathcal{G}^K$ (as formalised below). Intuitively, $\mathcal{G}^K$ simulates the game where players play in $\mathcal{G}$; Min may propose to stop playing and reach a fresh vertex $\mathtt{t}$ acting as the target; Max can then accept, in which case we reach the target, or refuse at most K times, in which case the game continues. Structurally, $\mathcal{G}^K$ consists of a sequence of copies of $\mathcal{G}$ along with some new states that we now describe formally. We let $\mathtt{t}$ be a fresh vertex, and, for all $n \geqslant 1$, we define the min-cost reachability game $\mathcal{G}^n = \langle V^n, E^n, \omega^n, \{\mathtt{t}\}\text{-}\mathbf{MCR}\rangle$ where V^n_{Max} (respectively, V^n_{Min}) consists of n copies (v, j), with $1 \leqslant j \leqslant n$, of each vertex $v \in V_{\mathsf{Max}}$ (respectively, $v \in V_{\mathsf{Min}}$) and some *exterior vertices* $(\mathtt{ex}, v, j)$ for all $v \in V$ and $1 \leqslant j \leqslant n$ (respectively, *interior vertices* $(\mathtt{in}, v, j)$ for all $v \in V$ and $1 \leqslant j \leqslant n$). Moreover, V^n_{Max} contains the fresh target vertex $\mathtt{t}$. Edges are given by

$$\begin{aligned} E^n = {} & \{(\mathtt{t},\mathtt{t})\} \uplus \big\{\big((v,j),(\mathtt{in},v',j)\big) \mid (v,v') \in E, 1 \leqslant j \leqslant n\big\} \\ & \uplus \big\{\big((\mathtt{in},v,j),(v,j)\big) \mid v \in V, 1 \leqslant j \leqslant n\big\} \uplus \big\{\big((\mathtt{ex},v,j),\mathtt{t}\big) \mid v \in V, 1 \leqslant j \leqslant n\big\} \\ & \uplus \big\{\big((\mathtt{in},v,j),(\mathtt{ex},v,j)\big) \mid v \in V, 1 \leqslant j \leqslant n\big\} \\ & \uplus \big\{\big((\mathtt{ex},v,j),(v,j-1)\big) \mid v \in V, 1 < j \leqslant n\big\}. \end{aligned}$$

All edge weights are zero, except edges $\big((v,j),(\mathtt{in},v',j)\big)$ that have weight $\omega(v,v')$.

For example, considering the weighted graph of Figure 2(a), the corresponding reachability total-payoff game $\mathcal{G}^3$ is depicted in Figure 3 (where weights 0 have been removed). The next proposition formalises the relationship between the two games.

▶ **Proposition 2.** *Let $K = |V|(2(|V|-1)W+1)$. For all $v \in V$ and $k \geqslant K$,*

- $\mathsf{Val}_{\mathcal{G}}(v) \neq +\infty$ *if and only if* $\mathsf{Val}_{\mathcal{G}}(v) = \mathsf{Val}_{\mathcal{G}^k}((v,k))$*;*
- $\mathsf{Val}_{\mathcal{G}}(v) = +\infty$ *if and only if* $\mathsf{Val}_{\mathcal{G}^k}((v,k)) \geqslant (|V|-1)W+1$.

The bound K is found by using the fact (informally described in the previous section) that if not infinite, the value of a min-cost reachability game belongs in $[-(|V|-1)\times W+1, |V|\times W]$,

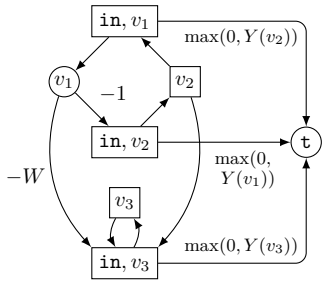


Figure 4 MCR game $\mathcal{G}_Y$ associated with the total-payoff game of Figure 2(a).

and that after enough visits of the same vertex, an adequate loop ensures that $\mathcal{G}^k$ verifies the above properties.

Value iteration algorithm for total-payoff games. By Proposition 2, an immediate way to obtain a value iteration algorithm for total-payoff games is to build game $\mathcal{G}^K$, run Algorithm 1 on it, and map the computed values back to $\mathcal{G}$. We take advantage of the structure of $\mathcal{G}^K$ to provide a better algorithm that avoids building $\mathcal{G}^K$. We first compute the values of the vertices in the *last copy of the game* (vertices of the form $(v, 1)$, $(\texttt{in}, v, 1)$ and $(\texttt{ex}, v, 1)$), then of those in the penultimate (vertices of the form $(v, 2)$, $(\texttt{in}, v, 2)$ and $(\texttt{ex}, v, 2)$), and so on.

We formalise this idea as follows. Let Z^j be a vector mapping each vertex v of $\mathcal{G}$ to the value $Z^j(v)$ of vertex (v, j) in $\mathcal{G}^K$. Then, let us define an operator $\mathcal{H}$ such that $Z^{j+1} = \mathcal{H}(Z^j)$. The intuition behind the definition of $\mathcal{H}(Y)$ for some vector Y, is to extract from $\mathcal{G}^K$ one copy of the game, and make Y appear in the weights of some edges as illustrated in Figure 4. This game, $\mathcal{G}_Y$, simulates a play in $\mathcal{G}$ in which Min can opt for 'leaving the game' at each round (by moving to the target), obtaining $\max(0, Y(v))$, if v is the current vertex. Then $\mathcal{H}(Y)(v)$ is defined as the value of v in $\mathcal{G}_Y$. By construction, it is easy to see that $Z^{j+1} = \mathcal{H}(Z^j)$ holds for all $j \geqslant 1$. Furthermore, we define $Z^0(v) = -\infty$ for all v, and have $Z^1 = \mathcal{H}(Z^0)$. One can prove the following properties of $\mathcal{H}$: (i) $\mathcal{H}$ is monotonic, but may not be Scott-continuous; (ii) the sequence $(Z^j)_{j \geqslant 0}$ converges towards $\mathsf{Val}_{\mathcal{G}}$.

We are now ready to introduce Algorithm 2 to solve total-payoff games. Intuitively, the outer loop computes, in variable Y, a non-decreasing sequence of vectors whose limit is $\mathsf{Val}_{\mathcal{G}}$, and that is stationary (this is not necessarily the case for the sequence $(Z^j)_{j \geqslant 0}$). Line 1 initialises Y to Z^0. Each iteration of the outer loop amounts to running Algorithm 1 to compute $\mathcal{H}(\mathsf{Y}_{pre})$ (lines 3 to 10), then detecting if some vertices have value $+\infty$, updating Y accordingly (line 11, following the second item of Proposition 2). One can show that, for all $j > 0$, if we let Y^j be the value of Y after the j-th iteration of the main loop, $Z^j \preccurlyeq Y^j \preccurlyeq \mathsf{Val}_{\mathcal{G}}$, which ensures the correctness of the algorithm.

▶ **Theorem 3.** *If a total-payoff game $\mathcal{G} = \langle V, E, \omega, \mathbf{TP} \rangle$ is given as input, Algorithm 2 outputs the vector $\mathsf{Val}_{\mathcal{G}}$ of optimal values, after at most $K = |V|(2(|V| - 1)W + 1)$ iterations of the external loop. The complexity of the algorithm is $O(|V|^4|E|W^2)$.*

The number of iterations in each internal loop is controlled by Theorem 1. On the example of Figure 2(a), only 2 external iterations are necessary, but the number of iterations of each internal loop would be $2W$. By contrast, for the total-payoff game depicted in Figure 2(b), each internal loop requires 2 iterations to converge, but the external loop takes W iterations to stabilise. A combination of both examples would experience a pseudo-polynomial number of iterations to converge in both the internal and external loops, matching the W^2 term of the above complexity.

Algorithm 2: A value iteration algorithm for total-payoff games.

Input: Total-payoff game $\mathcal{G} = \langle V, E, \omega, \mathbf{TP}\rangle$, W largest weight in absolute value

foreach $v \in V$ **do** $\mathsf{Y}(v) := -\infty$
repeat
  foreach $v \in V$ **do** $\mathsf{Y}_{pre}(v) := \mathsf{Y}(v)$; $\mathsf{Y}(v) := \max(0, \mathsf{Y}(v))$; $\mathsf{X}(v) := +\infty$
  repeat
    $\mathsf{X}_{pre} := \mathsf{X}$
    foreach $v \in V_{\mathsf{Max}}$ **do** $\mathsf{X}(v) := \max_{v' \in E(v)} \left[\omega(v, v') + \min(\mathsf{X}_{pre}(v'), \mathsf{Y}(v'))\right]$
    foreach $v \in V_{\mathsf{Min}}$ **do** $\mathsf{X}(v) := \min_{v' \in E(v)} \left[\omega(v, v') + \min(\mathsf{X}_{pre}(v'), \mathsf{Y}(v'))\right]$
    foreach $v \in V$ such that $\mathsf{X}(v) < -(|V| - 1)W$ **do** $\mathsf{X}(v) := -\infty$
  until $\mathsf{X} = \mathsf{X}_{pre}$
  $\mathsf{Y} := \mathsf{X}$
  foreach $v \in V$ such that $\mathsf{Y}(v) > (|V| - 1)W$ **do** $\mathsf{Y}(v) := +\infty$
until $\mathsf{Y} = \mathsf{Y}_{pre}$
return Y

Optimal strategies. In Section 3, we have shown, for any min-cost reachability game, the existence of a pair of memoryless strategies permitting to reconstruct a *switching* optimal strategy for Min (if every vertex has value different from $-\infty$, or a strategy ensuring any possible threshold for vertices with value $-\infty$). If we apply this construction to the game $\mathcal{G}_{\mathsf{Val}_\mathcal{G}}$, we obtain a pair $(\sigma^1_{\mathsf{Min}}, \sigma^2_{\mathsf{Min}})$ of strategies (remember that σ^2_{Min} is a strategy obtained by the attractor construction, hence it will not be useful for us for total-payoff games). Consider the strategy $\overline{\sigma_{\mathsf{Min}}}$, obtained by projecting σ^1_{Min} on V as follows: for all finite plays π and vertex $v \in V_{\mathsf{Min}}$, let $\overline{\sigma_{\mathsf{Min}}}(\pi v) = v'$ if $\sigma^1_{\mathsf{Min}}(v) = (\mathtt{in}, v')$. We can show that $\overline{\sigma_{\mathsf{Min}}}$ is optimal for Min in $\mathcal{G}$. Notice that σ^1_{Min}, and hence $\overline{\sigma_{\mathsf{Min}}}$, can be computed during the last iteration of the value iteration algorithm, as explained in the case of min-cost reachability. A similar construction can be done to compute an optimal strategy of Max.

5 Implementation and heuristics

In this section, we report on a prototype implementation of our algorithms.[4] For convenience reasons, we have implemented them as an add-on to PRISM-games [5], although we could have chosen to extend another model-checker as we do not rely on the probabilistic features of PRISM models (i.e., we use the PRISM syntax of *stochastic multi-player games*, allowing arbitrary rewards, and forbidding probability distributions different of Dirac ones). We then use rPATL specifications of the form $\langle\langle C\rangle\rangle \mathsf{R}^{\min\,/\,\max=?}[\mathsf{F}^\infty \varphi]$ and $\langle\langle C\rangle\rangle \mathsf{R}^{\min\,/\,\max=?}[\mathsf{F}^c \bot]$ to model respectively min-cost reachability games and total-payoff games, where C represents a coalition of players that want to minimise/maximise the payoff, and φ is another rPATL formula describing the target set of vertices (for total-payoff games, such a formula is not necessary). We have tested our implementation on toy examples. On the parametric one studied after Theorem 3, obtained by mixing the graphs of Figure 2 and repeating them for n layers, results obtained by applying our algorithm for total-payoff games are summarised in Table 1, where for each pair (W, n), we give the time t in seconds, the number k_e of iterations in the external loop, and the total number k_i of iterations in the internal loop.

[4] Source and binary files, as well as some examples, can be downloaded from `http://www.ulb.ac.be/di/verif/monmege/tool/TP-MCR/`.

Table 1 Results of value iteration on a parametric example.

		without heuristics			with heuristics		
W	n	t	k_e	k_i	t	k_e	k_i
50	100	0.52s	151	12,603	0.01s	402	1,404
50	500	9.83s	551	53,003	0.42s	2,002	7,004
200	100	2.96s	301	80,103	0.02s	402	1,404
200	500	45.64s	701	240,503	0.47s	2,002	7,004
500	1,000	536s	1,501	1,251,003	2.37s	4,002	14,004

We close this section by sketching two techniques that can be used to speed up the computation of the fixed point in Algorithms 1 and 2. We fix a weighted graph $\langle V, E, \omega\rangle$. Both accelerations rely on a topological order of the strongly connected components (SCC for short) of the graph, given as a function $\mathsf{c}\colon V \to \mathbb{N}$, mapping each vertex to its *component*, verifying that (i) $\mathsf{c}(V) = \{0, \dots, p\}$ for some $p \geqslant 0$, (ii) $\mathsf{c}^{-1}(q)$ is a maximal SCC for all q, (iii) and $\mathsf{c}(v) \geqslant \mathsf{c}(v')$ for all $(v, v') \in E$.[5] In case of an MRC game with $\mathtt{t}$ the unique target, $\mathsf{c}^{-1}(0) = \{\mathtt{t}\}$. Intuitively, c induces a directed acyclic graph whose vertices are the sets $\mathsf{c}^{-1}(q)$ for all $q \in \mathsf{c}(V)$, and with an edge (S_1, S_2) if and only if there are $v_1 \in S_1, v_2 \in S_2$ such that $(v_1, v_2) \in E$.

The *first acceleration heuristic* is a divide-and-conquer technique that consists in applying Algorithm 1 (or the inner loop of Algorithm 2) iteratively on each $\mathsf{c}^{-1}(q)$ for $q = 0, 1, 2, \dots, p$, using at each step the information computed during steps $j < q$ (since the value of a vertex v depends only on the values of the vertices v' such that $\mathsf{c}(v') \leqslant \mathsf{c}(v)$). The *second acceleration heuristic* consists in studying more precisely each component $\mathsf{c}^{-1}(q)$. Having already computed the optimal values $\mathsf{Val}(v)$ of vertices $v \in \mathsf{c}^{-1}(\{0, \dots, q-1\})$, we ask an oracle to precompute a finite set $S_v \subseteq \mathbb{Z}_\infty$ of possible optimal values for each vertex $v \in \mathsf{c}^{-1}(q)$. For MCR games and the inner iteration of the algorithm for total-payoff games, one way to construct such a set S_v is to consider that possible optimal values are the one of non-looping paths inside the component exiting it, since, in MCR games, there exist optimal strategies for both players whose outcome is a non-looping path (see Section 3).

We can identify classes of weighted graphs for which there exists an oracle that runs in polynomial time and returns, for all vertices v, a set S_v of polynomial size. On such classes, Algorithms 1 and 2, enhanced with our two acceleration techniques, *run in polynomial time*. For instance, for all fixed positive integers L, the class of weighted graphs where every component $\mathsf{c}^{-1}(q)$ uses at most L distinct weights (that can be arbitrarily large in absolute value) satisfies this criterion. Table 1 contains the results obtained with the heuristics on the parametric example presented before. Observe that the acceleration technique permits here to decrease drastically the execution time, the number of iterations in both loops depending not even anymore on W. Even though the number of iterations in the external loop increases with heuristics, due to the decomposition, less computation is required in each internal loop since we only apply the computation for the active component.

References

1 Henrik Björklund and Sergei Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Mathematics*, 155:210–229, 2007.

[5] Such a mapping is computable in linear time, e.g., by Tarjan's algorithm [15].

2 Thomas Brihaye, Gilles Geeraerts, Axel Haddad, and Benjamin Monmege. To reach or not to reach? Efficient algorithms for total-payoff games. Research Report 1407.5030, arXiv, July 2014.

3 Thomas Brihaye, Gilles Geeraerts, Shankara Narayanan Krishna, Lakshmi Manasa, Benjamin Monmege, and Ashutosh Trivedi. Adding negative prices to priced timed games. In *Proceedings of the 25th International Conference on Concurrency Theory (CONCUR'14)*, volume 8704 of *Lecture Notes in Computer Science*, pages 560–575. Springer, 2014.

4 Luboš Brim, Jakub Chaloupka, Laurent Doyen, Rafaella Gentilini, and Jean-François Raskin. Faster algorithms for mean-payoff games. *Formal Methods for System Design*, 38(2):97–118, 2011.

5 Taolue Chen, Vojtěch Forejt, Marta Kwiatkowska, David Parker, and Aistis Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.

6 Carlo Comin and Romeo Rizzi. An improved pseudo-polynomial upper bound for the value problem and optimal strategy synthesis in mean payoff games. Technical Report 1503.04426, arXiv, 2015.

7 Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979.

8 Emmanuel Filiot, Rafaella Gentilini, and Jean-François Raskin. Quantitative languages defined by functional automata. In *Proceedings of the 23rd International Conference on Concurrency theory (CONCUR'12)*, volume 7454 of *Lecture Notes in Computer Science*, pages 132–146. Springer, 2012.

9 Thomas Martin Gawlitza and Helmut Seidl. Games through nested fixpoints. In *Proceedings of the 21st International Conference on Computer Aided Verification (CAV'09)*, volume 5643 of *Lecture Notes in Computer Science*, pages 291–305. Springer, 2009.

10 Hugo Gimbert and Wiesław Zielonka. When can you play positionally? In *Proceedings of the 29th International Conference on Mathematical Foundations of Computer Science (MFCS'04)*, volume 3153 of *Lecture Notes in Computer Science*, pages 686–698. Springer, 2004.

11 Leonid Khachiyan, Endre Boros, Konrad Borys, Khaled Elbassioni, Vladimir Gurvich, Gabor Rudolf, and Jihui Zhao. On short paths interdiction problems: Total and node-wise limited interdiction. *Theory of Computing Systems*, 43:204–233, 2008.

12 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102(2):363–371, 1975.

13 Martin L. Puterman. *Markov Decision Processes.* John Wiley & Sons, Inc., New York, NY, 1994.

14 Ralph E. Strauch. Negative dynamic programming. *The Annals of Mathematical Statistics*, 37:871–890, 1966.

15 Robert E. Tarjan. Depth first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972.

16 Wolfgang Thomas. On the synthesis of strategies in infinite games. In *Symposium on Theoretical Aspects of Computer Science (STACS'95)*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1995.

17 Uri Zwick and Michael S. Paterson. The complexity of mean payoff games. *Theoretical Computer Science*, 158:343–359, 1996.

On the Value Problem in Weighted Timed Games*

Patricia Bouyer, Samy Jaziri, and Nicolas Markey

LSV – CNRS & ENS Cachan, France

Abstract

A weighted timed game is a timed game with extra quantitative information representing e.g. energy consumption. Optimizing the weight for reaching a target is a natural question, which has already been investigated for ten years. *Existence* of optimal strategies is known to be undecidable in general, and only very restricted classes of games have been identified for which optimal weight and almost-optimal strategies can be computed. In this paper, we show that the *value problem* is undecidable in weighted timed games. We then introduce a large subclass of weighted timed games (for which the undecidability proof above applies), and provide an algorithm to compute arbitrary approximations of the value in such games. To the best of our knowledge, this is the first approximation scheme for an undecidable class of weighted timed games.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Timed games, undecidability, approximation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.311

1 Introduction

Timed automata [3] have been introduced in the early 1990's as a powerful model to reason about (the correctness of) real-time computerized systems. Timed automata extend finite-state automata with several clocks, which can be used to enforce timing constraints between various events in the system. They provide a convenient formalism and enjoy reasonably-efficient algorithms (e.g. reachability can be decided using polynomial space), which explains the enormous interest that they provoked in the community of formal methods. *Timed games* [5] extend timed automata with a way of modeling systems interacting with external, uncontrollable components: some transitions of the automaton cannot be forced or prevented to happen. The reachability problem then asks whether there is a *strategy* to reach a given state, whatever the uncontrollable components do. This problem is also decidable, in exponential time.

Hybrid automata [2] are another extension of timed automata, involving *hybrid variables*: those variables can be used to measure other quantities than time (e.g. temperature, energy consumption, ...). Their evolution may follow differential equations, depending on the state of the system. Those variables unfortunately make the reachability problem undecidable [18], even in the restricted case of stopwatches, which are clocks that can be stopped and restarted. Weighted (or priced) timed automata [4, 6] and games [19, 1, 11] have been proposed in the early 2000's as an intermediary model for modelling resource consumption or allocation problems in real-time systems (*e.g.* optimal scheduling [7]). As opposed to (linear) hybrid systems, an execution in a weighted timed model is simply one in the underlying timed model: the extra quantitative information is just an *observer* of the system, and it does not

* This work has been partially supported by the EU under ERC project EQualIS (FP7-308087) and FET project Cassting (FP7-601148).

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 311–324

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

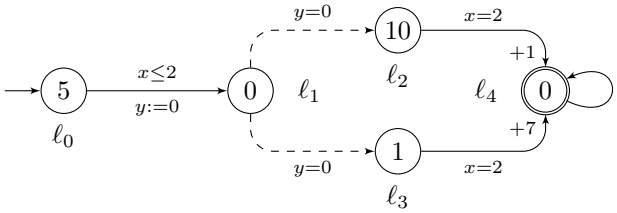


Figure 1 A two-clock weighted timed game.

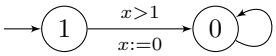


Figure 2 A weighted timed game with value 1 where Player 1 has no optimal strategy.

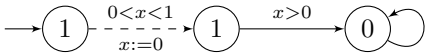


Figure 3 A weighted timed game with value 1 where Player 1 has a strategy to secure weight strictly less than 1.

modify the possible behaviors of the system. Figure 1 displays an example of a weighted timed game: each location carries an integer, which is the rate by which the weight increases when time elapses in that location. Some edges also carry a weight, which indicates how much the weight increases when crossing this edge. Dashed edges are *uncontrollable*, and cannot be forced or prevented to occur. Notice that the constraints on edges never depend on the value of the weight, but only on the values of the clocks.

While optimal weight and almost-optimal schedules can be computed in weighted timed automata [4, 6, 8], the situation is less appealing in the context of weighted timed games: indeed, it is in general undecidable whether a player has a strategy to reach a target with total weight no more than a given value [14], even for timed games with only three clocks [9]. Optimal weight and almost-optimal winning strategies can be computed in restricted classes of weighted timed games, such as games with strong divergence properties on the weight [1, 11], or one-clock turn-based games [13, 22, 17, 15].

We point out a discrepancy in the set of results mentioned above: decidability results concern the *value problem* (is the infimum, over all strategies of Player 1, of the accumulated weight, less than or equal to some constant?), whereas undecidability results deal with the *existence problem* (is there a strategy for Player 1 under which the accumulated weight is less than or equal to some constant?). Both problems are obviously related, but the undecidability of the existence problem does not entail the undecidability of the value problem: indeed, there are obvious examples (see Fig. 2) – and more complex ones (see [12]) – where Player 1 has no optimal strategy for securing the exact value of the game. More surprisingly, there also exist games where Player 1 has *super-optimal* strategies, which when combined with any strategy of Player 2, achieves final weight strictly better than the value of the game. The value of the game of Fig. 3 is 1, but if Player 1 plays a delay $\epsilon/2$ when Player 2 (controlling the dashed edge) played a delay of $1 - \epsilon$ in the first location, she gets final weight strictly less than 1 against any Player-2 strategy.

Our contributions in this paper are the following:

- We show that the value problem is undecidable: given a game and a rational c, no algorithm can decide whether the value of the game is less than or equal to c. The proof of this result shares similarities with the undecidability proof for the existence problem, but requires a more careful analysis of the strategies of the players.
- We exhibit a subclass of timed games for which arbitrary precise approximations of the value of a game can be computed. This subclass is large enough to include the

games that are used in the undecidability proof mentioned above. We believe that this approximability result is an important result, since getting the exact value is rarely needed in practice, and optimal strategies need not exist anyway. We notice that in all cases we know of where the optimal weight can be computed (namely [4, 6, 8, 1, 11, 13, 22, 17, 15]), only almost-optimal strategies are actually synthesized; hence it is not really meaningful to know the precise value, and an approximation thereof is sufficient. As a side-result, we get that the optimal weight is co-recursively enumerable in this subclass.

For lack of space, detailed proofs are deferred to the research report [12].

2 Definitions

2.1 Weighted timed games

Let X be a finite set of variables (called *clocks* in our context). A clock valuation for X is a mapping $X \to \mathbb{R}_{\geq 0}$. Given such a clock valuation v, $d \in \mathbb{R}_{\geq 0}$ and $Y \subseteq X$, we define the valuations $v + d$ and $v[Y \leftarrow 0]$ respectively by $(v+d)(x) = v(x) + d$ for every $x \in X$, and by $(v[Y \leftarrow 0])(x) = 0$ if $x \in Y$ and $(v[Y \leftarrow 0])(x) = v(x)$ otherwise.

A clock constraint over a finite set of clocks X is a conjunction of atomic constraints of the form $x \bowtie c$, where $\bowtie \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$. We say that a valuation $v\colon X \mapsto \mathbb{R}_{\geq 0}$ satisfies a constraint $x \bowtie c$ whenever $v(x) \bowtie c$; the semantics of conjunction is natural. We also allow $\top$ and $\bot$ as trivial clock constraints (which always evaluate to true and false, respectively).

▶ **Definition 1.** A *weighted timed game* is a tuple $\mathcal{G} = \langle L, \ell_0, L_{\mathsf{f}}, X, E, E_1, E_2, \mathsf{wt}\rangle$ where L is a finite set of locations; $\ell_0 \in L$ is the initial location; $L_{\mathsf{f}} \subseteq L$ is the subset of target locations; X is a finite set of clocks; $E \subseteq L \times \mathcal{C}(X) \times 2^X \times L$ is a finite set of edges, partitioned into the edges E_1 of Player 1, and the edges E_2 of Player 2; and $\mathsf{wt}\colon L \cup E \to \mathbb{Q}_{\geq 0}$ assigns a value to every location and to every edge. We assume that for any $\ell \in L$ and any $v \in \mathbb{R}^X_{\geq 0}$, there is a transition $(\ell, g, r, \ell') \in E$ such that $v \models g$.

The game is *turn-based* whenever L can be partitioned into L_1 and L_2 such that all the edges in E_1 (resp. E_2) have their sources in L_1 (resp. L_2).

The semantics of such a weighted timed game is defined as a game on an infinite graph: a configuration of $\mathcal{G}$ is a pair (ℓ, v) consisting of a location and a clock valuation of X. The configuration $(\ell_0, \mathbf{0})$, where valuation $\mathbf{0}$ assigns 0 to every clock, is the initial configuration. A target configuration is a configuration (ℓ, v) where $\ell \in L_{\mathsf{f}}$ and v is any clock valuation. For any $e = (\ell, g, Y, \ell') \in E$ and $d \geq 0$, there is a transition $(\ell, v) \xrightarrow{d,e} (\ell', v')$ whenever $v + d \models g$ and $v' = v[Y \leftarrow 0]$. In that case, we say that action (d, e) is enabled at (ℓ, v). The weight of such a transition is $\mathsf{wt}_{\mathcal{G}}\big((\ell, v) \xrightarrow{d,e} (\ell', v')\big) = d \cdot \mathsf{wt}(\ell) + \mathsf{wt}(e)$.

A *run* (or *path*) in $\mathcal{G}$ is a finite or infinite sequence of consecutive transitions. A run is initial whenever it starts in $(\ell_0, \mathbf{0})$. If ρ is finite, we write $\mathit{last}(\rho)$ for its last configuration. We write $\mathsf{FPath}^{\mathcal{G}}(\ell, v)$ for the set of finite runs in $\mathcal{G}$ starting in configuration (ℓ, v). We write $\mathsf{FPath}^{\mathcal{G}}$ for the set of all finite runs in $\mathcal{G}$. Given a run ρ, we write $\rho_{|\mathsf{f}}$ for the prefix of ρ ending in the first target configuration it reaches (or $\rho_{|\mathsf{f}} = \rho$ if no target location is visited along ρ). If ρ is a finite run, its weight, written $\mathsf{wt}_{\mathcal{G}}(\rho)$, is defined as the sum of the weights of all its intermediary transitions. If ρ is an infinite run, its weight is $+\infty$ if it does not visit a target configuration, and it is $\mathsf{wt}(\rho_{|\mathsf{f}})$ otherwise.

We now explain how the game is played between Player 1 and Player 2. Let $p \in \{1, 2\}$. A *strategy* for Player p is a mapping $\sigma_p\colon \mathsf{FPath}^{\mathcal{G}} \to (\mathbb{R}_+ \times E_p) \cup \{\bot\}$ such that for every run

$\rho \in \mathsf{FPath}^{\mathcal{G}}$, the action $\sigma_p(\rho)$ is enabled at $last(\rho)$. The special value $\bot$ is used in case no action is enabled for Player p (and only in this case). We write $\mathsf{Strat}_p^{\mathcal{G}}$ for the set of strategies of Player p in $\mathcal{G}$. The (unique) *outcome* of a pair of strategies $(\sigma_1, \sigma_2) \in \mathsf{Strat}_1^{\mathcal{G}} \times \mathsf{Strat}_2^{\mathcal{G}}$ from some configuration s, denoted $\mathsf{Out}_{\mathcal{G}}((\sigma_1, \sigma_2), s)$, is the unique infinite run $\rho = s_0 \xrightarrow{d_1,e_1} s_1 \xrightarrow{d_2,e_2} \dots$ such that $s_0 = s$ and for every $n \in \mathbb{N}$, writing $\rho_{\leq n}$ for $s_0 \xrightarrow{d_1,e_1} s_1 \xrightarrow{d_2,e_2} \dots s_n$, it holds:

- if for some $p \in \{1,2\}$, $\sigma_p(\rho_{\leq n}) = \bot$, then necessarily $\sigma_{3-p}(\rho_{\leq n}) = (d,e)$ (it cannot be $\bot$, according to our definitions), and $(d_{n+1}, e_{n+1}) = (d,e)$;
- if $\sigma_1(\rho_{\leq n}) = (d^1, e^1)$ and $\sigma_2(\rho_{\leq n}) = (d^2, e^2)$, then $d_{n+1} = \min(d^1, d^2)$. Then the action of the player with the smallest delay is selected: if $d^p < d^{3-p}$ with $p \in \{1,2\}$, then $e_{n+1} = e^p$, whereas if $d^1 = d^2$ then $e_{n+1} = e^2$. This last condition expresses a priority given to Player 2 (this choice is arbitrary; other variants, e.g. with non-determinism [16], could be handled similarly).

Given two strategies $(\sigma_1, \sigma_2) \in \mathsf{Strat}_1^{\mathcal{G}} \times \mathsf{Strat}_2^{\mathcal{G}}$, their joint weight from configuration s is defined as the weight of their outcome from s. We write $\mathsf{wt}_{\mathcal{G}}((\sigma_1, \sigma_2), s) = \mathsf{wt}(\mathsf{Out}_{\mathcal{G}}((\sigma_1, \sigma_2), s)$. If σ_1 (resp. σ_2) is a Player-1 (resp. Player-2) strategy, we define its weight from s as:

$$\mathsf{wt}_{\mathcal{G}}(\sigma_1, s) = \sup_{\sigma_2' \in \mathsf{Strat}_2^{\mathcal{G}}} \mathsf{wt}_{\mathcal{G}}((\sigma_1, \sigma_2'), s) \qquad \mathsf{wt}_{\mathcal{G}}(\sigma_2, s) = \inf_{\sigma_1' \in \mathsf{Strat}_1^{\mathcal{G}}} \mathsf{wt}_{\mathcal{G}}((\sigma_1', \sigma_2), s)$$

We then define the optimal weight for Player 1 (resp. Player 2) in s as follows:

$$\mathsf{optwt}_{\mathcal{G}}^1(s) = \inf_{\sigma_1 \in \mathsf{Strat}_1^{\mathcal{G}}} \mathsf{wt}_{\mathcal{G}}(\sigma_1, s) \qquad \mathsf{optwt}_{\mathcal{G}}^2(s) = \sup_{\sigma_2 \in \mathsf{Strat}_2^{\mathcal{G}}} \mathsf{wt}_{\mathcal{G}}(\sigma_2, s)$$

One easily notices that $\mathsf{optwt}_{\mathcal{G}}^1(s) \geq \mathsf{optwt}_{\mathcal{G}}^2(s)$ for any s. The converse does not hold in general, but it holds for the class of turn-based games, thanks to Martin's theorem [20]. In the sequel, we call $\mathsf{optwt}_{\mathcal{G}}^1(s)$ the *value* of the game from s (even in the case where $\mathsf{optwt}_{\mathcal{G}}^1(s) \neq \mathsf{optwt}_{\mathcal{G}}^2(s)$), and write it $\mathsf{val}_{\mathcal{G}}(s)$. In the sequel, for all the notations introduced in this paragraph, we may omit to mention the configuration s in case we mean the initial configuration $(\ell_0, \mathbf{0})$. A strategy σ_1 of Player 1 is said *optimal* whenever $\mathsf{wt}_{\mathcal{G}}(\sigma_1) = \mathsf{val}_{\mathcal{G}}$. Given $\epsilon > 0$, σ_1 is said *ϵ-optimal* whenever $\mathsf{wt}_{\mathcal{G}}(\sigma_1) \leq \mathsf{val}_{\mathcal{G}} + \epsilon$. Optimal strategies need not exist, first due to strict clock constraints, or due to more complex convergence phenomena that may happen.

▶ **Example 2.** Consider the weighted timed automaton $\mathcal{G}_{\text{ex}}$ of Fig. 1, and initial configuration $s_0 = (\ell_0, \mathbf{0})$. Locations of Player 1 (resp. Player 2) are depicted with circles (resp. squares), and target location is marked with a double circle. Weight information labels the locations and the transitions (if the weight is 0, then it is omitted on the picture). In this game, Player 1 always reaches the target state. For minimizing the weight, the only choice she has is the time at which she takes the transition leaving ℓ_0. Then Player 2 decides either to switch to location ℓ_2 or to location ℓ_3. We can write the following equation:

$$\mathsf{optwt}_{\mathcal{G}_{\text{ex}}}^1 = \inf_{0 \leq t \leq 2} \max(5t + 10(2-t) + 1, 5t + (2-t) + 7) = 14 + \frac{1}{3}.$$

The optimal strategy for Player 1 is to fire the first transition when $x = \frac{4}{3}$.

2.2 Decision problems

In the following, a *threshold* is a pair $(\bowtie, c)$ (which we more often write $\bowtie c$) with $\bowtie \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{Q}$.

▶ **Definition 3** (existence problem)**.** Given a weighted timed game $\mathcal{G}$ and a threshold $\bowtie c$, the existence problem asks whether there is a strategy σ_1 for Player 1 s.t. for every strategy σ_2 for Player 2, it holds $\mathsf{wt}_{\mathcal{G}}(\sigma_1, \sigma_2) \bowtie c$.

▶ **Definition 4** (value problem)**.** Given a weighted timed game $\mathcal{G}$ and a threshold $\bowtie c$, the value problem asks whether $\mathsf{optwt}^1_{\mathcal{G}} \bowtie c$.

The existence problem has been shown undecidable (for threshold $\leq c$) 10 years ago [14, 9]. In the present paper, we extend this undecidability result to the value problem. We then introduce a large subclass of weighted timed games (imposing a technical condition on the accumulated weight along cycles), and propose an algorithm for computing an approximation (up to any $\epsilon > 0$) of any game in this subclass, together with almost-optimal strategies. Based on this approximation algorithm, we prove that the value problem for threshold $\leq c$ is co-recursively enumerable on our subclass.

3 Undecidability of the value problem

In this section we show that the value problem in weighted timed games is undecidable. More precisely, we reduce the non-halting problem for a two-counter machine to the value problem of weighted timed games with threshold $\leq c$. Our reduction adapts an earlier reduction of the halting problem for a two-counter machine to the existence problem for weighted timed games [9]. The correctness proof makes use of more refined arguments developed in Section 3.2.

3.1 Reduction

We assume the reader is familiar with the model of counter machines, whose (non-)halting problem is known to be undecidable [21]. We fix a deterministic two-counter machine $\mathcal{M}$, and we define a three-counter machine $\mathcal{M}^\star$, which is obtained by adding to $\mathcal{M}$ a third counter, and by inserting an incrementing instruction for this counter after each transition of the original machine $\mathcal{M}$. In particular, $\mathcal{M}$ halts if and only if $\mathcal{M}^\star$ halts.

Our reduction consists in mimicking the behaviour of the deterministic three-counter machine $\mathcal{M}^\star$ using a (turn-based) weighted timed game $\mathcal{G}_{\mathcal{M}^\star}$: the role of Player 1 is to simulate the execution of $\mathcal{M}^\star$, resetting the clocks at well-chosen times so that the counter values c_i are encoded as clock values $1/2^{c_i}$ when entering selected locations. The role of Player 2 is to check that Player 1 simulates the run of the two-counter machine correctly, and in particular that she accurately updates the values of the clocks. At any time, Player 2 can decide to stop simulating the counter machine and leave the game, resulting in a final weight $3+\epsilon$, where ϵ is positive if Player 1 did not simulate the run of $\mathcal{M}^\star$ correctly. Player 1 in turn can decide to stop the simulation and to leave the game at any time, securing a final weight $3+\alpha_N$, where $\alpha_N > 0$ tends to zero when the length N of the execution simulated so far tends to $+\infty$.

If the machine does not halt, Player 1 can accurately simulate the machine for a large number of steps, before leaving the game when $\alpha_N < \epsilon$. Hence for any $\epsilon > 0$, Player 1 can secure final weight at most $3+\epsilon$, and the value of the game is 3. Conversely, if the machine does halt, its unique computation has finite length N; our construction enforces that Player 1 will not be able to secure final accumulated weight better than $3+\beta_N$ for some $\beta_N > 0$ (which only depends on N), yielding value 3 for the game.

We now explain this construction in more details. The construction is based on a few simple modules that we then plug together. Those modules explicitly use some clocks,

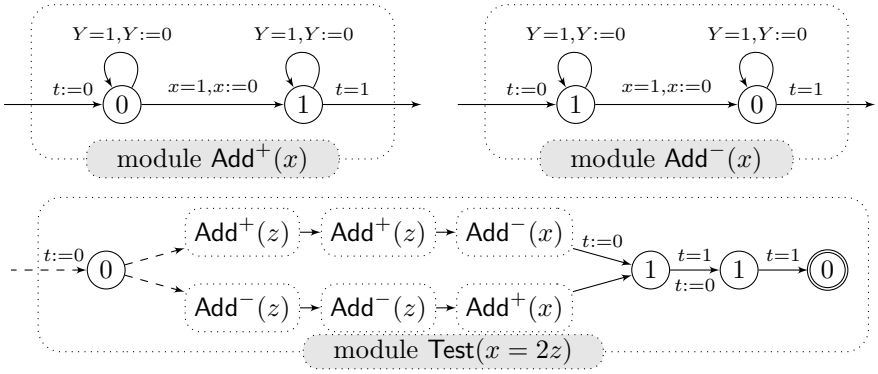


Figure 4 Modules Add^+ and Add^-: the weight is increased by x_0, resp. $1-x_0$ (x_0: initial value of x when entering the module). Module $\mathsf{Test}(x=2z)$: Player 2 can increase the cost by $3+|x_0-2z_0|$.

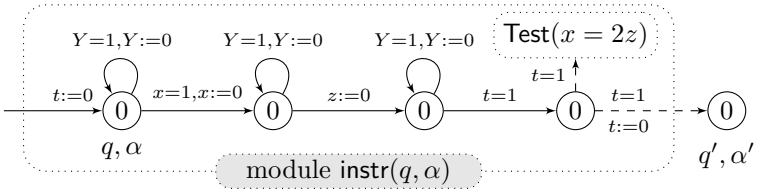


Figure 5 Module $\mathsf{instr}(q,\alpha)$ encoding an incrementing transition (q,i,q').

while some other clocks are only useful for the global reduction; the values of the latter are then preserved by each module, thanks to self-loops on all locations (we symbolically write $Y=1, Y:=0$ to indicate that each other clock is reset when it reaches value 1) and to an (implicit) global invariant requiring that no clock may exceed 1.

3.1.1 Comparing clock values

As sketched above, Player 2 is in charge of checking that Player 1 updates the clocks so as to preserve the encoding $x=1/2^c$ (c is a counter value). When incrementing this counter, this amounts to checking that clock z (after the increment) equals $x/2$. Following the ideas of [9], we build a module $\mathsf{Test}(x=2z)$ in which Player 2 can achieve final accumulated weight $3+|x-2z|$. In other words, the final weight is 3 if $x=2z$ when entering this module, and it is strictly more otherwise. Technically, as shown on Fig. 4, this is achieved by offering two branches to Player 2: one in which the final weight is $3+x-2z$ and one where it is $3+2z-x$. Hence Player 2 can enforce cost $2+|x-2z|$. Accumulating these weights is easily achieved by elapsing delays x, z, $1-z$ and $1-x$ in locations with local weights 0 and 1.

3.1.2 Incrementing and decrementing counters

Incrementing counter c is achieved by asking Player 1 to reset some clock z at a well-chosen time, in such a way that $z=x/2$ when Player 2 is given the opportunity to enter the Test module. The new value of the counter is then encoded by clock z.

Figure 5 displays the module used for encoding increments: clock t serves as a tick ($t=0$ when entering the module, and $t=1$ at the end); clock x encodes the value of the counter initially (that is, $x=1/2^c$ when entering the module), while clock z is used to encode the value of the same counter after the increment (hence $z=x/2$ at the end of the module).

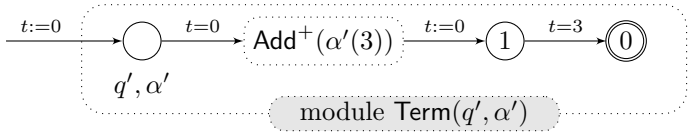


Figure 6 Module $\mathsf{Term}(q', \alpha')$: $\alpha'(3)$ is the clock encoding the third counter, which counts the number of steps simulated so far.

Finally, notice that the module depends on a function α, which is used to keep track of which clock encodes which counter.

Decrementing counter c follows the same idea, but it first performs a zero-test at entrance. Counter $c = 0$ if, and only if, the corresponding clock $x = 1$ when entering the module (i.e., when clock $t = 0$).

3.1.3 Leaving the game

In the game built so far, Player 1 does not have a way to leave for sure to a final location (e.g. when the counter machine does not halt). We give her the opportunity to do so right after incrementing the third counter (which is done every two instructions), by plugging a copy of module Term of Fig. 6 in the corresponding locations.

Notice that this transition is the only possible transition in the locations corresponding to q_{halt}.

3.2 Analysis of the construction

Our construction does not check that the values of the clocks are of the form $1/2^c$. Hence we don't have a correspondence between configurations of $\mathcal{G}_{\mathcal{M}^\star}$ and configurations of $\mathcal{M}^\star$. We define *pseudo-configurations* of $\mathcal{M}^\star$ to tackle this problem: a *pseudo-configuration* of $\mathcal{M}^\star$ is a pair $\gamma = (q, v)$, where q is a discrete state of $\mathcal{M}^\star$ and $v\colon \{1, 2, 3\} \to \mathbb{R}_{\geq 0}$ assigns a non-negative *real* number to every counter. A *pseudo-run* in $\mathcal{M}^\star$ is a sequence of pseudo-configurations. Let $\rho^\star = \gamma_0^\star \to \gamma_1^\star \to \dots$ be the unique maximal (finite or infinite) execution of $\mathcal{M}^\star$. Consider a (finite or infinite) pseudo-run $\rho = \gamma_0 \to \gamma_1 \to \dots$. If there is some $k \geq 0$ such that the discrete states of γ_k and $\gamma_k^\star$ do not coincide, we say that ρ is *strongly-invalid*, which means that compared to the valid run $\rho^\star$, some discrete transition has not been taken appropriately. Writing k_0 for the first position where this occurs, the consecution $\gamma_{k_0-1} \to \gamma_{k_0}$ is said to be *strongly-invalid*. If all discrete states coincide, but ρ is not a prefix of $\rho^\star$, we say that ρ (and its erroneous consecutions) are *weakly-invalid* (in that case, some counter values may not be correctly encoded, but this has no impact on the sequence of visited states, hence on the nature – halting, non-halting – of the path).

Let $\sigma_2^\perp$ be the strategy of Player 2 that consists in never switching to a Test module (*i.e.*, it remains in the *main part* of the game). With any strategy $\sigma_1 \in \mathsf{Strat}_1(\mathcal{G}_{\mathcal{M}^\star})$, we associate the unique maximal outcome in $\mathsf{Out}_{\mathcal{G}_{\mathcal{M}^\star}}(\sigma_1, \sigma_2^\perp)$ (it can either end at a target location of a terminating module, or be infinite). When entering the modules of the instructions, it is clear from the syntax which clock encodes which counter, hence we can extract from that path the sequence of configurations (q_k, α_k, ν_k) when entering (or leaving) an instruction module, where q_k is the discrete state, α_k is the encoding mapping, and ν_k is the clock valuation. The corresponding counter values can be recovered by defining $v_k(i) = -\log_2(\nu_k(\alpha_k(i)))$, and we thus have that $\gamma_k = (q_k, v_k)$ is a pseudo-configuration of $\mathcal{M}^\star$. We then write ρ_{σ_1} for the pseudo-run $\gamma_0 \to \gamma_1 \to \dots$ associated with σ_1.

The following lemma is the crux of our proof: it states that the earlier an occurrence of a strongly-invalid transition, the larger the penalty that can be inflicted to Player 1.

▶ **Lemma 5.** *Let $\sigma_1 \in \mathsf{Strat}_1^{\mathcal{G}_{\mathcal{M}^\star}}$ be a strategy such that ρ_{σ_1} is a strongly-invalid pseudo-run of $\mathcal{M}^\star$. Let $\gamma_k \longrightarrow \gamma_{k+1}$ be the first strongly-invalid consecution of ρ_{σ_1}. Then $k > 0$, and there is a strategy $\sigma_2 \in \mathsf{Strat}_2^{\mathcal{G}_{\mathcal{M}^\star}}$ such that $\mathsf{wt}_{\mathcal{G}_{\mathcal{M}^\star}}(\sigma_1, \sigma_2) \geq 3 + \frac{1}{4k \cdot 2^k}$.*

Using this lemma, we can prove:

▶ **Proposition 1.** *The counter machine $\mathcal{M}^\star$ does not halt if, and only if, the value of game $\mathcal{G}_{\mathcal{M}^\star}$ is (at most) 3.*

Sketch of proof. Assume $\mathcal{M}^\star$ halts, and let N be the length of the halting computation. We argue that the value of the game is lower-bounded by $3 + \epsilon_N$, where ϵ_N is a positive number that depends only on N. Then, either Player 1 decides to leave the game at some point before having simulated the N steps of the counter machine, or Player 1 cheats so that the simulation goes for longer than N steps. In the former case, the weight will be given by gadget Term; it will be $3+\alpha_N$, where α_N is the value of the clock encoding third counter; it is lower-bounded, since the third counter is upper-bounded by N. In the latter case, there must be a strongly-invalid consecution before N steps have been simulated; applying Lemma 5 with $k \leq N$, we again get a lower-bound of the form $3 + \beta_N$ for some positive β_N. In both cases, the weight of the strategy is lower-bounded by a constant that only depends on N.

If $\mathcal{M}^\star$ does not halt, then Player 1 can correctly mimic the counter machine, and leave the game after an arbitrary long simulation, yielding a weight arbitrary close to 3. Hence the value of the game is 3. ◀

This reduction proves undecidability of the value problem for thresholds $= c$ and $\leq c$. By swapping the roles of Players 1 and 2 and slightly modifying the construction, we can prove that the value problem is also undecidable for threshold $\geq c$.

4 Computing an approximation of the value

In this section, we introduce a subclass of weighted timed games, and explain how to approximate the values of games in this class. Our subclass is the set of games $\mathcal{G}$ for which there exists $\kappa > 0$ such that for any finite run in the game that follows a region cycle of the region automaton[1] of $\mathcal{G}$, either the weight is 0, or it is larger than or equal to κ;[2] We call such games *almost strongly non-Zeno weighted timed games*. It is decidable whether a weighted timed game is almost strongly non-Zeno or not (by enumerating all simple cycles of the region abstraction of $\mathcal{G}$). Notice that the undecidability proof for the existence problem [9], as well as our undecidability proof for the value problem in Section 3, is valid for this subclass of weighted timed games. Finally note that if we strengthen the first condition above by assuming that all cyclic runs have weight at least κ (forbidding zero cycles), then we get the class of strongly non-Zeno weighted timed games, for which the value can be computed exactly [1, 11].

In the sequel, we prove the approximability of the optimal weight:

▶ **Theorem 6.** *Given an almost strongly non-Zeno weighted timed game $\mathcal{G}$ and a positive real ϵ, we can compute a rational v, and a strategy σ_1 for Player 1, such that $|v - \mathsf{val}_{\mathcal{G}}| \leq \epsilon$ and $|\mathsf{wt}_{\mathcal{G}}(\sigma_1) - v| \leq \epsilon$.*

[1] We assume familiarity with region equivalence, and refer to [3] for details.
[2] Applying results for weighted timed automata [10], we may assume $\kappa = 1$.

4.1 A basic characterization of the value.

The following fixpoint characterization was given in [11]:

▶ **Proposition 2** ([11]). *Optimal weight for Player* 1 *is the largest fixpoint of the following equation. For every state* s *of* $\mathcal{G}$*:*

$$\mathit{val}_{\mathcal{G}}(s) = \inf_{\substack{d\geq 0, e\in E_1 \\ s\xrightarrow{d,e}s'}} \max \left\{ \begin{array}{ll} (1) & d\cdot \mathsf{wt}(\ell) + \mathsf{wt}(e) + \mathit{val}_{\mathcal{G}}(s'), \\ (2) & \sup\limits_{\substack{d'\leq d, e'\in E_2 \\ s\xrightarrow{d',e'}s''}} \big(d'\cdot \mathsf{wt}(\ell) + \mathsf{wt}(e') + \mathit{val}_{\mathcal{G}}(s'')\big) \end{array} \right\}$$

However there is no obvious good property of this functional (that we know of) that could be useful for designing an approximation algorithm. Instead, we will partially unfold the game in a careful way in order to preserve the optimal weight, and prove that we can approximate the value of the resulting game.

4.2 The semi-unfolded game

In order to approximate the value of the game, we first build a tree-shaped weighted timed game $\widetilde{\mathcal{G}}$, with the same value as $\mathcal{G}$. Then we explain how to approximate the value of $\widetilde{\mathcal{G}}$.

Let W be an upper bound on the optimal weight from every winning state of $\mathcal{G}$; such a bound is easy to compute, *e.g.* by picking a memoryless and region-uniform winning strategy for Player 1 (in the underlying timed game) [5], and computing a bound on its weight from all configurations.

We write $\mathcal{R}(\mathcal{G})$ for the timed game obtained from $\mathcal{G}$ by splitting its state space into regions (that is, we apply the standard region-automaton construction [3] and interpret it as a timed game). The weighted timed game $\mathcal{R}(\mathcal{G})$ is called the *region game* of $\mathcal{G}$. In $\mathcal{R}(\mathcal{G})$, we additionally assume – for technical reasons – that for every state (ℓ, r) with $\mathsf{wt}(\ell) > 0$, for every $v \in r$, there is no transition $(\ell, v) \xrightarrow{0,e}$. This can be achieved by a simple transformation (at the expense of an extra clock to isolate the case where no time is elapsed in (ℓ, r)), hence it causes no loss of generality.

The values $\mathsf{val}_{\mathcal{G}}$ and $\mathsf{val}_{\mathcal{R}(\mathcal{G})}$ obviously coincide, as there is a tight correspondence between the runs in both games. Now, in game $\mathcal{R}(\mathcal{G})$, we mark in green all locations with weight 0, as well as all edges with discrete weight 0. All other locations and edges are marked in red. Fully green cycles in $\mathcal{R}(\mathcal{G})$ (involving only green locations and edges) then characterize cycles with weight 0. We define the *kernel* $\mathcal{K}$ of $\mathcal{G}$ as the restriction of $\mathcal{R}(\mathcal{G})$ to fully-green strongly connected components. Edges that leave $\mathcal{K}$ are called the *output edges* of $\mathcal{K}$. Notice that any segment of a run from an output edge back to the kernel must visit a red state or a red edge.

The idea is to partially unfold the game $\mathcal{R}(\mathcal{G})$, to obtain a finite tree-like structure (see the left part of Fig. 7). Before formally describing the construction, we begin by informally explaining how it is obtained: we first unfold the game $\mathcal{R}(\mathcal{G})$, and along a branch, as soon as we enter the kernel, we put a copy of $\mathcal{K}$ (removing all states that are not reachable, but without unfolding $\mathcal{K}$); we restart unfolding again from the output edges of that copy of $\mathcal{K}$. We stop this process when along any branch, a red state or edge of $\mathcal{R}(\mathcal{G})$ is visited at least $W + 2$ times.

We now formalize this construction. We first build a tree $\mathcal{T}$, which carries labels both on its nodes and on the edges between nodes. The root n_0 of $\mathcal{T}$ is labelled with the initial location (ℓ_0, r_0) of $\mathcal{R}(\mathcal{G})$. The tree is then built inductively, starting from the root:

- If a node n is labelled with (ℓ, r), then, for every transition $(\ell, r) \xrightarrow{g,Y} (\ell', r')$ in $\mathcal{R}(\mathcal{G})$, we consider two cases:

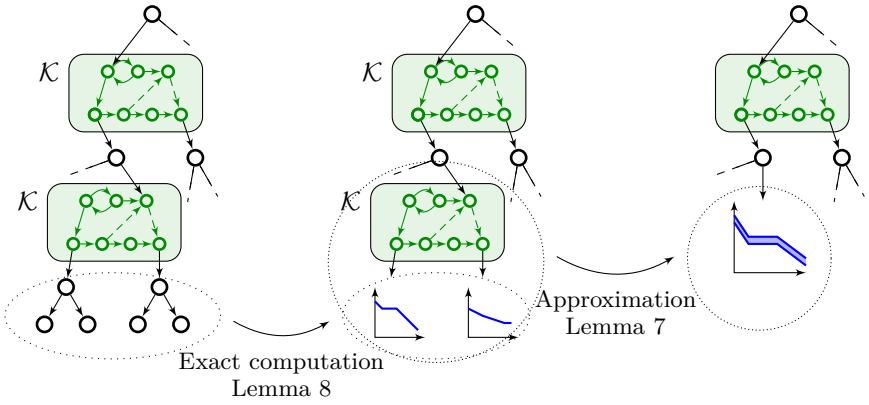


Figure 7 Approximation scheme: in the tree-shaped parts of the semi-unfolding, an exact computation can be performed; in the kernels, we apply under- and over-approximation algorithms.

- if (ℓ', r') belongs to the kernel $\mathcal{K}$, then node n has a son n' labelled with $\mathcal{K}_{(\ell',r')}$;
- Otherwise, n has a son n' labelled with (ℓ', r').

In both cases, the edge between n and n' is labelled with (g, Y).

- If n is labelled with $\mathcal{K}_{(\ell,r)}$, then, for every output edge $e = (\ell'', r'') \xrightarrow{g,Y} (\ell', r')$ of $\mathcal{K}$ that is reachable from (ℓ, r), we add a son n_e labelled with (ℓ', r'). We label the corresponding edge with (g, Y).

We stop this construction as soon as all the branches of this tree contain either $W + 2$ occurrences of the same pair (ℓ, r), or $W + 2$ occurrences of a red transition. We require all the branches to end with some non-kernel node. One quickly realizes that tree $\mathcal{T}$ is finite.

From this tree $\mathcal{T}$, we define a weighted timed game $\widetilde{\mathcal{G}}$, by replacing the kernel nodes with copies of the kernel. More formally, pick a node n labelled with $\mathcal{K}_{(\ell,r)}$, and write n_1 to n_k for its sons. These sons originate from output edges e_1 to e_k of the kernel, which leave some locations s_1 to s_k of the kernel. We then replace node n with a set of locations (n, s), one for each location $s \in \mathcal{K}_{(\ell,r)}$. The edge entering n is now directed to $(\mathsf{n}, (\ell, r))$. Each edge (s, g, Y, s') in the kernel gives rise to an edge $((\mathsf{n}, s), g, Y, (\mathsf{n}, s'))$. And finally, each son n_i of the former node n has an incoming edge from location $(\mathsf{n}, \mathsf{s}_i)$, labelled in the same way as the former edge e_i.

After adding self loops on the leaves with weight 0, and importing the weights on locations and edges and the partition of edges between both players, we end up with a weighted timed game $\widetilde{\mathcal{G}}$, which we can prove satisfies the following:

▶ **Proposition 3.** *The values of games $\mathcal{G}$ and $\widetilde{\mathcal{G}}$ coincide.*

▶ Remark. It is worth noticing that the tree built from a game used in the undecidability reduction (Section 3) is one single kernel with finite trees from the output edges (corresponding to the (acyclic) test or leaving modules). Any outcome visits only once the kernel.

4.3 Approximation algorithm

The approximation algorithm is made of two distinct sub-algorithms (see Fig. 7):

- the **tree-shaped part analysis:** an exact computation of the optimal weight can be achieved in each tree-shaped part [19, 1];
- the **kernel analysis:** this is the difficult part of the algorithm. The idea is to under- and over-approximate the optimal weight computed so far from the output edges by piecewise-constant functions, that are constant over subregions with a known small granularity.

After this transformation, the game played within a kernel is a *(non-weighted) timed game* with an extended reachability condition over output edges, which can be solved with basic techniques. Each kernel analysis induces a bounded imprecision in the computation.

The algorithm then proceeds upwards from the leaves to the root of the built tree, alternatively applying the analysis of the tree-shaped parts and of the kernel.

In order to describe the approximation algorithm, we consider an extension of the model of weighted timed games with *outside weight functions* associated to target locations [13]. A generalized weighted timed game is now a tuple $\langle L, \ell_0, L_{\mathsf{f}}, X, E, E_1, E_2, \mathsf{wt}, \mathsf{outwt}\rangle$, where the first eight components form a weighted timed game as of Def. 1, and $\mathsf{outwt}\colon L_{\mathsf{f}} \to (\mathbb{R}_+^X \to (\mathbb{R}_+ \cup \{+\infty\}))$ assigns a weight function with every target location. Compared to classical weighted timed games, the weight of a run ρ reaching a target state is augmented by $\mathsf{outwt}(\ell)(v)$, assuming $last(\rho) = (\ell, v)$ with $\ell \in L_{\mathsf{f}}$. All notations are extended to this model in a straightforward way. From a given configuration $(\ell_{\mathsf{init}}, v)$, the aim of Player 1 now is to reach the target states while minimizing this modified weight. Following [13], we use this natural extension of the original model to iteratively compute the value in $\widetilde{\mathcal{G}}$.

The main idea of our algorithm is to approximate the value of outside weight functions by piecewise-constant functions. For this, we split regions into smaller sets that we call *regions of granularity* $1/2^N$, or simply $1/2^N$*-regions*, hereafter. Formally, a set R is a $1/2^N$-region for a maximal constant M if, and only if, the set $2^N \cdot R$ (obtained by scaling R by 2^N in all dimensions) is a region for maximal constant $2^N \cdot M$. In order to be able approximate outside weight functions with piecewise-constant functions, we have to restrict them: an outside weight function outwt is said smooth whenever there exists an integer M and a granularity $1/2^N$ such that for every $\ell \in L_{\mathsf{f}}$, the function $\mathsf{outwt}(\ell)$ is uniformly continuous (or constantly equal to $+\infty$) over $1/2^N$-regions for maximal constant M. We additionally require that within any $1/2^N$-region, the function $\mathsf{outwt}(\ell)$ does not depend on the exact values of the clocks whose value is larger than M: for any two clock valuations v and v' in the same $1/2^N$-region such that $v(x) = v'(x)$ whenever $v(x) \leq M$ or $v'(x) \leq M$, it holds $\mathsf{outwt}(\ell)(v) = \mathsf{outwt}(\ell)(v')$. Notice that contrary to [13], we impose no other conditions (affineness, monotonicity) on outside weight functions.

Computing the value in the kernel. We now explain how we compute an approximation of the value in the kernel, assuming that we have smooth outside weight functions in the locations reached by output edges.

▶ **Lemma 7.** *Consider a maximal strongly connected component K of the kernel $\mathcal{K}$ of some region game $\mathcal{R}(\mathcal{G})$, and pick a state (ℓ, r) of K. Let $S_O = \{(\ell_e, r_e) \mid e \text{ output edge of } K\}$ be the set of target locations in $\mathcal{R}(\mathcal{G})$ of the output edges of K. Define a game $\mathcal{H} = \langle K \cup S_O, (\ell, r), S_O, X, E_{|K\cup S_0}, E_1 \cap E_{|K\cup S_0}, E_2 \cap E_{|K\cup S_0}, \mathsf{wt}_{|\mathcal{H}}, \mathsf{outwt}\rangle$, where outwt is a smooth outside weight function with maximal partial derivative P.*

Then, for every $\epsilon > 0$, we can compute ϵ-over-approximations and ϵ-under-approximations of the value of the game $\mathcal{H}$ at any configuration (ℓ, v) with $v \in r$, and (2ϵ)-optimal strategies from all those configurations. Additionally, the computed approximations are constant on each $1/2^N$-subregion of r, as soon as $1/2^N \leq \epsilon/P$. Finally, the computation can be achieved in time $O(|\mathcal{R}(\mathcal{G})| \cdot (P/\epsilon)^{|X|})$.

The key idea behind that lemma is to under- and over-approximate smooth outside weight functions by constant functions over refined ($1/2^N$-grained) regions, which then reduces the game to some extended reachability timed game: each output edge is assigned a single value by the computed constant outside weight function (the constant under- or

over-approximation), and this value is the weight for leaving *via* this edge (there is no weight involved elsewhere in the kernel); hence this defines a preference order over output edges, and the aim of Player 1 now is to enforce the 'less expensive' edge; this game is timed, but with a (qualitative) extended-reachability condition over output edges; it can be solved using standard attractor techniques over timed games.

Computing the value in a finite tree. The computation of the optimal weight in tree-shaped parts of the weighted timed games $\widetilde{\mathcal{G}}$ is known to be computable [19, 1] using an iterative backward algorithm (which can also compute almost-optimal strategies) based on equations given in Prop. 2. In our setting, the techniques developed in [1] entail the following lemma:

▶ **Lemma 8.** *Let $\mathcal{S}$ be a finite unfolding of a region game $\mathcal{R}(\mathcal{G})$ from some region (ℓ_0, r_0). We equip $\mathcal{S}$ with an outside weight function outwt associating with every leaf (ℓ, r) of $\mathcal{S}$ a function from r to $\mathbb{R}_{\geq 0}$. We require that there exists an integer N such that for any terminal node (ℓ, r), $\mathsf{outwt}(\ell, r)$ is constant over $1/2^N$-subregions of r.*

Then we can compute the function $v \in r_0 \mapsto \mathsf{val}_{\mathcal{S}}(\ell_0, v)$ in time $2^{O(N\cdot|X|^2+|\mathcal{S}|^2)}$. Moreover, those functions are piecewise-affine, with partial derivatives in $\{0\} \cup \{\mathsf{wt}(\ell) \mid \ell \in L\}$.

Global algorithm. Our general algorithm consists in iteratively applying the two lemmas above, hence computing approximation functions from the leaves to the root of the tree $\mathcal{T}$. More precisely, fix $\epsilon > 0$, and pick a node n of $\mathcal{T}$. This node is labelled either with (ℓ, r), or with $\mathcal{K}_{(\ell,r)}$. Our algorithm computes two functions $f^+_\epsilon(\mathsf{n})\colon r \to \mathbb{R}$ and $f^-_\epsilon(\mathsf{n})\colon r \to \mathbb{R}$ that are smooth and respectively ϵ-over-approximate and ϵ-under-approximate the value of the game $\widetilde{\mathcal{G}}$ from all configurations in the region (ℓ, r) corresponding to node n.

Write $\epsilon' = \epsilon/(|\mathcal{R}(\mathcal{G})| \cdot (W+2)+1)$. We begin with initializing the computation at the leaves of $\mathcal{T}$. Let n be a leaf. By construction, it must be labelled with some (ℓ, r). We define $f^+_{\epsilon'}(\mathsf{n})$ and $f^-_{\epsilon'}(\mathsf{n})$ as follows:

- if $\ell \in L_{\mathsf{f}}$, we let $f^+_{\epsilon'}(\mathsf{n})(v) = f^-_{\epsilon'}(\mathsf{n})(v) = 0$ for every $v \in r$;
- otherwise, we let $f^+_{\epsilon'}(\mathsf{n})(v) = f^-_{\epsilon'}(\mathsf{n})(v) = +\infty$ for every $v \in r$.

These functions obviously correspond to the exact value. Moreover, they fulfill the hypothesis of both Lemmas 7 and 8.

Now pick a node n of $\mathcal{T}$, assuming that each son n' of n has been assigned approximation functions $f^+_{k\epsilon'}(\mathsf{n})$ and $f^-_{k\epsilon'}(\mathsf{n})$. We consider two cases:

- if n is labelled with (ℓ, r) and is the son of a kernel node, then we apply Lemma 8 to compute $f^+_{k\epsilon'}(\mathsf{n})$ and $f^-_{k\epsilon'}(\mathsf{n})$;
- if n is labelled with $\mathcal{K}_{(\ell,r)}$, then we apply Lemma 7 with approximation value ϵ' and smooth outside weight function $f^+_{k\epsilon'}(\mathsf{n}')$ (resp. $f^-_{k\epsilon'}(\mathsf{n}')$), and get piecewise-constant weight functions $f^+_{(k+1)\epsilon'}(\mathsf{n})$ (resp. $f^-_{(k+1)\epsilon'}(\mathsf{n})$).

The correctness of the algorithm is stated as follows:

▶ **Lemma 9.** *Pick a node n of $\mathcal{T}$ labelled by (ℓ, r) or $\mathcal{K}_{(\ell,r)}$. Assume node n has been assigned a value by $f^+_{k\epsilon'}$ and $f^-_{k\epsilon'}$. Then for every $v \in r$,*

$$f^-_{k\epsilon'}(\mathsf{n})(v) \leq \mathsf{val}_{\widetilde{\mathcal{G}}}(\ell, v) \leq f^+_{k\epsilon'}(\mathsf{n})(v).$$

Furthermore, for any node n where $f^-_{k\epsilon'}(\mathsf{n})$ and $f^+_{k\epsilon'}(\mathsf{n})$ are defined, it holds

$$\left|f^+_{k\epsilon'}(\mathsf{n})(v) - f^-_{k\epsilon'}(\mathsf{n})(v)\right| \leq k \cdot \epsilon'.$$

Additionally, we can compute $(2k\epsilon')$-optimal winning strategies.

The maximal number of kernels along any branch of $\mathcal{T}$ being bounded by $(W+2)\cdot|\mathcal{R}(\mathcal{G})|+1$, there exists $k \leq (W+2)\cdot|\mathcal{R}(\mathcal{G})|+1$ for which $f^{-}_{k\epsilon'}$ and $f^{+}_{k\epsilon'}$ are defined at the root of $\mathcal{T}$. Those functions ϵ-under-approximate and ϵ-overapproximate the value of $\mathcal{G}$, as required.

Complexity of the algorithm. Our algorithm inductively applies Lemma 7 on each copy of the kernel, and Lemma 8 on each intermediary tree between kernels. This may result in $|\mathcal{R}(\mathcal{G})|^{(W+2)\cdot|\mathcal{R}(\mathcal{G})|+1}$ applications of both lemmas. As W can be bounded by $|\mathcal{R}(\mathcal{G})|\cdot P$ where P is the maximal rate appearing in the automaton, the time-complexity is bounded by $\left(\frac{1}{\epsilon}\right)^{|X|^2}\cdot 2^{O(|\mathcal{R}(\mathcal{G})|^4)}$ (hence doubly-exponential in the size of the original timed game).

4.4 The value problem is (co-)recursively enumerable

Using our approximation algorithm, we immediately get the following result:

▶ **Theorem 10.** *Over the class of almost strongly non-Zeno weighted timed games, the value problem is co-recursively enumerable for thresholds $\leq c$, $= c$ and $\geq c$ (for $c \in \mathbb{Q}$). It is recursively enumerable for thresholds $< c$ and $> c$ (with $c \in \mathbb{Q}$).*

Proof. We prove this result using the approximation algorithm above. We consider the case of threshold $\leq c$, but the other cases are similar. Given a game $\mathcal{G}$, we build a Turing machine that halts if, and only if, the value of $\mathcal{G}$ is **not** less than or equal to c.

The machine runs as follows: it first sets the tolerance ϵ to 1, and the approximation v to $-\infty$. Then, as long as $c \geq v-\epsilon$, it divides ϵ by 2, and computes an ϵ-approximation v of the value of $\mathcal{G}$ using the approximation algorithm above.

One easily sees that if the value of $\mathcal{G}$ is indeed larger than c, then eventually ϵ will be less than $\mathsf{val}_{\mathcal{G}}-c$, and the machine will halt. Conversely, if the value of $\mathcal{G}$ is larger than or equal to c, the machine will run forever. ◀

5 Conclusion

We proved in this paper that for weighted timed games, no algorithm can compute the value of the game. On the other hand, we developed an approximation algorithm for a large subclass of weighted timed games, for which the undecidability proof already applies.

We have two natural ways for continuing this work: the first one is to extend the algorithm to handle the full class of weighted timed automata; the second one is to improve the complexity of our approximation algorithm, which currently is doubly-exponential.

References

1 Rajeev Alur, Mikhail Bernadsky, and P. Madhusudan. Optimal reachability in weighted timed games. In *ICALP'04*, volume 3142 of *LNCS*, pages 122–133. Springer, 2004.

2 Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei-Hsin Ho. Hybrid automata: an algorithmic approach to specification and verification of hybrid systems. In *HSCC'91-'92*, volume 736 of *LNCS*, pages 209–229. Springer, 1993.

3 Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

4 Rajeev Alur, Salvatore La Torre, and George J. Pappas. Optimal paths in weighted timed automata. In *HSCC'01*, volume 2034 of *LNCS*, pages 49–62. Springer, 2001.

5 Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. Controller synthesis for timed automata. In *Proc. IFAC Symp. System Structure & Control*, pages 469–474. Elsevier, 1998.

6 Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim G. Larsen, Paul Pettersson, Judi Romijn, and Frits Vaandrager. Minimum-cost reachability for priced timed automata. In *HSCC'01*, volume 2034 of *LNCS*, pages 147–161. Springer, 2001.

7 Gerd Behrmann, Kim G. Larsen, and Jacob I. Rasmussen. Optimal scheduling using priced timed automata. *ACM Sigmetrics Performance Eval. Review*, 32(4):34–40, 2005.

8 Patricia Bouyer, Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On the optimal reachability problem. *Formal Methods in System Design*, 31(2):135–175, 2007.

9 Patricia Bouyer, Thomas Brihaye, and Nicolas Markey. Improved undecidability results on weighted timed automata. *Information Processing Letters*, 98(5):188–194, 2006.

10 Patricia Bouyer, Ed Brinksma, and Kim G. Larsen. Optimal infinite scheduling for multi-priced timed automata. *Formal Methods in System Design*, 32(1):2–23, 2008.

11 Patricia Bouyer, Franck Cassez, Emmanuel Fleury, and Kim G. Larsen. Optimal strategies in priced timed game automata. In *FSTTCS'04*, volume 3328 of *LNCS*, pages 148–160. Springer, 2004.

12 Patricia Bouyer, Samy Jaziri, and Nicolas Markey. On the value problem in weighted timed games. Research Report LSV-14-12, Laboratoire Spécification et Vérification, ENS Cachan, France, October 2014. 24 pages.

13 Patricia Bouyer, Kim G. Larsen, Nicolas Markey, and Jacob I. Rasmussen. Almost optimal strategies in one-clock priced timed automata. In *FSTTCS'06*, volume 4337 of *LNCS*, pages 345–356. Springer, 2006.

14 Thomas Brihaye, Véronique Bruyère, and Jean-François Raskin. On optimal timed strategies. In *FORMATS'05*, volume 3821 of *LNCS*, pages 49–64. Springer, 2005.

15 Thomas Brihaye, Gilles Geeraerts, Shankara Narayanan Krishna, Lakshmi Manasa, Benjamin Monmege, and Ashutosh Trivedi. Adding negative prices to priced timed games. In *CONCUR'14*, volume 8704 of *LNCS*, pages 560–575. Springer, 2014.

16 Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In Roberto Amadio and Denis Lugiez, editors, *CONCUR'03*, volume 2761 of *LNCS*, pages 142–156. Springer, 2003.

17 Thomas Dueholm Hansen, Rasmus Ibsen-Jensen, and Peter Bro Miltersen. A faster algorithm for solving one-clock priced timed games. In *CONCUR'13*, volume 8052 of *LNCS*, pages 531–545. Springer, 2013.

18 Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94–124, 1998.

19 Salvatore La Torre, Supratik Mukhopadhyay, and Aniello Murano. Optimal-reachability and control for acyclic weighted timed automata. In *TCS'02*, volume 223 of *IFIP Conf. Proc.*, pages 485–497. Kluwer, 2002.

20 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, 102:363–371, 1975.

21 Marvin Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1967.

22 Michał Rutkowski. Two-player reachability-price games on single-clock timed automata. In *QAPL'11*, volume 57 of *ENTCS*, pages 31–46, 2011.

Repairing Multi-Player Games*

Shaull Almagor, Guy Avni, and Orna Kupferman

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

Abstract

Synthesis is the automated construction of systems from their specifications. Modern systems often consist of interacting components, each having its own objective. The interaction among the components is modeled by a *multi-player game*. Strategies of the components induce a trace in the game, and the objective of each component is to force the game into a trace that satisfies its specification. This is modeled by augmenting the game with ω-regular winning conditions. Unlike traditional synthesis games, which are zero-sum, here the objectives of the components do not necessarily contradict each other. Accordingly, typical questions about these games concern their stability – whether the players reach an equilibrium, and their social welfare – maximizing the set of (possibly weighted) specifications that are satisfied.

We introduce and study *repair* of multi-player games. Given a game, we study the possibility of modifying the objectives of the players in order to obtain stability or to improve the social welfare. Specifically, we solve the problem of modifying the winning conditions in a given concurrent multi-player game in a way that guarantees the existence of a *Nash equilibrium*. Each modification has a value, reflecting both the cost of strengthening or weakening the underlying specifications, as well as the benefit of satisfying specifications in the obtained equilibrium. We seek optimal modifications, and we study the problem for various ω-regular objectives and various cost and benefit functions. We analyze the complexity of the problem in the general setting as well as in one with a fixed number of players. We also study two additional types of repair, namely redirection of transitions and control of a subset of the players.

1998 ACM Subject Classification J.4 Social and Behavioral Sciences, F.1.2 Modes of Computation

Keywords and phrases Nash Equilibrium, Concurrent games, Repair

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.325

1 Introduction

Synthesis is the automated construction of systems from their specifications [19]. Modern systems often consist of interacting components, each having its own objective. The interaction among the components is modeled by a *multi-player game*. Each player in the game corresponds to a component in the interaction. In each round of the game, each of the players chooses an action, and the next vertex of the game depends on the current vertex and the vector of actions chosen. A strategy for a player is then a mapping from the history of the game so far to her next action.

The strategies of the players induce a trace in the game, 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 winning conditions, describing the objectives of the players. Unlike traditional synthesis games, which are zero-sum, here the objectives of the

* The research has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement no 278410.

players do not necessarily contradict each other. Accordingly, typical questions about these games concern their stability – whether the players reach an equilibrium, and their social welfare – maximizing the set of (possibly weighted) specifications that are satisfied [23].

Different types of games can model different schemes of interaction among the components. In particular, we distinguish between *turn-based* and *concurrent* games. In the first, a single player chooses an action and determines the successor vertex in each step of the interaction. In the second, all players choose actions in all steps [1]. Another parameter is the way in which the winning conditions in the game are specified. Most common are *reachability*, *Büchi*, *co-Büchi*, and *parity* winning conditions [17], which are used to specify the set of winning traces. [1] As for stability and social welfare, here too, several types have been suggested and studied. The most common criterion for stability is the existence of a *Nash equilibrium* (NE) [18]. A profile of strategies, one for each player, is an NE if no (single) player can benefit from unilaterally changing her strategy. In the general setting of game theory, the outcome of a game fixes a reward to each of the players, thus "benefiting" stands for increasing the reward. In our setting here, the objective of a player is to satisfy her specification. Accordingly, "benefiting" amounts to moving from the set of losers – those players whose specifications are not satisfied, to the set of winners – those whose specifications are satisfied.

In [7, 22], the authors study the existence of an NE in games with Borel objectives. It turns out that while a turn-based game always has an NE [7, 22], this is not the case for concurrent games [2]. The problem of deciding whether a given concurrent game has an NE can be solved in polynomial time for Büchi games, but is NP-complete for reachability and co-Büchi games. Interestingly, this is one of the few examples in which reasoning about the Büchi acceptance condition is easier than reasoning about co-Büchi and reachability. The above results hold for a model with *nondeterministic* transition functions and with *imperfect monitoring*, where the players can observe the outcome of each transition and the vertex in which the game is, but cannot observe the actions taken by the other players [21]. In Remark 2.1 we elaborate on the difference between the two models. As we show in the paper, the results for reachability, co-Büchi, and Büchi stay valid also for our full-information model. For the parity condition, however, our model simplifies the setting and the problem of deciding the existence of an NE is NP-complete, as opposed to $\mathrm{P}_{\|}^{\mathrm{NP}}$ in the nondeterministic model with imperfect monitoring.

We introduce and study *repair* of multi-player games. We consider a setting with an authority (the designer) that aims to stabilize the interaction among the components and to increase the social welfare. In standard reactive synthesis [19], there are various ways to cope with situations when a specification is not realizable. Obviously, the specification has to be weakened, and this can be done either by adding assumptions on the behavior of the environment, fairness included, or by giving up some of the requirements on the system [6, 15]. In our setting, where the goal is to obtain stability, and the game is not zero-sum, a repair may both weaken and strengthen the specifications, which, in our main model, is modeled by modifications to the winning conditions.

The input to the *specification-repair problem* (SR problem, for short) is a game along with a *cost function*, describing the cost of each repair. For example, in Büchi games the cost function specifies, for each vertex v and player i, the cost of making v accepting for Player i and the cost of making v rejecting. The cost may be 0, reflecting the fact that v is accepting or rejecting in the original specification of Player i, or it may be ∞, reflecting the fact that

[1] A game may also involve incomplete information or stochastic transitions or strategies. The setting we consider here is not stochastic and players have full observability on the other players actions.

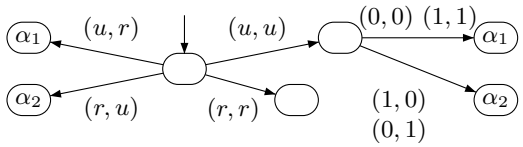


Figure 1 File sharing game. Initially, each player chooses to request either from the other user (action u) or from the repository (action r). In case both players choose u, the XOR game is played. The objective of Player i is to reach a vertex labeled α_i, in which case the other player sends her the file.

the original classification of v is a feature of the specification that the designer is not allowed to modify. We consider some useful classes of cost functions, like *uniform costs* – where all assignments cost 1, except for one that has cost 0 and stands for the original classification of the vertex, or *don't-care costs* – where several assignments have cost 0, reflecting a don't-care original classification, and all other assignments have cost ∞. In reachability, Büchi, and co-Büchi games, we also refer to one-way costs, where repair may only add or only remove vertices from the set of accepting vertices.

The goal of the designer is to suggest a repair to the winning conditions with which the game has an NE. One way to quantify the quality of a repair is its cost, and indeed the problem also gets as input a bound on the budget that can be used in the repairs. Another way, which has to do with the social welfare, considers the specifications that are satisfied in the obtained NE. Specifically, in the *rewarded specification-repair problem* (RSR problem, for short), the input also includes a *reward function* that maps subsets of specifications to rewards. When the suggested repair leads to an NE with a set W of winners, the designer gets a reward that corresponds to the specifications of the players in W. The quality of a solution then refers both to the budget it requires and to its reward. In particular, a reward function may prioritize the players and give a reward only to one player. Then, the question of finding an NE is similar to that of *rational synthesis*, where a winning strategy for the system can take into an account the objectives of the players that constitute the environment [9]. Thus, a special case of our contribution is repair of specifications in rational synthesis.

In [4], Brenguier describes several examples in which concurrent games and their stability model real-life scenarios. This includes peer-to-peer networks, wireless channel with a shared access, shared file systems, and more. The examples there also demonstrate the practicality of specification repair in these scenarios. We give an explicit example below.

▶ **Example 1.** Consider a file-sharing system serving two users. Each user requests a file from the other user or from a repository. Accessing the repository takes longer than transmitting between users, but the connection between the users can be used only in one direction at a time. If both users request the file from each other, they each choose a bit, and the file is transmitted to one of them according to the XOR of the bits. We model the interaction between the players as a reachability game, depicted in Fig. 1.

Observe that the game has no NE. Indeed, if w.l.o.g Player 1 does not reach α_1, then either Player 2 chose u and Player 1 lost in the XOR game, in which case Player 1 can deviate by choosing a different bit in the XOR, or Player 2 chose r, in which case Player 1 can deviate by playing u.

There are several ways to repair the game such that it has an NE. One is to break the symmetry between the players and make the vertex reached by playing (u,r) accepting for both players, and similarly for the vertex reached by playing (r,u). The cost involved in

this repair corresponds to the cost of communicating with the slower repository, and it is particularly useful when the reward function gives a priority to one of the players. Another possibility is to make the vertex reachable by playing (r, r) accepting for both players. Again, this involves a cost. ◀

Studying the SR and RSR problems, we distinguish between several classes, characterized by the type of winning conditions, cost functions, and reward functions. From a complexity point of view, we also distinguish between the case where the number of players is arbitrary and the one where it is constant. Recall that the problem of deciding whether an NE exists with an arbitrary number of players is NP-complete for reachability, co-Büchi, and parity games and can be solved in polynomial time for Büchi games. It is not too hard to lift the NP lower bound to the SR and RSR problems. The main challenge is the Büchi case, where one should find the cases where the polynomial complexity of deciding whether an NE exists can be lifted to the SR and RSR problems, and the cases where the need to find a repair shifts the complexity of the problem to NP. We show that the polynomial complexity can be maintained for don't-care costs, but the other settings are NP-complete. Our lower bounds make use of the fact that the unilateral change of a strategy that is examined in an NE can be linked to a change of the XOR of votes of all players, thus a single player can control the target of such transitions in a concurrent game.[2] We continue to study a setting with an arbitrary number of players. We check whether fixing the number of players can reduce the complexity of the SR and RSR problems, either by analyzing the complexity of the algorithms for an arbitrary number of players, or by introducing new algorithms. We show that in many cases, we can solve the problem in polynomial time, mainly thanks to the fact that it is possible to go over all possible subsets of players in search for a subset that can win in an NE.

In the context of verification, researchers have studied also other types of repairs (c.f., [11]). After focusing on the SR and RSR problems, we turn to study two other repair models. The first is *transition-repair*, in which a repair amounts to redirecting some of the transitions in the game. As with the SR problem, each redirection has a cost, and we seek repairs of minimal cost that would guarantee the existence of an NE. The transition-repair model is suitable in settings where the actions of the players do not induce a single successor state and we can choose between several alternatives. The second model we consider is that of *controlled-players*, in which we are allowed to dictate a strategy for some players. Also here, controlling players has a cost, and we want to minimize the cost and still guarantee the existence of an NE. We study several classes of the two types, and show that they are at least as difficult as specification repair.

Due to lack of space, most proofs can be found in the full version, in the authors' home pages.

2 Preliminaries

2.1 Concurrent games

A *concurrent game* is a tuple $\mathcal{G} = \langle \Omega, V, A, v_0, \delta, \{\alpha_i\}_{i\in\Omega}\rangle$, where Ω is a set of k players; V is a set of vertices; A is a set of actions, partitioned into sets A_i of actions for Player i, for

[2] We note that while the representation of our games is big, as the transition function specifies all vectors of actions, our complexity results hold also for games with a succinct representation of the transition function, in particular games with an arbitrary number of players in which only a constant number of players proceed in each vertex. The complexity of finding NE in succinctly represented games was studied in [10]. Succinctly represented games were studied in [16] the context of ATL model checking.

$i \in \Omega$; $v_0 \in V$ is an initial vertex; $\delta : V \times A_1 \times A_2 \times \cdots \times A_k \to V$ is a transition function, mapping a vertex and actions taken by the players to a successor vertex; and α_i, for $i \in \Omega$, specifies the objective for Player i. We describe several types of objectives in the sequel. For $v, v' \in V$ and $\overline{a} \in A_1 \times A_2 \times \cdots \times A_k$ with $\delta(v, \overline{a}) = v'$, we sometimes refer to $\langle v, \overline{a}, v' \rangle$ as a transition in $\mathcal{G}$.

A *strategy* for Player i is a function $\pi_i : (A_1 \times ... \times A_k)^* \to A_i$, which directs Player i which action to take, given the history of the game so far. Note that the history is given by means of the sequence of actions taken by all players so far.[3]

A *profile* is a tuple $P = \langle \pi_1, ..., \pi_k \rangle$ of strategies, one for each player. The profile P induces a sequence $\overline{a}_0, \overline{a}_1, \ldots \in (A_1 \times ... \times A_k)^\omega$ as follows: $\overline{a}_0 = \langle \pi_1(\epsilon), ..., \pi_k(\epsilon) \rangle$ and for every $i > 0$ we have $\overline{a}_i = \langle \pi_1(\overline{a}_0, ..., \overline{a}_{i-1}), ..., \pi_k(\overline{a}_0, ..., \overline{a}_{i-1}) \rangle$. For a profile P we define its *outcome* $\tau = \mathit{outcome}(P) \in V^\omega$ to be the path of vertices in $\mathcal{G}$ that is taken when all the players follow their strategies in P. Formally, $\tau = v_0, v_1, ...$ starts in v_0 and proceeds according to δ, thus $v_{i+1} = \delta(v_i, \overline{a}_i)$. The set of *winners* in P, denoted $W(P) \subseteq \Omega$, is the set of players whose objective is satisfied in $\mathit{outcome}(P)$. The set of *losers* in P, denote $L(P)$, is then $\Omega \setminus W(P)$, namely the set of players whose objective is not satisfied in $\mathit{outcome}(P)$.

A profile $P = \langle \pi_1, ..., \pi_k \rangle$ is a *Nash equilibrium* (NE, for short) if, intuitively, no (single) player can benefit from unilaterally changing her strategy. In the general setting, the outcome of P associates a reward with each of the players, thus "benefiting" stands for increasing the reward. In our setting here, the objective of Player i is binary – either α_i is satisfied or not. Accordingly, "benefiting" amounts to moving from the set of losers to the set of winners. Formally, for $i \in \Omega$ and some strategy π'_i for Player i, let $P[i \leftarrow \pi'_i] = \langle \pi_1, \ldots, \pi_{i-1}, \pi'_i, \pi_{i+1}, ..., \pi_k \rangle$ be the profile in which Player i *deviates* to the strategy π'_i. We say that P is an NE if for every $i \in \Omega$, if $i \in L(P)$, then for every strategy π'_i we have $i \in L(P[i \leftarrow \pi'_i])$.

We consider the following types of objectives. Let $\tau \in V^\omega$ be an infinite path.

- In *reachability* games, $\alpha_i \subseteq V$, and τ satisfies α_i if τ reaches α_i.
- In *Büchi* games, $\alpha_i \subseteq V$, and τ satisfies α_i if τ visits α_i infinitely often.
- In *co-Büchi* games, $\alpha_i \subseteq V$, and τ satisfies α_i if τ visits $V \setminus \alpha_i$ only finitely often.
- In *parity* games, $\alpha_i : V \to \{1, ..., d\}$, for the *index* d of the game, and τ satisfies α_i if the maximal rank that is visited by τ infinitely often is even. Formally, let $\tau = v_0, v_1, ...$, then τ satisfies α_i if $\max\{j \in \{1, ..., d\} : \alpha_i(v_l) = j \text{ for infinitely many } l \geq 0\}$ is even.

Note that Büchi and co-Büchi games are special cases of parity games, with ranks $\{1, 2\}$ and $\{2, 3\}$, respectively. We sometimes refer to a winning condition $\alpha_i \subseteq V$ also as a function $\alpha_i : V \to \{\top, \bot\}$, with $\alpha_i(v) = \top$ iff $v \in \alpha_i$.

▶ Remark. Our definition of strategy is based on the history of actions played. This is different from the setting in [4], where strategies are based on the history of visited vertices. Our setting reflects the fact that players have full knowledge of the actions played by other players, and not only the outcome of these actions. As we now demonstrate, our setting is different as it enables the players to make use of this full knowledge to obtain an NE. In Section 2.4 we elaborate on the algorithmic differences between the settings.

Consider the concurrent three-player Büchi game $\mathcal{G} = \langle \Omega, V, A, v_0, \delta, \{\alpha_i\}_{i \in \Omega} \rangle$, where $\Omega = \{1, 2, 3\}$, $V = \{v_0, v_1, a, b, c\}$, $A_i = \{0, 1\}$ for $i \in \Omega$, $\alpha_1 = \{a\}, \alpha_2 = \{b\}, \alpha_3 = \{c\}$, and the transition function is as follows. In v_0, if 1 and 2 play $(0, 0)$, the game moves to c, and

[3] Note that strategies observe the history of actions, rather than the history of vertices. In Remark 2.1 we elaborate on this aspect.

otherwise to v_1. In v_1, Player 3 can choose to go to a or to b. The vertices a, b, and c are sinks.

There is an NE in $\mathcal{G}$, whose outcome is the path v_0, c^ω. That is, players 1 and 2 play $(0, 0)$. However, in order for this to be an NE profile, Player 3 needs to be able to "punish" either Player 1 or Player 2 if they deviate to v_1. For that, Player 3 needs to know the action that leads to v_1: if Player 1 deviates, then Player 3 chooses to proceed to b, and if Player 2 deviates, then Player 3 chooses to proceed to a. If we consider strategies that refer to histories of vertices, then there is no NE in the game. ◀

2.2 Partial games with costs and rewards

Let $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$. A *partial concurrent parity game* $\mathcal{G}$ is a concurrent parity game in which the winning conditions are replaced by a cost function that describes the cost of augmenting $\mathcal{G}$ with different winning conditions. Formally, $\mathcal{G} = \langle \Omega, V, A, v_0, \delta, \mathit{cost} \rangle$, where the cost function $\mathit{cost} : V \times \Omega \times \{1, ..., d\} \to \mathbb{N}_\infty$ states, for each vertex $v \in V$, player $i \in \Omega$, and rank $j \in \{1, ..., d\}$, what the cost of setting $\alpha_i(v)$ to be j. We can think of a concrete game (one with fully specified winning conditions α_i, for $i \in \Omega$) as a partial game in which the cost function is such that $\mathit{cost}(v, i, j) = 0$ if $\alpha_i(v) = j$ and $\mathit{cost}(v, i, j) = \infty$ otherwise. Intuitively, leaving $\alpha_i(v)$ as specified is free of charge, and changing $\alpha_i(v)$ is impossible, as it costs ∞. Partial games enable us to model settings where a designer can play with the definition of the winning conditions, subject to some cost function and a given budget.

Consider a partial parity game $\mathcal{G}$. A *winning-condition assignment* for $\mathcal{G}$ is $f : V \times \Omega \to \{1, ..., d\}$. The parity game induced by $\mathcal{G}$ and f, denoted $\mathcal{G}^f$, has $\alpha_i(v) = f(v, i)$, for all $v \in V$ and $i \in \Omega$. The *cost* of f is $\mathit{cost}(f) = \sum_{i \in \Omega} \sum_{v \in V} \mathit{cost}(v, i, f(v, i))$.

In the case of reachability, Büchi, and co-Büchi, the cost function is $\mathit{cost} : V \times \Omega \times \{\top, \bot\} \to \mathbb{N}_\infty$, and the winning-condition assignment is of the form $f : V \times \Omega \to \{\top, \bot\}$.

Consider a partial game $\mathcal{G}$ and a cost function *cost*. For every player $i \in \Omega$ and vertex $v \in V$, we define the set $\mathit{free}_{\mathit{cost}}(v, i) \subseteq \{1, ..., d\}$ as the set of ranks we can assign to $\alpha_i(v)$ free of charge. Formally, $\mathit{free}_{\mathit{cost}}(v, i) = \{j : \mathit{cost}(v, i, j) = 0\}$. We consider the following two classes of cost functions in parity games.

- *Uniform costs:* For every $i \in \Omega$ and $v \in V$, we have $|\mathit{free}_{\mathit{cost}}(v, i)| = 1$ and for every $j \notin \mathit{free}_{\mathit{cost}}(v, i)$, we have $\mathit{cost}(v, i, j) = 1$. Thus, a partial game with a uniform cost function corresponds to a concrete game in which we can modify the winning condition with a uniform cost of 1 for each modification.
- *Don't cares:* For every $i \in \Omega$, $v \in V$, and $j \in \{1, ..., d\}$, we have $\mathit{cost}(v, i, j) \in \{0, \infty\}$, and $|\mathit{free}_{\mathit{cost}}(v, i)| \geq 1$. Thus, as in concrete games, we cannot modify the rank of vertices that are not in $\mathit{free}_{\mathit{cost}}(v, i)$, but unlike concrete games, here $\mathit{free}_{\mathit{cost}}(v, i)$ need not be a singleton, reflecting a situation with "don't cares", where a designer can choose among several possible ranks free of charge.

For the special case of reachability, Büchi, and co-Büchi games, we also consider the following classes.

- *Negative one-way costs:* For every $i \in \Omega$ and $v \in V$, either $\mathit{cost}(v, i, \top) = 0$ and $\mathit{cost}(v, i, \bot) = 1$, or $\mathit{cost}(v, i, \bot) = 0$ and $\mathit{cost}(v, i, \top) = \infty$. Intuitively, we are allowed only to modify $\top$ vertices to $\bot$ ones, thus we are only allowed to make satisfaction harder by removing vertices from α_i.
- *Positive One-way costs:* For every $i \in \Omega$ and $v \in V$, either $\mathit{cost}(v, i, \top) = 0$ and $\mathit{cost}(v, i, \bot) = \infty$, or $\mathit{cost}(v, i, \bot) = 0$ and $\mathit{cost}(v, i, \top) = 1$. Intuitively, we are allowed only to modify $\bot$ vertices to $\top$ ones, thus we are only allowed to make satisfaction easier by adding vertices to α_i.

Reward function. Consider a game $\mathcal{G}$. A *reward function* for $\mathcal{G}$ is $\zeta : 2^\Omega \to \mathbb{N}$. Intuitively, if the players follow a profile P of strategies, then the reward to the designer is $\zeta(W(P))$. Thus, a designer has an incentive to suggest to the players a stable profile of strategies that maximizes her reward. We assume that ζ is monotone w.r.t. containment.

2.3 The specification-repair problem

Given a partial game $\mathcal{G}$ and a threshold $p \in \mathbb{N}$, the *specification-repair problem* (SR problem, for short) is to find a winning-condition assignment f such that $cost(f) \leq p$ and $\mathcal{G}^f$ has an NE. Thus, we are willing to invest at most p in order to be able to suggest to the players a stable profile of strategies.

In the *rewarded specification-repair problem* (RSR problem, for short) we are also given a reward function ζ and a threshold q, and the goal is to find a winning-condition assignment f such that $cost(f) \leq p$ and $\mathcal{G}^f$ has an NE with a winning set of players W for which $\zeta(W) \geq q$.

▶ Remark. An alternative definition to the RSR problem would have required all NEs in $\mathcal{G}^f$ to have a reward greater than q. This is similar to the cooperative vs. non-cooperative definitions of rational synthesis [9, 14]. In the cooperative setting, which we follow here, we assume that the authority can suggest a profile of strategies to the players, and if this profile is an NE, then they would follow it. In the non-cooperative one, the authority cannot count on the players to follow its suggested profile even if it is an NE. We find the cooperative setting more realistic, especially in the context of repairs, which assumes rational cooperative agents (indeed, they are willing to apply a repair for a cost). Moreover, all existing work in Algorithmic Game Theory follow the cooperative setting in games that are similar to the ones we study.

Also, rather than including in the input to the RSR problem two thresholds, one could require that $\zeta(W) \geq cost(f)$ or to compare $\zeta(W)$ with $cost(f)$ in some other way. Our results hold also for such definitions. ◀

We distinguish between several classes of the SR and RSR problems, characterized by the type of winning conditions, cost function, and reward function. From a complexity point of view, we also distinguish between the case where the number of players is arbitrary and the one where it is constant.

▶ Remark. Another complexity issue has to do with the size of the representation of the game. Recall that, specifying $\mathcal{G}$, we need to specify the transition $\delta(v, \overline{a})$ for every vertex $v \in V$ and action vector $\overline{a} \in A^{|\Omega|}$. Thus, the description of $\mathcal{G}$ is exponential in the size of Ω. While this may make the lower bounds more challenging, it may also makes polynomial upper bounds easy. In Remark 3.1 we argue that our complexity results hold also in a settings with a succinct representation of $\mathcal{G}$. For example, when $\mathcal{G}$ is *c-concurrent* for some $c \geq 1$, meaning that in each vertex, only c players *control* the vertex. That is, in each vertex only c players choose actions and determine the successor vertex. Then, the size of δ is bounded by $|V \times A^c|$, for a constant c. ◀

2.4 Deciding the existence of an NE

The problem of deciding the existence of an NE, which is strongly related to the SR problem was studied in [4]. The model there subsumes our model. First, as discussed in Remark 2.1, our strategies have full knowledge of actions, whereas the strategies in [4] only observe vertices. Second, the transition function in [4] is *nondeterministic*, thus a vertex and a vector of actions are mapped to a set of possible successors. We can efficiently convert a game in

our model into an "equivalent" game in the model of [4] (in the sense that the existence of a NE is preserved). Thus, algorithmic upper bounds from [4] apply to our setting as well. Conversely, however, lower bounds from [4] do not apply to our model, and indeed the lower bounds we show differ from those of [4].

Specifically, it is shown in [4, 3] that the problem of deciding whether a given game has an NE is $\mathrm{P}_{\|}^{\mathrm{NP}}$-complete for parity objectives; that is, it can be solved in polynomial time with parallel queries to an NP oracle. The problem is NP-complete for reachability and co-Büchi objectives, and can be solved in polynomial time for Büchi games. We show that in our model, while the complexity of the problem for reachability, Büchi, and co-Büchi objectives coincides with that of [4], the complexity for parity objectives is NP-complete. In Section 3.1 we present Theorem 5, which entails an explicit algorithm for deciding the existence of an NE in Büchi games. Our algorithm is significantly simpler than the one in [4] as it considers a deterministic model.

Additionally, we emphasize that the main contribution of this work is the introduction of repairs, and our choice of model is in part for its clarity. Indeed, repair can similarly be defined in the model of [4], as partial observation is an orthogonal notion.

In the full version we prove the following theorem.

▶ **Theorem 2.** *The problem of deciding whether a concurrent reachability, co-Büchi, or parity game has an NE is NP-complete.*

In particular, we note that the problem of *verifying* the existence of an NE can be solved in polynomial time, using an appropriate witness. See the full version for details.

3 Solving the SR and RSR Problems

3.1 An Arbitrary Number of Players

In this section we consider the SR problem for an arbitrary number of players. Recall that the problem of deciding whether an NE exists is NP-hard for reachability, co-Büchi, and parity games and is in P for Büchi games. It is not too hard to lift the NP lower bound to the SR problem. The main challenge is the Büchi case, where one should find the cases where the polynomial complexity of deciding whether an NE exists can be lifted to the SR problem, and the cases where the need to find a repair shifts the complexity of the problem to NP.

▶ **Theorem 3.** *The SR problem for reachability, co-Büchi, and parity games with uniform, don't cares, positive one-way, or negative one-way costs is NP-complete.*

Proof. Membership in NP is easy, as given a game $\mathcal{G}$, a cost function *cost*, and a threshold p, we can guess a winning-condition assignment f, and then proceed to nondeterministically check whether there exists an NE in $\mathcal{G}^f$ as described in Section 2.4.

For the lower bound, we describe a reduction from the problem of deciding whether an NE exists in a given co-Büchi or reachability game is NP-complete, proved to be NP-hard in Theorem 2.

Consider a game $\mathcal{G}$, and let *cost* be the cost function induced naturally by it. That is, for every $v \in V$ and $i \in \Omega$, we have $cost(v, i, \alpha_i(v)) = 0$, and the rest of the cost function is defined to involve a positive cost and respect the definition of uniform, don't cares, positive one-way, or negative one-way cost function. With this cost function, the only assignment with cost 0 is such that $f(v, i) = \alpha_i(v)$ for every $i \in \Omega$ and $v \in V$. Thus, $\mathcal{G}$ has an NE iff there is a winning-condition assignment f such that $cost(f) \leq 0$ and $\mathcal{G}^f$ has an NE. ◀

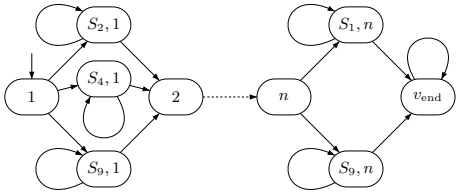


Figure 2 Reduction in the uniform costs setting in Theorem 4. Here, $1 \in S_2 \cap S_4 \cap S_9$, and $n \in S_1 \cap S_9$.

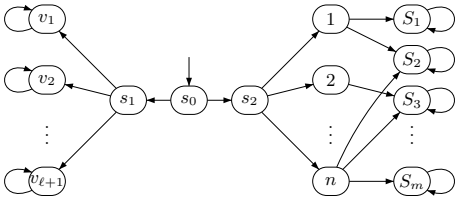


Figure 3 Reduction of the negative one-way costs setting in Theorem 4. Here, $1 \in S_1 \cap S_2$, $2 \in S_3$ and $n \in S_2 \cap S_3 \cap S_m$.

We turn to Büchi games, where the goal is to find the cases where the polynomial complexity of deciding the existence of an NE can be maintained. We start with the negative cases.

▶ **Theorem 4.** *The SR problem for Büchi games with uniform, positive one-way, or negative one-way costs is NP-complete.*

Proof. Membership in NP is easy, as given a game $\mathcal{G}$, a cost function *cost*, and a threshold p, we can guess a winning-condition assignment f, and then check in polynomial time whether there exists an NE in $\mathcal{G}^f$ [4]. For the lower bounds we describe reductions from SET-COVER, which is well known to be NP-complete [12]. We bring its definition here for completeness. Consider a set $U = \{1, \ldots, n\}$ of elements and a set $S = \{S_1, ..., S_m\}$ of subsets of U, thus $S_i \subseteq U$ for every $1 \leq j \leq m$. A set-cover of size ℓ is $\{S_{j_1}, ..., S_{j_\ell}\} \subseteq S$ such that for every $i \in U$ there exists $1 \leq l \leq \ell$ such that $i \in S_{j_l}$. The SET-COVER problem is to decide, given U, S, and ℓ, whether there exists a set-cover of size ℓ. We assume w.l.o.g that $\ell < \min\{n, m\}$.

Uniform costs. Consider an input $\langle U, S, \ell\rangle$ for SET-COVER. We construct a partial concurrent game $\mathcal{G} = \langle \Omega, V, A, v_0, \delta, \mathit{cost}\rangle$ such that there is a set cover of U of size ℓ iff there exists a winning-condition assignment f with $\mathit{cost}(f) \leq \ell$ such that $\mathcal{G}^f$ has an NE.

The players in $\mathcal{G}$ are $\Omega = U \cup S$. That is, there is one player, referred to as Player i, for every $i \in U$, and one player, referred to as Player S_j, for every $S_j \in S$. The set of vertices in $\mathcal{G}$ is $V = U \cup \{\langle S_j, i\rangle : i \in S_j \in S\} \cup \{v_{\text{end}}\}$. The initial vertex is $1 \in U$. We now describe the transitions and actions (see Fig. 2).

At vertex $i \in U$, Player i alone has control, in the sense that only her action is taken into an account in deciding the successor. Player i can choose to move to a vertex $\langle S_j, i\rangle$ for which $i \in S_j$. At vertex $\langle S_j, i\rangle$, all players in $U \cup \{S_j\}$ have control on the choice of the successor vertex and can choose either to stay at $\langle S_j, i\rangle$, or to proceed, either to vertex $i + 1$, if $i < n$, or to v_{end}, if $i = n$. This choice is made as follows. The actions of the players are $\{0, 1\}$, and the transition depends on the XOR of the actions. If the XOR is 0, then the game stays in $\langle S_j, i\rangle$ and if the XOR is 1, the game proceeds to $i + 1$ or to v_{end}. Finally, v_{end} has a self loop.

We now describe the cost function. Intuitively, we define *cost* so that the default for v_{end} is to be accepting for all players $i \in U$ and rejecting for all players $S_j \in S$. Thus, $\mathit{cost}(v_{\text{end}}, i, \top) = 0$ for all $i \in U$ and $\mathit{cost}(v_{\text{end}}, S_j, \bot) = 0$ for all $S_j \in S$. Also, for every $S_j \in S$ and $i \in S_j$, we have $\mathit{cost}(\langle S_j, i\rangle, S_j, \top) = 0$ and $\mathit{cost}(\langle S_j, i\rangle, i, \bot) = 0$. Thus, $\langle S_j, i\rangle$ is accepting for Player S_j and is rejecting for Player i. All other costs are set to 1, as required by a uniform cost.

We claim that $\langle U, S, \ell\rangle \in$ SET-COVER iff $\mathcal{G}$ has a winning-condition assignment f with cost at most ℓ such that $\mathcal{G}^f$ has an NE. In the full version we formally prove the correctness of the reduction. Intuitively, every assignment f of cost at most ℓ must set $f(v_{\text{end}}, i) = \top$

and $f(v,i) = \bot$ for $v \neq v_{\text{end}}$, for some $i \in U$. Thus, an NE must end in v_{end}, as otherwise Player i uses the XOR transitions in order to deviate to a strategy whose outcome reaches v_{end}. Hence, an assignment must set $f(v_{\text{end}}, S_j) = \top$ for at most ℓ players $S_{j_1}, ..., S_{j_\ell}$, such that it is possible to get from 1 to v_{end} by going only through vertices $\langle S_{j_k}, i\rangle$ for $1 \leq k \leq \ell$. These ℓ players induce a set cover. The other direction is easy.

Positive one-way costs. In the correctness proof of the reduction above we show that in fact, the only assignments that need to be considered are positive one-way. Thus, the same reduction, in fact with a simpler correctness argument, can be used to show NP-hardness of the setting with positive one-way costs.

Negative one-way costs. Finally, we consider the setting of negative one-way costs. Again, we describe a reduction from SET-COVER. Consider an input $\langle U, S, \ell\rangle$ for SET-COVER. We construct a partial two-player game $\mathcal{G} = \langle \Omega, V, A, v_0, \delta, \mathit{cost}\rangle$ such that there is a set cover of U of size ℓ iff there exists a winning-condition assignment f with $\mathit{cost}(f) \leq \ell$ such that $\mathcal{G}^f$ has an NE. The game $\mathcal{G}$ is constructed as follows. The players are $\Omega = \{1, 2\}$. The vertices are $V = U \cup S \cup \{s_0, s_1, s_2\} \cup \{v_1, ..., v_{\ell+1}\}$. The game starts in s_0, where the actions for the players are $\{0, 1\}$. If the XOR of the actions is 0, the game moves to vertex s_1, where Player 1 chooses a vertex from $v_1, ..., v_{\ell+1}$, all of which have self loops. We set $\mathit{cost}(v_i, 1, \top) = 0$ for $1 \leq i \leq \ell + 1$. Intuitively, if the game proceeds to s_1, then Player 1 can choose a winning vertex, and the play gets stuck there. If the XOR in s_0 was 1, the game proceeds to vertex s_2 from which player 2 chooses a vertex $i \in U$. In vertex i, player 1 chooses a vertex S_j such that $i \in S_j$. For every $1 \leq j \leq m$, the vertex S_j has only a self loop. We set $\mathit{cost}(S_j, 2, \top) = 0$ for $1 \leq j \leq m$. Intuitively, if the game proceeds to s_2, then Player 2 "challenges" Player 1 with a value $i \in U$, and Player 1 has to respond with some set S_j such that $i \in S_j$, and then the play gets stuck in S_j. See Fig. 3 for an illustration.

The rest of the *cost* function is set to give $\bot$ cost 0, and is completed to be a negative one-way cost. That is, we set $\mathit{cost}(v_j, 2, \bot) = 0$ for every $1 \leq j \leq \ell + 1$, $\mathit{cost}(S_j, 2, \bot) = 0$ for every $1 \leq j \leq m$, and $\mathit{cost}(x, 1, \bot) = \mathit{cost}(x, 2, \bot) = 0$ for $x \in U \cup \{s_0, s_1, s_2\}$. Finally, $\mathit{cost}(v, i, \bot) = 1$ if $\mathit{cost}(v, i, \top) = 0$ and $\mathit{cost}(v, i, \top) = \infty$ if $\mathit{cost}(v, i, \bot) = 0$, for every $i \in \{1, 2\}$ and $v \in V$, as per the definition of a negative one-way cost.

In the full version we formally prove the correctness of the reduction. Intuitively, every assignment f of cost at most ℓ must set $f(v_i, 1) = \top$ for some $1 \leq i \leq \ell + 1$. Thus, Player 1 is guaranteed to be able to deviate and win in any profile. In order to have an NE, we must be able to set $f(S_j, 2) = \bot$ for at most ℓ vertices $S_{j_1}, ..., S_{j_\ell}$, such that for every $i \in U$ that Player 2 chooses, there exists $1 \leq k \leq \ell$ such that $i \in S_{j_k}$, and so there is a set-cover. The other direction is again, easy. ◀

We now turn to consider the positive case, where the polynomial complexity of deciding whether an NE exists can be lifted to the SR problem.

▶ **Theorem 5.** *The SR problem for Büchi games and don't-cares can be solved in polynomial time.*

Proof. Consider a partial Büchi game $\mathcal{G} = \langle \Omega, V, A, v_0, \delta, \mathit{cost}\rangle$ with don't-cares. For every $i \in \Omega$, the set of vertices V can be partitioned into three sets:

1. The set $F_i = \{v : \mathit{cost}(v, i, \top) = 0 \wedge \mathit{cost}(v, i, \bot) = \infty\}$, of *accepting* vertices.
2. The set $R_i = \{v : \mathit{cost}(v, i, \bot) = 0 \wedge \mathit{cost}(v, i, \top) = \infty\}$, of *rejecting* vertices.
3. The set $DC_i = \{v : \mathit{cost}(v, i, \bot) = \mathit{cost}(v, i, \top) = 0\}$, of *don't-care* vertices.

The SR problem then amounts to deciding whether there is an assignment $f : \bigcup_{i\in\Omega} DC_i \to \{\top, \bot\}$ such that $\mathcal{G}^f$ has an NE. Note the cost of every such assignment is 0.

For a set $S \subseteq V$, let $W_S \subseteq \Omega$ be the set of *potential winners* in S: players that either have an accepting or don't-care vertex in S. Formally, $W_S = \{i \in \Omega : (F_i \cup DC_i) \cap S \neq \emptyset\}$. The set of *losers* in S is then $L_S = \Omega \setminus W_S$, thus $i \in L_S$ iff $S \subseteq R_i$.

We describe the intuition behind our algorithm. An outcome of a profile is an infinite path in $\mathcal{G}$, which gets stuck in a SCC S. We distinguish between the case S is an ergodic SCC – one that has no outgoing edges to other SCCs in $\mathcal{G}$, and the case S is not ergodic. Our algorithm tries to find a *witness* ergodic SCC S: one for which there is an assignment f such that $\mathcal{G}^f$ has an NE whose outcome gets stuck in S. When an ergodic SCC cannot serve as a witness, it is removed from $\mathcal{G}$ along with transitions that guarantee the soundness of such a removal, and the search for a witness ergodic SCC in the new game continues. When all SCCs are removed, the algorithm concludes that no NE exists.

In order to examine whether an ergodic SCC S can serve as a witness, the algorithm checks whether the players in W_S can force the game to reach S. Once the game reaches S, every outcome would not satisfy the objective of the players in L_S. Moreover, consider the assignment f that sets, for $i \in W_S$, every vertex in DC_i to $\top$. The profile whose outcome visits all the vertices in S is an NE in G^f. Checking whether the players W_S can force the game to reach S is not straightforward, as it should take into an account possible collaboration from players in L_S that are doomed to lose anyway and thus have no incentive to deviate from a strategy in which they collaborate with the players in W_S.

Formalizing this intuition involves the following definitions. Consider a player $i \in \Omega$. We define the *game against* i to be a two-player zero-sum concurrent game, where the players are Player i and the coalition $\Omega \setminus \{i\}$. The game is played on $\mathcal{G}$, where the objective of player i is α_i, and the objective of $\Omega \setminus \{i\}$ is to prevent i from satisfying α_i. The *cage* for player i is the set of vertices $C_i \subseteq V$ that consists of all vertices from which the coalition wins the game against i.

Deciding whether a vertex v is in C_i amounts to solving a two-player zero-sum concurrent game. These games can be solved in polynomial time for reachability, Büchi, and co-Büchi objectives [8].

Next, consider a transition $t = \langle v, \overline{a}, v' \rangle$ with $v, v' \in C_i$, thus $\delta(v, \overline{a}) = v'$. We say that t is *doomed for Player* i if Player i cannot alter her action in $\overline{a}$ and escape the cage C_i. We denote by doomed(B) the set of transitions that are doomed for all players in B.

For a game with don't-cares, whenever we consider the game against i, we refer to the concrete two-player games obtained from $\mathcal{G}$ with the assignment that assigns $\bot$ to the vertices in DC_i.

Our algorithm checks whether there is a path τ to S that traverses only transitions in doomed(L_S). If so, it concludes that $\mathcal{G}$ can be repaired to have an NE with the assignment f that is defined as follows. For every $v \in V$, for $j \in L_S$, we have $f(v, j) = \bot$ and, for $j \in W_S$, we have $f(v, j) = \top$. Indeed, the profile whose outcome is τ followed by a path that visits all the vertices in S infinitely often, and which punishes a player that deviates from her expected action in τ is an NE in $\mathcal{G}^f$. ◀

▶ Remark. As discussed in Remark 2.3, our results stay valid when the games are c-concurrent for a constant $c \geq 2$. In particular, the running time of the algorithm described in Theorem 5 is polynomial in the representation size of $\mathcal{G}$. As for lower bounds, the second reduction described in the proof of Theorem 4 generates a game with only two players. In addition, the first reduction there can be slightly modified to capture 2-concurrent games. For that, we replace the vertices $S \times U$ in $\mathcal{G}$ by a cycle of n vertices, where $\langle S_j, i \rangle$ is the first vertex in the cycle $\langle S_j, i \rangle_1, ..., \langle S_j, i \rangle_n$. The players that control the l-th vertex, for $1 \leq l \leq n$, are S_j and l. Both players have two possible actions $\{0, 1\}$. If the XOR of their choices is 0, the

Table 1 Complexity results for the setting with a constant number of players.

Problem ╲ Game	Büchi	co-Büchi	Reachability	Parity
NE Existence	P [3]	P	P [2]	NP∩coNP
Uniform	P	NP-C	P	NP-C
Don't care	P	P	P	NP∩coNP
Negative One-way	NP-C			–
Positive One-way	P	P	P	–

game continues to $(l+1) \mod n$, the next vertex in the cycle, and if it is 1, then the game exits the cycle and proceeds to vertex $i+1$. Clearly, each player in $U \cup \{S_j\}$ can force the game to stay in the gadget or exit it assuming the other players fix a strategy.

3.2 A Constant Number of Players

In this section, we consider the SR problem for a constant number of players. The algorithms presented in Section 3.1 can be clearly applied in this setting. For example, Theorem 5 implies that the SR problem for Büchi games with don't cares can be solved in polynomial time, and in particular this holds when the number of players is fixed. For NP-complete problems, however, the upper bounds in Section 3.1 only imply exponential time algorithms. In this section, we check whether fixing the number of players can reduce the complexity, either by analyzing the complexity of the algorithms from Section 3.1, or by introducing new algorithms.

The results are summarized in Table 1, and the proofs appear in the full version with the exception of Theorem 6 below.

▶ **Theorem 6.** *The SR problem for co-Büchi games with positive one-way costs and a constant number of players can be solved in polynomial time.*

Proof. We solve the problem by presenting a polynomial time algorithm for checking, given a game $\mathcal{G}$, a bound $p \in \mathbb{N}$ on the budget for the repair, and a set $W \subseteq \Omega$, whether there is a positive one-way assignment f with cost at most p, for which $\mathcal{G}^f$ has an NE profile P with $W \subseteq W(P)$. We then iterate over all subsets $W \subseteq \Omega$ to obtain a polynomial time algorithm.

Under the definitions used in the proof of Theorem 5, let $L = \Omega \setminus W$, and let $\mathcal{G}_W = \mathcal{G}|_{\text{doomed}(L)}$. Consider a vertex $v \in V$ that is reachable from v_0 in $\mathcal{G}_W$. We look for an assignment f for which there is a cycle that contains v and traverses only vertices in $\bigcap_{i \in W} \alpha_i^f$. Such a cycle satisfies the objectives of the players in W. In order to do so, we add weights to $\mathcal{G}_W$ as follows. The weight of an edge $\langle u, u' \rangle$ in $\mathcal{G}_W$ is the repair budget that is needed in order to make u' accepting for all players in W. That is, $\langle u, u' \rangle$ gets the weight $\sum_{i \in W} cost(u', i, \top)$. Then, we run Dijkstra's shortest-path algorithm from v to find the minimal-weight cycle that contains v. If the weight of the cycle is at most p, we return "yes". If there is no such cycle for every $v \in V$ and $W \subseteq \Omega$, we return "no". We then repeat this process for every $W \subseteq \Omega$.

In the full version we analyze the runtime and prove the correctness of the algorithm. ◀

3.3 Solving the RSR problem

Recall that in the RSR problem we are given, in addition to $\mathcal{G}$, *cost*, and $p \in \mathbb{N}$, a reward function $\zeta : 2^\Omega \to \mathbb{N}$ and a threshold q, and we need to decide whether we can repair $\mathcal{G}$ with cost at most p in a way that the set of winners W in the obtained NE is such that $\zeta(W) \geq q$.

In the full version we argue that the additional requirement about the reward maintains the complexity of the problem.

▶ **Theorem 7.** *The complexity of the SR and RSR problems coincide for all classes of objectives and cost functions, for both an arbitrary and a constant number of players.*

4 Other Types of Repairs

So far, we studied repairs that modify the winning conditions of the players. Other types of repairs can be considered. In this section, we examine two such types: *transition repair*, which modifies the transitions of the game, and *controlled-players repair*, where we can control (that is, force a strategy) on a subset of the players. The later is related to the *Stackelberg model*, which has been extensively studied in economics and more recently in Algorithmic Game Theory [13, 20], and in which some of the players are selfish whereas others are controllable.

4.1 Transition repair

In the *transition repair* model, we are allowed to redirect the transitions of a game. This is suitable in cases where a system is composed of several concurrent components, and we have some control on the flow of the entire composition. For example, consider a system in which several threads request a lock and granting a lock to a certain thread is modeled by a transition. Redirecting this transition can correspond to the lock being given to a different thread. Typically, not all repairs are possible, which is going to be modeled by an ∞ cost to impossible repairs. Finally, the games we study are sometimes obtained from LTL specifications of the players. Repairs in the winning conditions then have the flavor of switching between "until" and "weak-until" in the LTL specification. In this setting, one may find transition-repair to be more appropriate. First, it enables more elaborate changes in the specifications. Secondly, changes in the acceptance condition of the nondeterministic Büchi automata for the specifications induce transition changes in their deterministic parity automata, which compose the game.

In the full version we formalize this model, and define the *transition-repair problem* (TR problem, for short) similarly to the SR problem, with the goal being to find a cheap transition-repair that guarantees the existence of an NE. We prove the following results.

▶ **Theorem 8.** *The TR problem is NP-complete for the following cases:*

- *A constant number of players, for all objectives.*
- *Uniform costs with an arbitrary number of players, for all objectives.*
- *Uniform costs with a constant number of players, and co-Büchi and parity objectives.*

and can be solved in polynomial time for uniform costs with a constant number of players and Büchi and reachability objectives. The TR problem with uniform costs can be solved in polynomial time for Büchi and reachability objectives, and is NP complete

4.2 Controlled-players repair

The underlying assumption in game theory is that players are selfish and rational. In particular, they would follow a suggested strategy only if it is in their interest. In the *controlled-players repair* model, we assume that we can control some of the players and guarantee they would follow the strategy we assign them. The other players cooperate only if the profile is an NE. Controlling a player has a cost and our goal is to reach such a profile

with a minimal cost. This model is a type of Stackelberg model, where there is a *leader* player whose goal is to increase the social welfare. She moves first, selects a fraction α of the players, and assigns strategies to them. The rest of the players are selfish and choose strategies to maximize their revenue. Previous works in Algorithmic Game Theory study how the parameter α affects the social welfare in an NE. Clearly, when α is high, the social welfare increases.

Formally, given a game $\mathcal{G} = \langle \Omega, V, A, v_0, \delta, \{\alpha_i\}_{i\in\Omega}\rangle$ and a *control cost* function $cost : \Omega \to \mathbb{N}_\infty$, which maps each agent to the cost of controlling him, the *controlled-player repair* problem (the CR problem, for short), is to find a set of players of minimal cost such that if we are allowed to fully control these players, then the game has an NE. By *controlling* we mean that the players are not allowed to deviate from their strategies in the suggested profile.

Controlled-players repair arises in settings where an unstable system can be stabilized by restricting the environment, but this involves a cost. For example, controlling players is possible in settings where players accept an outside payment. As another example, taken from [20], the players are customers who can either pay a full price for using a system, and then their choices are unlimited, or they can pay a "bargain" price, and then their choices are limited, and hence their quality of service is not guaranteed. As a third example, consider a system that receives messages from the environment. We may want to require that messages arrive chronologically, otherwise our system is unstable. We can require this, but it involves a latency cost, and is effectively translated to asking the message dispatching thread to work in a non-optimal way, which is not the best strategy for the message dispatch server.

In the decision version of the problem, we are given a threshold p, and we need to determine if there exists a set $S \subseteq \Omega$ such that $cost(S) = \sum_{i\in S} cost(i) \leq p$ and controlling the players in S ensures the existence of an NE.

We start by studying the general case. In order to solve the CR problem, we observe that controlling Player i can be modeled by setting α_i to be the most permissive, thus for reachability, Büchi, and co-Büchi objectives, we set $\alpha_i = V$, and for parity objectives $\alpha_i(v)$ is the maximal even index. Indeed, if there is an NE profile P in $\mathcal{G}$ in which we control Player i, then P is also an NE when we set α_i as in the above (without controlling player i). Clearly, Player i has no incentive to deviate. Conversely, if there is an NE profile P after setting α_i to be the most permissive, then the same profile P is an NE in a game in which we control Player i and force it to play his strategy in P.

Theorem 9 below summarizes our results, and is proved in the full version.

▶ **Theorem 9.** *The CR problem is NP-complete for reachability, co-Büchi, and parity objectives, as well as for c-concurrent Büchi games, and is in P for general Büchi games, and for all objectives with a constant number of players.*

▶ Remark. In the future, we plan to investigate *scheduling repairs*, where a repair controls the set of players that proceed in a vertex, as well as *disabling repairs*, in which some actions of some players are disabled in some vertices.

References

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

2 P. Bouyer, R. Brenguier, and N. Markey. Nash equilibria for reachability objectives in multi-player timed games. In *Proc. 21st CONCUR*, pages 192–206, 2010.

3 P. Bouyer, R. Brenguier, N. Markey, and M. Ummels. Nash equilibria in concurrent games with Büchi objectives. In *Proc. 31st FSTTCS 2011*, pages 375–386, 2011.

4 R. Brenguier. Nash Equilibria in Concurrent Games – Application to Timed Games. PhD thesis, École normale supérieure, 2012.

5 K. Chatterjee, L. de Alfaro, and T.A. Henzinger. Qualitative concurrent parity games. *ACM Trans. Comput. Log.*, 12(4):28, 2011.

6 K. Chatterjee, T. Henzinger, and B. Jobstmann. Environment assumptions for synthesis. In *Proc. 19th CONCUR*, LNCS 5201, pages 147–161. Springer, 2008.

7 K. Chatterjee, R. Majumdar, and M. Jurdzinski. On nash equilibria in stochastic games. In *Proc. 13th CSL*, LNCS 3210, pages 26–40. Springer, 2004.

8 L. de Alfaro and T.A. Henzinger. Concurrent ω-regular games. In *15th LICS*, pages 141–154, 2000.

9 D. Fisman, O. Kupferman, and Y. Lustig. Rational synthesis. In *Proc. 16th TACAS*, LNCS 6015, pages 190–204. Springer, 2010.

10 G. Gottlob, G. Greco, and F. Scarcello. Pure Nash equilibria: hard and easy games. In *Proc. 9th TARK*, pages 215–230, 2003.

11 B. Jobstmann, A. Griesmayer, and R. Bloem. Program repair as a game. In *Proc. 17th CAV*, LNCS 3576, pages 226–238, Springer 2005,

12 R.M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

13 A. Korilis, A. Lazar and A. Orda. Achieving Network Optima Using Stackelberg Routing Strategies. In *IEEE/ACM Trans. Netw.*, 5(1):161–173, 1997.

14 O. Kupferman, G. Perelli, and M.Y. Vardi. Synthesis with rational environments. In *Proc. 12th EUMAS*. Springer, 2014.

15 W. Li, L. Dworkin, and S. A. Seshia. Mining assumptions for synthesis. In *Proc. 9th MEMOCODE*, pages 43–50, 2011.

16 F. Laroussinie, N. Markey, and G. Oreiby. On the Expressiveness and Complexity of ATL. *LMCS*, 4(2), 2008.

17 D.A. Martin. Borel determinacy. *Annals of Mathematics*, 65:363–371, 1975.

18 J.F. Nash. Equilibrium points in n-person games. In *Proceedings of the National Academy of Sciences of the United States of America*, 1950.

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

20 D. Rosenberg, E. Solan, and N. Vieille. The maxmin value of stochastic games with imperfect monitoring. *Int. J. Game Theory*, 32(1):133–150, 2003.

21 T. Roughgarden. Stackelberg Scheduling Strategies. *SIAM J. Comput.*, 33(2):332–350, 2004.

22 M. Ummels. The complexity of nash equilibria in infinite multiplayer games. In *Proc. 11th FOSSACS*, pages 20–34, 2008.

23 J. von Neumann and O. Morgenstern. Theory of games and economic behavior.

An Automata-Theoretic Approach to the Verification of Distributed Algorithms*

C. Aiswarya[1], Benedikt Bollig[2], and Paul Gastin[2]

1 **Uppsala University, Sweden**
aiswarya.cyriac@it.uu.se
2 **LSV, ENS Cachan, CNRS, Inria, France**
{bollig,gastin}@lsv.ens-cachan.fr

Abstract

We introduce an automata-theoretic method for the verification of distributed algorithms running on ring networks. In a distributed algorithm, an arbitrary number of processes cooperate to achieve a common goal (e.g., elect a leader). Processes have unique identifiers (pids) from an infinite, totally ordered domain. An algorithm proceeds in synchronous rounds, each round allowing a process to perform a bounded sequence of actions such as send or receive a pid, store it in some register, and compare register contents wrt. the associated total order. An algorithm is supposed to be correct independently of the number of processes. To specify correctness properties, we introduce a logic that can reason about processes and pids. Referring to leader election, it may say that, at the end of an execution, each process stores the maximum pid in some dedicated register. Since the verification of distributed algorithms is undecidable, we propose an underapproximation technique, which bounds the number of rounds. This is an appealing approach, as the number of rounds needed by a distributed algorithm to conclude is often exponentially smaller than the number of processes. We provide an automata-theoretic solution, reducing model checking to emptiness for alternating two-way automata on words. Overall, we show that round-bounded verification of distributed algorithms over rings is PSPACE-complete.

1998 ACM Subject Classification C.2.4 Distributed Systems, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases distributed algorithms, verification, leader election

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.340

1 Introduction

Distributed algorithms are a classic discipline of computer science and continue to be an active field of research [17]. A distributed algorithm employs several processes, which perform one and the same program to achieve a common goal. It is required to be correct independently of the number of processes. Prominent examples are leader-election algorithms, whose task is to determine a unique leader process and to announce it to all other processes. Those algorithms are often studied for ring architectures. One practical motivation comes from local-area networks that are based on a token-ring protocol. Moreover, rings generally allow one to nicely illustrate the main conceptual ideas of an algorithm.

However, it is well-known that there is no (deterministic) distributed algorithm over rings that elects a leader under the assumption of anonymous processes. Therefore, classical algorithms, such as Franklin's algorithm [12] or the Dolev-Klawe-Rodeh algorithm [7], assume

* Supported by LIA InForMel.

that every process is equipped with a unique process identifier (pid) from an infinite, totally ordered domain. In this paper, we consider such distributed algorithms, which work on ring architectures and can access unique pids as well as the associated total order.

Distributed algorithms are intrinsically hard to analyze. Correctness proofs are often intricate and use subtle inductive arguments. Therefore, it is worthwhile to consider automatic verification methods such as model checking. Besides a formal model of an algorithm, this requires a generic specification language that is feasible from an algorithmic point of view but expressive enough to formulate correctness properties. In this paper, we propose a language that can reason about processes, states, and pids. In particular, it will allow us to formalize when a leader-election algorithm is correct: *At the end of an execution, every process stores, in register r, the maximum pid among all processes.* Our language is inspired by Data-XPath, which can reason about trees over infinite alphabets [4, 11].

However, formal verification of distributed algorithms cumulates various difficulties that already arise, separately, in more standard verification: First, the number of processes is unknown, which amounts to parameterized verification [10]; second, processes manipulate data from an infinite domain [4, 11]. In each case, even simple verification questions are undecidable, and so is the combination of both.

A successful approach to retrieving decidability has been a form of *bounded model checking*. The idea is to consider correctness up to some parameter, which restricts the set of runs of the algorithm. This is natural in the context of distributed algorithms, which usually proceed in *rounds*. In each round, a process may emit some messages (here: pids) to its neighbors, and then receive messages from its neighbors. Pids can be stored in registers, and a process can check the relation between stored pids before it moves to a new state. The number of rounds is often exponentially smaller than the number of processes. Thus, a small number of rounds allows us to verify correctness of an algorithm for a large number of processes.

The key idea of our method is to interpret a (round-bounded) execution of a distributed algorithm symbolically as a word-like structure over a finite alphabet. The finite alphabet is constituted by the transitions that occur in the algorithm and possibly contain tests of pids wrt. equality or the associated total order. To determine feasibility of a symbolic execution (i.e., *is there a ring that satisfies all the guards employed?*), we use propositional dynamic logic with loop and converse (LCPDL) over words [13]. Basically, we translate a given distributed algorithm into a formula that detects cyclic (i.e., contradictory) smaller-than tests. Its models are precisely the feasible symbolic executions. A specification is translated into LCPDL as well so that verification amounts to checking satisfiability of a single formula. The latter can be reduced to an emptiness problem for alternating two-way automata over words so that we obtain a PSpace procedure for round-bounded model checking.

Related Work: Considerable effort has been devoted to the formal verification of fault-tolerant algorithms, which have to cope with faults such as lost or corrupted messages (e.g., [6, 15]). After all, there have been only very few generic approaches to model checking distributed algorithms. In [14], several possible reasons for this are identified, among them the presence of unbounded data types and an unbounded number of processes, which we have to treat simultaneously in our framework. Parameterized model checking of ring-based systems where communication is subject to a token policy and the message alphabet is finite has been studied in [9, 8, 3]. In [8], cutoff results are obtained for LTL\X specifications when a bound is placed on the number of times a token may change values.

The theory of words and trees over infinite alphabets (aka data words/trees) provides an elegant formal framework for database-related notions such as XML documents [4], or for the analysis of programs with data structures such as lists [2]. The difference to our work

is that we model distributed algorithms and provide a logical specification language which borrows concepts from [4, 11]. The paper [5] pursued a symbolic model-checking approach to *sequential* systems involving data, but pids could only be compared for equality. The ordering on the data domain has a subtle impact on the choice of the specification language.

Full proofs can be found in the full version of the paper [1].

2 Distributed Algorithms

By $\mathbb{N} = \{0, 1, 2, \ldots\}$, we denote the set of natural numbers. For $n \in \mathbb{N}$, we set $[n] = \{1, \ldots, n\}$ and $[n]_0 = \{0, 1, \ldots, n\}$. The set of finite words over an alphabet A is denoted by A^*, and the set of nonempty finite words by A^+.

Syntax of Distributed Algorithms. We consider distributed algorithms that run on arbitrary ring architectures. A ring consists of a finite number of processes, each having a unique process identifier (pid). Every process has a unique left neighbor (referred to by **left**) and a unique right neighbor (referred to by **right**). Formally, a *ring* is a tuple $\mathcal{R} = (n : p_1, \ldots, p_n)$, given by its size $n \geq 1$ and the pids $p_i \in \mathbb{N}$ assigned to processes $i \in [n]$. We require that pids are unique, i.e., $p_i \neq p_j$ whenever $i \neq j$. For a process $i < n$, process $i + 1$ is the right neighbor of i. Moreover, 1 is the right neighbor of n. Analogously, if $i > 1$, then $i - 1$ is the left neighbor of i. Moreover, n is the left neighbor of 1. Thus, processes 1 and n must not be considered as the "first" or "last" process. Actually, a distributed algorithm will not be able to distinguish between, for example, $(4 : 4, 1, 5, 2)$ and $(4 : 5, 2, 4, 1)$.

One given distributed algorithm can be run on *any* ring. It is given by a single program $\mathcal{D}$, and each process runs a copy of $\mathcal{D}$. It is convenient to think of $\mathcal{D}$ as a (finite) automaton. Processes proceed in synchronous rounds. In one round, *every* process executes one transition of its program. In addition to changing its state, each process may optionally perform the following phases within a round: (i) send some pids to its neighbors, (ii) receive pids from its neighbors and store them in registers, (iii) compare register contents with one another, (iv) update its registers. For example, consider the transition $t = \langle s\colon \mathbf{left}!r\,;\mathbf{right}!r'\,;\mathbf{right}?r'\,;r < r'\,;r := r'\,;\mathbf{goto}\ s'\rangle$. A process can execute t if it is in state s. It then sends the contents of register r to its left neighbor and the contents of r' to its right neighbor. If, afterwards, it receives a pid p from its right neighbor, it stores p in r'. If p is greater than what has been stored in r, it sets r to p and goes to state s'. Otherwise, the transition is not applicable. The first phase can, alternatively, be filled with a special command **fwd**. Then, a process will just forward any pid it receives. Note that a message can be forwarded, in one and the same round, across several processes executing **fwd**.

▶ **Definition 1.** A *distributed algorithm* $\mathcal{D} = (S, s_0, \mathit{Reg}, \Delta)$ consists of a nonempty finite set S of *(local) states*, an *initial state* $s_0 \in S$, a nonempty finite set *Reg* of *registers*, and a nonempty finite set Δ of *transitions*. A transition is of the form $\langle s\colon \mathit{send}\,;\mathit{rec}\,;\mathit{guard}\,;\mathit{update}\,;\mathbf{goto}\ s'\rangle$ where $s, s' \in S$ and the components *send*, *rec*, *guard*, and *update* are built as follows:

send ::= **skip** | **fwd** | **left**!r | **right**!r | **left**!r ; **right**!r'

rec ::= **skip** | **left**?r | **right**?r | **left**?r ; **right**?r'

guard ::= **skip** | $r < r'$ | $r = r'$ | $\mathit{guard}\,;\mathit{guard}$

update ::= **skip** | $r := r'$ | $\mathit{update}\,;\mathit{update}$

with r and r' ranging over *Reg*. We require that (1) in a *rec* statement of the form **left**?r ; **right**?r', we have $r \neq r'$ (actually, the order of the two receive actions does not matter), and (2) in an *update* statement, every register occurs at most once as a left-hand side. In the following, occurrences of "**skip** ;" are omitted. ◀

states: $active, passive$
$found$
initial state: $active$
registers: id, r, r_1, r_2

$t_1 = \langle active\colon \mathbf{left}!id\,;\mathbf{right}!id\,;\mathbf{left}?r_1\,;\mathbf{right}?r_2\,;r_1 < id\,;r_2 < id\,;\mathbf{goto}\ active\rangle$
$t_2 = \langle active\colon \underline{\qquad\qquad\qquad\qquad}\,;id < r_1\,;\mathbf{goto}\ passive\rangle$
$t_3 = \langle active\colon \underline{\qquad\qquad\qquad\qquad}\,;id < r_2\,;\mathbf{goto}\ passive\rangle$
$t_4 = \langle active\colon \underline{\qquad\qquad\qquad\qquad}\,;id = r_1\,;r := id\,;\mathbf{goto}\ found\rangle$
$t_5 = \langle passive\colon \mathbf{fwd}\,;\mathbf{left}?r\,;\mathbf{goto}\ passive\rangle$

Figure 1 Franklin's leader-election algorithm $\mathcal{D}_{\mathsf{Franklin}}$.

states: $active_0, active_1$
$passive, found$
initial state: $active_0$
registers: id, r, r', r''

$t_1 = \langle active_0\colon \mathbf{right}!r\,;\mathbf{left}?r'\,;\mathbf{goto}\ active_1\rangle$
$t_2 = \langle active_1\colon \mathbf{right}!r'\,;\mathbf{left}?r''\,;r'' < r'\,;r < r'\,;r := r'\,;\mathbf{goto}\ active_0\rangle$
$t_3 = \langle active_1\colon \underline{\qquad\qquad}\,;r' < r\,;\mathbf{goto}\ passive\rangle$
$t_4 = \langle active_1\colon \underline{\qquad\qquad}\,;r' < r''\,;\mathbf{goto}\ passive\rangle$
$t_5 = \langle active_1\colon \underline{\qquad\qquad}\,;r = r'\,;\mathbf{goto}\ found\rangle$
$t_6 = \langle passive\colon \mathbf{fwd}\,;\mathbf{left}?r\,;\mathbf{goto}\ passive\rangle$

Figure 2 Dolev-Klawe-Rodeh leader-election algorithm $\mathcal{D}_{\mathsf{DKR}}$.

Note that a guard $r \leq r'$ can be simulated in terms of guards $r < r'$ and $r = r'$, using several transitions. We separate $<$ and $=$ for convenience. They are actually quite different in nature, as we will see later in the proof of our main result.

At the beginning of an execution of an algorithm, every register contains the pid of the respective process. We also assume, wlog., that there is a special register $id \in Reg$ that is never updated, i.e., no transition contains a command of the form $\mathbf{left}?id$, $\mathbf{right}?id$, or $id := r$. A process can thus, at any time, access its own pid in terms of id.

In the semantics, we will suppose that all updates of a transition happen simultaneously, i.e., after executing $r := r'\,; r' := r$, the values previously stored in r and r' will be swapped (and do not necessarily coincide). As, moreover, the order of two sends and the order of two receives within a transition do not matter, this will allow us to identify a transition with the set of states, commands (apart from **skip**), and guards that it contains. For example, $t = \langle s\colon \mathbf{left}!r\,;\mathbf{right}!r'\,;\mathbf{right}?r'\,;r < r'\,;r := r'\,;\mathbf{goto}\ s'\rangle$ is considered as the set $t = \{s\,, \mathbf{left}!r\,, \mathbf{right}!r'\,, \mathbf{right}?r'\,, r < r'\,, r := r'\,, \mathbf{goto}\ s'\}$.

Before defining the semantics of a distributed algorithm, we will look at two examples.

▶ **Example 2** (Franklin's Leader-Election Algorithm)**.** Consider Franklin's algorithm $\mathcal{D}_{\mathsf{Franklin}}$ to determine a leader in a ring [12]. It is given in Figure 1. The goal is to assign leadership to the process with the highest pid. To do so, every process sends its own pid to both neighbors, receives the pids of its left and right neighbor, and stores them in registers r_1 and r_2, respectively (transitions $t_1, \ldots, t_4$). If a process is a local maximum, i.e., $r_1 < id$ and $r_2 < id$ hold, it is still in the race for leadership and stays in state *active*. Otherwise, it has to take t_2 or t_3 and goes into state *passive*. In *passive*, a process will just forward any pid it receives and store the message coming from the left in r (transition t_5). Notice that, within the same round, a message may be forwarded through (and stored by) several consecutive passive processes, until it reaches an active one. When an active process receives its own pid (transition t_4), it knows it is the only remaining active process. It copies its own pid into r, which henceforth refers to the leader. We may say that a run is accepting (or terminating) when all processes terminate in *passive* or *found*. Then, at the end of any accepting run, (i) there is exactly one process i_0 that terminates in *found*, (ii) all processes store the pid of i_0 in register r, and the pid of i_0 is the maximum of all pids in the ring. Since, in every round, at least half of the active processes become passive, the algorithm terminates after at most $\lfloor \log_2 n \rfloor + 1$ rounds where n is the number of processes. ◀

▶ **Example 3** (Dolev-Klawe-Rodeh Leader-Election Algorithm)**.** The Dolev-Klawe-Rodeh leader-election algorithm [7] is an adaptation of Franklin's algorithm to cope with unidirectional rings, where a process can only, say, send to the right and receive from the left. The algorithm, denoted $\mathcal{D}_{\mathsf{DKR}}$, is given in Figure 2. The idea is that the local maximum among the processes $i-2, i-1, i$ is determined by i (rather than $i-1$). Therefore, each process i will execute two transitions, namely t_1 and t_2, and store the pids sent by $i-2$ and $i-1$ in r'' and r', respectively. After two rounds, since r still contains the pid of i itself, i can test if $i-1$ is a local maximum among $i-2, i-1, i$ using the guards in transition t_2. If both guards are satisfied, i stores the pid sent by $i-1$ in r. It henceforth "represents" process $i-1$, which is still in the race, and goes to state $active_0$. Otherwise, it enters *passive*, which has the same task as in Franklin's algorithm. The algorithm is correct in the following sense: At the end of an accepting run (each process ends in *passive* or *found*), (i) there is exactly one process that terminates in *found* (but not necessarily the one with the highest pid), and (ii) all processes store the maximal pid in register r. The algorithm terminates after at most $2\lfloor \log_2 n \rfloor + 2$ rounds. Note that the correctness of $\mathcal{D}_{\mathsf{DKR}}$ is less clear than that of $\mathcal{D}_{\mathsf{Franklin}}$. ◀

Semantics of Distributed Algorithms. Now, we give the formal semantics of a distributed algorithm $\mathcal{D} = (S, s_0, \mathit{Reg}, \Delta)$. Recall that $\mathcal{D}$ can be run on any ring $\mathcal{R} = (n : p_1, \ldots, p_n)$. An ($\mathcal{R}$-)configuration of $\mathcal{D}$ is a tuple $(s_1, \ldots, s_n, \rho_1, \ldots, \rho_n)$ where s_i is the current state of process i and $\rho_i : \mathit{Reg} \to \{p_1, \ldots, p_n\}$ maps each register to a pid. The configuration is called *initial* if, for all processes $i \in [n]$, we have $s_i = s_0$ and $\rho_i(r) = p_i$ for all $r \in \mathit{Reg}$. Note that there is a unique initial $\mathcal{R}$-configuration.

In one round, the algorithm moves from one configuration to another one. This is described by a relation $C \overset{t}{\rightsquigarrow} C'$ where $C = (s_1, \ldots, s_n, \rho_1, \ldots, \rho_n)$ and $C' = (s'_1, \ldots, s'_n, \rho'_1, \ldots, \rho'_n)$ are $\mathcal{R}$-configurations and $t = (t_1, \ldots, t_n) \in \Delta^n$ is a tuple of transitions where t_i is executed by process i. To determine when $C \overset{t}{\rightsquigarrow} C'$ holds, we first define two auxiliary relations. For registers $r, r' \in \mathit{Reg}$ and processes $i, j \in [n]$, we write $r@i \rightarrowtail r'@j$ if the contents of r is sent to the right from i to j, where it is stored in r'. Thus, we require that

$$\mathbf{right}!r \in t_i \ \wedge\ \mathbf{left}?r' \in t_j \ \wedge\ \mathbf{fwd} \in t_k \text{ for all } k \in \mathit{Between}(i, j)$$

where $\mathit{Between}(i, j)$ means $\{i+1, \ldots, j-1\}$ if $i < j$ or $\{1, \ldots, j-1, i+1, \ldots, n\}$ if $j \leq i$. Note that, due to the **fwd** command, $r@i \rightarrowtail r'@j$ may hold for several r' and j. The meaning of $r'@j \leftarrowtail r@i$ is analogous, we just replace "right direction" by "left direction":

$$\mathbf{left}!r \in t_i \ \wedge\ \mathbf{right}?r' \in t_j \ \wedge\ \mathbf{fwd} \in t_k \text{ for all } k \in \mathit{Between}(j, i).$$

The guards in the transitions $t_1, \ldots, t_n$ are checked against "intermediate" register assignments $\hat{\rho}_1, \ldots, \hat{\rho}_n : \mathit{Reg} \to \{p_1, \ldots, p_n\}$, which are defined as follows:

$$\hat{\rho}_j(r') = \begin{cases} \rho_i(r) & \text{if } r@i \rightarrowtail r'@j \text{ or } r'@j \leftarrowtail r@i \\ \rho_j(r') & \text{if, for all } r, i, \text{ neither } r@i \rightarrowtail r'@j \text{ nor } r'@j \leftarrowtail r@i \end{cases}$$

Note that this is well-defined, due to condition (1) in Definition 1.

Now, we write $C \overset{t}{\rightsquigarrow} C'$ if, for all $j \in [n]$ and $r, r' \in \mathit{Reg}$, the following hold:

1. $s_j \in t_j$ and $(\mathbf{goto}\ s'_j) \in t_j$,
2. $\hat{\rho}_j(r) < \hat{\rho}_j(r')$ if $(r < r') \in t_j$,
3. $\hat{\rho}_j(r) = \hat{\rho}_j(r')$ if $(r = r') \in t_j$,
4. $\rho'_j(r) = \begin{cases} \hat{\rho}_j(r') & \text{if } (r := r') \in t_j \\ \hat{\rho}_j(r) & \text{if } t_j \text{ does not contain an update of the form } r := r'' \end{cases}$

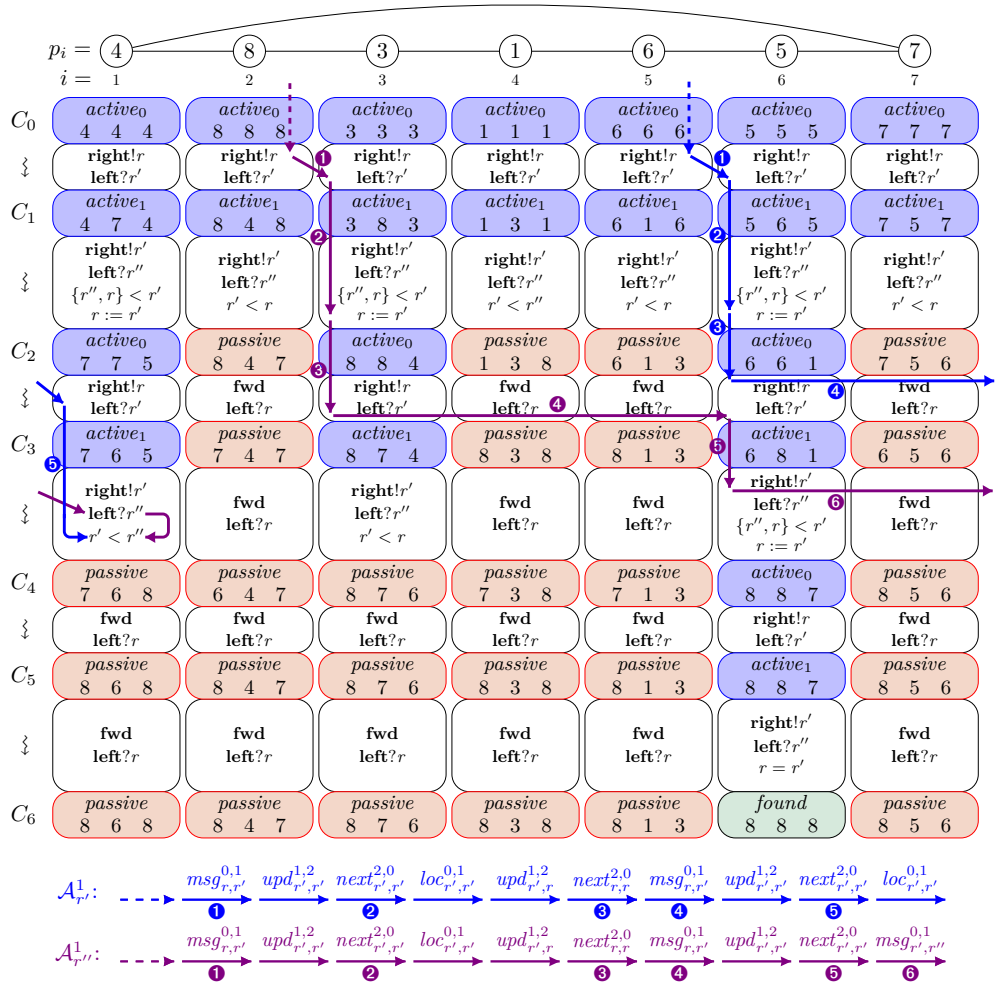


Figure 3 Run of Dolev-Klawe-Rodeh algorithm and runs of path automata.

Again, 4. is well-defined thanks to condition (2) in Definition 1.

An ($\mathcal{R}$-)*run* of $\mathcal{D}$ is a sequence $\chi = C_0 \overset{t^1}{\rightsquigarrow} C_1 \overset{t^2}{\rightsquigarrow} \cdots \overset{t^k}{\rightsquigarrow} C_k$ where $k \geq 1$, C_0 is the initial $\mathcal{R}$-configuration, and $t^j = (t^j_1, \ldots, t^j_n) \in \Delta^n$ for all $j \in [k]$. We call k the *length* of χ. Note that χ uniquely determines the underlying ring $\mathcal{R}$.

▶ Remark. A receive command is always non-blocking even if there is no corresponding send. As an alternative semantics, one could require that it can only be executed if there has been a matching send, or vice versa. One could even include tags from a finite alphabet that can be sent along with pids. All this will not change any of the forthcoming results. ◀

▶ **Example 4.** A run of $\mathcal{D}_{\mathsf{DKR}}$ from Example 3 on the ring $\mathcal{R} = (7 : 4, 8, 3, 1, 6, 5, 7)$ is depicted in Figure 3 (for the moment, we may ignore the blue and violet lines). A colored row forms a configuration. The three pids in a cell refer to registers r, r', r'', respectively (we ignore *id*). Moreover, a non-colored row forms, together with the states above and below, a transition tuple. When looking at the step from C_3 to C_4, we have, for example, $r'@3 \rightarrowtail r@4$ and $r'@3 \rightarrowtail r''@6$. Moreover, $r'@6 \rightarrowtail r@7$ and $r'@6 \rightarrowtail r''@1$ (recall that we are in a ring). Note that the run conforms to the correctness property formulated in Example 3. In particular, in the final configuration, all processes store the maximum pid in register r. ◀

3 The Specification Language

In Examples 2 and 3, we informally stated the correctness criterion for the presented algorithms (e.g., "at the end, all processes store the maximal pid in register r"). Now, we introduce a *formal* language to specify correctness properties. It is defined wrt. a given distributed algorithm $\mathcal{D} = (S, s_0, Reg, \Delta)$, which we fix for the rest of this section.

Typically, one requires that a distributed algorithm is correct no matter what the underlying ring is. Since we will bound the number of rounds, we moreover study a form of partial correctness. Accordingly, a property is of the form $\forall_{rings}\forall_{runs}\forall_{\mathsf{m}}\varphi$, which has to be read as "for all rings, all runs, and all processes m, we have φ". The marking m is used to avoid to "get lost" in a ring when writing the property φ. This is like placing a pebble in the ring that can be retrieved at any time. Actually, φ allows us to "navigate" back and forth ($\uparrow$ and $\downarrow$) in a run, i.e., from one configuration to the previous or next one (similar to a temporal logic with past operators). By means of $\leftarrow$ and $\rightarrow$, we may also navigate horizontally within a configuration, i.e., from one process to a neighboring one.

Essentially, a sequence of configurations is interpreted as a cylinder (cf. Figure 3) that can be explored using regular expressions π over $\{\epsilon, \leftarrow, \rightarrow, \uparrow, \downarrow\}$ (where ϵ means "stay"). At a given position/coordinate of the cylinder, we can check *local (or positional)* properties like the state taken by a process, or whether we are on the marked process m. Such a property can be combined with a regular expression π: The formula $[\pi]\varphi$ says that φ holds at every position that is reachable through a π-path (a path matching π). Dually, $\langle\pi\rangle\varphi$ holds if there is a π-path to some position where φ is satisfied. The most interesting construct in our logic is $\langle\pi\rangle r \bowtie \langle\pi'\rangle r'$, where $\bowtie \in \{=, \neq, <, \leq\}$, which has been used for reasoning about XML documents [4, 11]. It says that, from the current position, there are a π-path and a π'-path that lead to positions y and y', respectively, such that the pid stored in register r at y and the pid stored in r' at y' satisfy the relation $\bowtie$.

We will now introduce our logic in full generality. Later, we will restrict the use of $<$- and $\leq$-guards to obtain positive results.

▶ **Definition 5.** The logic DataPDL($\mathcal{D}$) is given by the following grammar:

$$\begin{aligned}\Phi &::= \forall_{rings}\forall_{runs}\forall_{\mathsf{m}}\varphi\\ \varphi, \varphi' &::= \mathsf{m} \mid s \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \varphi \Rightarrow \varphi' \mid [\pi]\varphi \mid \langle\pi\rangle r \bowtie \langle\pi'\rangle r'\\ \pi, \pi' &::= \{\varphi\}? \mid d \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^*\end{aligned}$$

where $s \in S$, $r, r' \in Reg$, $\bowtie \in \{=, \neq, <, \leq\}$, and $d \in \{\epsilon, \leftarrow, \rightarrow, \uparrow, \downarrow\}$. ◀

We call φ a *local formula*, and π a *path formula*. We use common abbreviations such as $\mathit{false} = \mathsf{m} \wedge \neg\mathsf{m}$, $\langle\pi\rangle\varphi = \neg[\pi]\neg\varphi$, and $\varphi \vee \varphi' = \neg(\neg\varphi \wedge \neg\varphi')$, and we may write $\pi\pi'$ instead of $\pi \cdot \pi'$. Implication $\Rightarrow$ is included explicitly in view of the restriction defined below.

Next, we define the semantics. Consider a run $\chi = C_0 \overset{t^1}{\rightsquigarrow} C_1 \overset{t^2}{\rightsquigarrow} \cdots \overset{t^k}{\rightsquigarrow} C_k$ of $\mathcal{D}$ where $C_j = (s_1^j, \ldots, s_n^j, \rho_1^j, \ldots, \rho_n^j)$, i.e., n is the number of processes in the underlying ring. A local formula φ is interpreted over χ wrt. a marked process $m \in [n]$ and a position $(i, j) \in \mathit{Pos}(\chi)$ where $\mathit{Pos}(\chi) = [n] \times [k]_0$. Let us define when $\chi, m, (i, j) \models \varphi$ holds. The operators $\neg$, $\wedge$, and $\Rightarrow$ are as usual. Moreover, $\chi, m, (i, j) \models \mathsf{m}$ if $i = m$, and $\chi, m, (i, j) \models s$ if $s_i^j = s$.

The other local formulas use path formulas. The semantics of a path formula π is given in terms of a binary relation $[\![\pi]\!]_{\chi,m} \subseteq \mathit{Pos}(\chi) \times \mathit{Pos}(\chi)$, which we define below. First, we set:

- $\chi, m, (i, j) \models [\pi]\varphi$ if $\forall (i', j')$ such that $((i, j), (i', j')) \in [\![\pi]\!]_{\chi,m}$, we have $\chi, m, (i', j') \models \varphi$
- $\chi, m, (i, j) \models \langle\pi\rangle r \bowtie \langle\pi'\rangle r'$ (where $\bowtie \in \{=, \neq, <, \leq\}$) if $\exists (i_1, j_1), (i_2, j_2)$ such that $((i, j), (i_1, j_1)) \in [\![\pi]\!]_{\chi,m}$ and $((i, j), (i_2, j_2)) \in [\![\pi']\!]_{\chi,m}$ and $\rho_{i_1}^{j_1}(r) \bowtie \rho_{i_2}^{j_2}(r')$

It remains to define $[\![\pi]\!]_{\chi,m}$ for a path formula π. First, a local test and a stay ϵ do not "move" at all: $[\![\{\varphi\}?]\!]_{\chi,m} = \{(x,x) \mid x \in Pos(\chi) \text{ such that } \chi, m, x \models \varphi\}$, and $[\![\epsilon]\!]_{\chi,m} = \{(x,x) \mid x \in Pos(\chi)\}$. Using $\rightarrow$, we move to the right neighbor of a process: $[\![\rightarrow]\!]_{\chi,m} = \{((i,j),(i+1,j)) \mid i \in [n-1] \text{ and } j \in [k]_0\} \cup \{((n,j),(1,j)) \mid j \in [k]_0\}$. We define $[\![\leftarrow]\!]_{\chi,m}$ accordingly. Moreover, $[\![\downarrow]\!]_{\chi,m} = \{((i,j),(i,j+1)) \mid i \in [n] \text{ and } j \in [k-1]_0\}$, and similarly for $[\![\uparrow]\!]_{\chi,m}$. The regular constructs, $+$, $\cdot$, and $*$ are as expected and refer to the union, relation composition, and star over binary relations.

Finally, $\mathcal{D}$ satisfies the DataPDL formula $\forall_{rings}\forall_{runs}\forall_{\mathsf{m}}\varphi$, written $\mathcal{D} \models \forall_{rings}\forall_{runs}\forall_{\mathsf{m}}\varphi$, if, for all rings $\mathcal{R} = (n : \ldots)$, all $\mathcal{R}$-runs χ, and all processes $m \in [n]$, we have $\chi, m, (m,0) \models \varphi$. Thus, φ is evaluated at the first configuration, wrt. process m which can be chosen arbitrarily.

Next, we define a restricted logic, DataPDL$^{\ominus}(\mathcal{D})$, for which we later present our main result. We say that a path formula π is *unambiguous* if, from a given position, it defines at most one reference point. Formally, for all rings $\mathcal{R} = (n : \ldots)$, $\mathcal{R}$-runs χ of $\mathcal{D}$, processes $m \in [n]$, and positions $x \in Pos(\chi)$, there is at most one $x' \in Pos(\chi)$ such that $(x,x') \in [\![\pi]\!]_{\chi,m}$. For example, ϵ, $\downarrow$, $\rightarrow$, and $\rightarrow^*\{\mathsf{m}\}?$ are unambiguous, while $\rightarrow^*$ and $\leftarrow + \rightarrow$ are not unambiguous.

▶ **Definition 6.** A DataPDL($\mathcal{D}$) formula is contained in DataPDL$^{\ominus}(\mathcal{D})$ if every subformula $\varphi = \langle\pi\rangle r \bowtie \langle\pi'\rangle r'$ with $\bowtie \in \{<, \leq\}$ is such that π and π' are *unambiguous*. Moreover, φ must *not* occur (i) in the scope of a negation, (ii) on the left-hand side of an implication $\_ \Rightarrow \_$, or (iii) within a test $\{\_\}?$. Note that guards using $=$ and $\neq$ are still unrestricted. ◀

▶ **Example 7.** Let us *formalize*, in DataPDL$^{\ominus}(\mathcal{D})$, the correctness criteria for $\mathcal{D}_{\mathsf{Franklin}}$ and $\mathcal{D}_{\mathsf{DKR}}$ that we stated informally in Examples 2 and 3. Consider the following local formulas:

$$\varphi_{\mathsf{last}} = [\downarrow]\mathit{false} \qquad \varphi_{\mathsf{max}} = [\rightarrow^*]\big(\langle\epsilon\rangle id \leq \langle\pi_{\mathsf{found}}\rangle r\big)$$
$$\varphi_{\mathsf{acc}} = [\rightarrow^*](\mathit{passive} \vee \mathit{found}) \qquad \varphi_{r=id} = \langle\pi_{\mathsf{found}}\rangle\big(\langle\epsilon\rangle r = \langle\epsilon\rangle id\big)$$
$$\varphi_{\mathsf{found}} = \langle\pi_{\mathsf{found}}\rightarrow(\{\neg\mathit{found}\}?\rightarrow)^*\rangle\mathsf{m} \qquad \varphi_{r=r} = \neg\big(\langle\epsilon\rangle r \neq \langle\rightarrow^*\rangle r\big)$$

where $\pi_{\mathsf{found}} = (\{\neg\mathit{found}\}?\rightarrow)^*\{\mathit{found}\}?$. Note that π_{found} is unambiguous: while going to the right, it always stops at the *nearest* process that is in state *found*. Thus, φ_{max} is indeed a local DataPDL$^{\ominus}$ formula. Consider the DataPDL$^{\ominus}$ formula

$$\Phi_1 = \forall_{rings}\forall_{runs}\forall_{\mathsf{m}}[\downarrow^*]\big((\varphi_{\mathsf{last}} \wedge \varphi_{\mathsf{acc}}) \Rightarrow (\varphi_{\mathsf{found}} \wedge \varphi_{\mathsf{max}} \wedge \varphi_{r=r} \wedge \varphi_{r=id})\big)\,.$$

It says that, at the end (i.e., in the last configuration) of each accepting run, expressed by $[\downarrow^*]\big((\varphi_{\mathsf{last}} \wedge \varphi_{\mathsf{acc}}) \Rightarrow \cdots\big)$, we have that (i) there is exactly one process i_0 that ends in state *found* (guaranteed by φ_{found}), (ii)register r of i_0 contains the maximum over all pids (φ_{max}), (iii)register r of i_0 contains the pid of i_0 itself ($\varphi_{r=id}$), and (iv) all processes store the same pid in r ($\varphi_{r=r}$). Thus, $\mathcal{D}_{\mathsf{Franklin}} \models \Phi_1$. On the other hand, we have $\mathcal{D}_{\mathsf{DKR}} \not\models \Phi_1$, because in $\mathcal{D}_{\mathsf{DKR}}$ the process that ends in *found* is not necessarily the process with the maximum pid. However, we still have $\mathcal{D}_{\mathsf{DKR}} \models \Phi_2$ where

$$\Phi_2 = \forall_{rings}\forall_{runs}\forall_{\mathsf{m}}[\downarrow^*]\big((\varphi_{\mathsf{last}} \wedge \varphi_{\mathsf{acc}}) \Rightarrow (\varphi_{\mathsf{found}} \wedge \varphi_{\mathsf{max}} \wedge \varphi_{r=r})\big)\,.$$

The next example formulates the correctness constraint for a distributed sorting algorithm. We would like to say that, at the end of an accepting run, the pids stored in registers r are strictly totally ordered. Suppose φ_{acc} represents an acceptance condition and φ_{least} says that there is exactly one process that terminates in some dedicated state *least*, similarly to φ_{found} above. Then,

$$\Phi_3 = \forall_{rings}\forall_{runs}\forall_{\mathsf{m}}[\downarrow^*]\big((\varphi_{\mathsf{last}} \wedge \varphi_{\mathsf{acc}}) \Rightarrow (\varphi_{\mathsf{least}} \wedge [\rightarrow^*\{\neg\mathit{least}\}?](\langle\leftarrow\rangle r < \langle\epsilon\rangle r))\big)$$

makes sure that, whenever process j is not terminating in *least*, its left neighbor i stores a smaller pid in r than j does.

Note that Φ_1, Φ_2, and Φ_3 are indeed DataPDL$^{\ominus}$ formulas. ◀

▶ **Example 8.** We give a couple of examples to illustrate how the logic can be used to reason about temporal properties. Consider the specifications: (a) Every process remains active until it becomes found or passive forever; (b) The value of the register r on any process is monotonously non-decreasing. These can be expressed by the DataPDL$^{\ominus}$ formulas below:

$$\Phi_a = \forall_{rings}\forall_{runs}\forall_{\mathsf{m}}\langle(\{active\}?\downarrow)^*\rangle(found \vee [\downarrow^*](passive))\,.$$

$$\Phi_b = \forall_{rings}\forall_{runs}\forall_{\mathsf{m}}[\downarrow^*]([\downarrow]false \vee (\langle\epsilon\rangle r \leq \langle\downarrow\rangle r))\,.$$ ◀

Unsurprisingly, model checking distributed algorithms against DataPDL$^{\ominus}$ is undecidable:

▶ **Theorem 9.** *The following problem is undecidable: Given a distributed algorithm $\mathcal{D}$ and $\Phi \in$ DataPDL$^{\ominus}(\mathcal{D})$, do we have $\mathcal{D} \models \Phi$? (Actually, this even holds for formulas Φ that express simple state-reachability properties and do not use any guards on pids.)*

4 Round-Bounded Model Checking

In situations where model checking is undecidable, a fruitful approach has been to underapproximate the behavior of a system. The idea is to introduce a parameter that measures a characteristic of a run. One then imposes a bound on this parameter and explores all behaviors up to that bound. In numerous distributed algorithms (cf. Examples 2 and 3), the number b of rounds needed to conclude is exponentially smaller than the number of processes, Recall that, in a single round, a message may be forwarded through an arbitrarily long sequence of processes. Therefore, b seems to be a promising parameter for bounded model checking of distributed algorithms.

For a distributed algorithm $\mathcal{D}$, a formula $\Phi = \forall_{rings}\forall_{runs}\forall_{\mathsf{m}}\varphi \in$ DataPDL$(\mathcal{D})$, and $b \geq 1$, we write $\mathcal{D} \models_b \Phi$ if, for all rings $\mathcal{R} = (n : \ldots)$, all $\mathcal{R}$-runs χ of length $k \leq b$, and all processes $m \in [n]$, we have $\chi, m, (m, 0) \models \varphi$. We now present our main result:

▶ **Theorem 10.** *The following problem is* PSPACE*-complete: Given a distributed algorithm $\mathcal{D}$, $\Phi \in$ DataPDL$^{\ominus}(\mathcal{D})$, and a natural number $b \geq 1$ (encoded in unary), do we have $\mathcal{D} \models_b \Phi$?*

The lower-bound can be obtained by a reduction from the intersection-emptiness problem for a list of finite automata. Before we prove the upper bound, let us discuss the result in more detail. We will first compare it with "naïve" approaches to solve related questions. Consider the problem to determine whether a distributed algorithm satisfies its specification for all rings up to size n and all runs up to length b. This problem is in coNP: We guess a ring (i.e., essentially, a permutation of pids) and a run, and we check, using [16], whether the run does *not* satisfy the formula. Next, suppose only b is given and the question is whether, for all rings up to size 2^b and all runs up to length b, the property holds. Then, the above procedure gives us a coNEXPTIME algorithm.

Thus, our result is interesting complexity-wise, but it offers some other advantages. First, it actually checks correctness (up to round number b) for *all* rings. This is essential when verifying distributed *protocols* against safety properties. Second, it reduces to a satisfiability check in the well-studied propositional dynamic logic with loop and converse (LCPDL) [13] on tables of bounded height. In Theorem 11 we show that this satifiability problem can be solved in PSPACE by a reduction to an emptiness check of alternating two-way automata

(A2As) [21] over words. The "naïve" approaches, on the other hand, do not seem to give rise to viable algorithms. Finally, our approach is uniform in the following sense: We will construct, in polynomial time, an LCPDL formula that describes precisely the symbolic abstractions of runs (over arbitrary rings) that violate (or satisfy) a given formula. Our construction is *independent* of the parameter b. The satisfiability check then requires a bound on the number of rounds (or on the number of processes), which can be adjusted gradually without changing the automaton.

Proof Outline for Upper Bound of Theorem 10. Let $\mathcal{D}$ be the given distributed algorithm and $\Phi \in \text{DataPDL}^{\ominus}(\mathcal{D})$. We will reduce model checking to the satisfiability problem for LCPDL [13]. While DataPDL$^{\ominus}$ is interpreted over runs, containing pids from an infinite alphabet, the new logic will reason about symbolic abstractions over a *finite* alphabet. A symbolic abstraction of a run only keeps the transitions and discards pids. Thus, it can be seen as a table whose entries are transitions (cf. Figure 3).

First, we translate $\mathcal{D}$ into an LCPDL formula. Essentially, it checks that guards are not used in a contradictory way. To compare $\mathcal{D}$ with Φ, the latter is translated into an LCPDL formula, too. However, there is a subtle point here. For simplicity, let us write $r < r'$ instead of $\langle\epsilon\rangle r < \langle\epsilon\rangle r'$. Satisfaction of a formula $r < r'$ can only be guaranteed in a symbolic execution if the flow of pids provides *evidence* that $r < r'$ really holds. More concretely, the (hypothetic) formula $(r < r') \vee (r = r') \vee (r' < r)$ is a tautology, but it may not be possible to prove $r < r'$ or $r' < r$ on the basis of a symbolic run. This is the reason why DataPDL$^{\ominus}$ restricts $<$- and $\leq$-tests. It is then indeed enough to reason about symbolic runs (cf. Lemma 13 below). We leave open whether one can deal with full DataPDL.

Overall, we reduce model checking to satisfiability of the conjunction of two LCPDL formulas of polynomial size: the formula representing the algorithm, and the negation of the formula representing the specification. Satisfiability of LCPDL over symbolic runs (of bounded height) can be checked in PSPACE as stated in Theorem 11. Our approach is, thus, automata theoretic in spirit, though the power of alternation is used differently than in [20], which translates LTL formulas into automata.

Next, we present the logic LCPDL over symbolic runs. Then, we translate $\mathcal{D}$ as well as its DataPDL$^{\ominus}$ specification into LCPDL. For the remainder of this section, we fix a distributed algorithm $\mathcal{D} = (S, s_0, \mathit{Reg}, \Delta)$.

PDL with Loop and Converse (LCPDL). As mentioned before, a symbolic abstraction of a run of $\mathcal{D}$ is a table, whose entries are transitions from the finite alphabet Δ. A *table* is a triple $T = (n, k, \lambda)$ where $n, k \geq 1$ and $\lambda : \mathit{Pos}(T) \to \Delta$ labels each position/coordinate from $\mathit{Pos}(T) = [n] \times [k]_0$ with a transition. Thus, we may consider that T has n columns and $k+1$ rows. In the following, we will write $T[i,j]$ for $\lambda(i,j)$, and $T[i]$ for the i-th column of T, i.e., $T[i] = T[i,0] \cdots T[i,k] \in \Delta^+$. Let Δ^{++} denote the set of all tables.

Formulas $\psi \in \text{LCPDL}(\mathcal{D})$ are interpreted over tables. Their syntax is given as follows:

$$\begin{aligned} \psi, \psi' \ ::= \ & t \mid s \mid \textbf{goto}\ s \mid \textbf{fwd} \mid \textbf{left}!r \mid \textbf{right}!r \mid \textbf{left}?r \mid \textbf{right}?r \mid r < r' \mid r = r' \mid r := r' \mid \\ & \neg\psi \mid \psi \wedge \psi' \mid \langle\pi\rangle\psi \mid \mathsf{loop}(\pi) \\ \pi, \pi' \ ::= \ & \{\psi\}? \mid d \mid \pi + \pi' \mid \pi \cdot \pi' \mid \pi^* \mid \pi^{-1} \mid \mathcal{A} \end{aligned}$$

where $t \in \Delta$, $s \in S$, $r, r' \in \mathit{Reg}$, $d \in \{\epsilon, \rightarrow, \downarrow\}$, and $\mathcal{A}$ is a *path automaton*[1]: a non-deterministic finite automaton whose transitions are labeled with path formulas π. Again, ψ

[1] We use automata in addition to regular expressions since using states makes it easier to describe a

is called a *local formula*. We use common abbreviations for disjunction, implication, *true*, and *false*, and we let $\pi^+ = \pi \cdot \pi^*$, $[\pi]\psi = \neg\langle\pi\rangle\neg\psi$, $\langle\pi\rangle = \langle\pi\rangle \mathit{true}$, $\leftarrow = \rightarrow^{-1}$, and $\uparrow = \downarrow^{-1}$.

The semantics of LCPDL is very similar to that of DataPDL. A local formula ψ is interpreted over a table $T \in \Delta^{++}$ and a position $x \in \mathit{Pos}(T)$. When it is satisfied, we write $T, x \models \psi$. Moreover, a path formula π determines a binary relation $[\![\pi]\!]_T \subseteq \mathit{Pos}(T) \times \mathit{Pos}(T)$, relating those positions that are connected by a path matching π.

We consider only the most important cases: We have $T, (i,j) \models t$ if $T[i,j] = t$. For a state, command, guard, or update γ, let $T, (i,j) \models \gamma$ if $\gamma \in T[i,j]$. Loop and converse are as expected: $T, x \models \mathsf{loop}(\pi)$ if $(x,x) \in [\![\pi]\!]_T$, and $[\![\pi^{-1}]\!]_T = \{(y,x) \mid (x,y) \in [\![\pi]\!]_T\}$. The semantics of $\rightarrow$ (and $\leftarrow$) is slightly different than in DataPDL, since we are not allowed to go beyond the last and first column. Thus, $[\![\rightarrow]\!]_T = \{((i,j),(i+1,j)) \mid i \in [n-1] \text{ and } j \in [k]_0\}$. However, we can simulate the "roundabout" of a ring and set $\hookrightarrow = \rightarrow + \{\neg\langle\rightarrow\rangle\}?\leftarrow^*\{\neg\langle\leftarrow\rangle\}?$ as well as $\hookleftarrow = \hookrightarrow^{-1}$. By symmetry, the first column of a table will play the role of a marked process in a ring (later, m will be translated to $\neg\langle\leftarrow\rangle$).

Finally, the semantics of path automata is given by $[\![\mathcal{A}]\!]_T = \{(x,y) \mid \text{there is } \pi_1 \cdots \pi_\ell \in L(\mathcal{A}) \text{ with } (x,y) \in [\![\pi_1 \cdots \pi_\ell]\!]_T\}$ where $L(\mathcal{A})$ contains a *sequence* $\pi_1 \cdots \pi_\ell$ of path formulas if $\mathcal{A}$ admits a path $q_0 \xrightarrow{\pi_1} q_1 \xrightarrow{\pi_2} \cdots \xrightarrow{\pi_\ell} q_\ell$ from its initial state q_0 to a final state q_ℓ.

A formula $\psi \in \mathrm{LCPDL}(\mathcal{D})$ defines the language $L(\psi) = \{T \in \Delta^{++} \mid T, (1,0) \models \psi\}$. For $b \geq 1$, we denote by $L_b(\psi)$ the set of tables $(n,k,\lambda) \in L(\psi)$ such that $k \leq b$. The bounded height satisfiability problem for LCPDL asks the following: Given a distributed algorithm $\mathcal{D}$, a formula $\psi \in \mathrm{LCPDL}(\mathcal{D})$, and $b \geq 1$ (encoded in unary), do we have $L_b(\psi) = \emptyset$? Note that the input $\mathcal{D}$ is only needed to determine the signature of the logic.

▶ **Theorem 11.** *The bounded height satisfiability problem for* LCPDL *is* PSPACE*-complete.*

Proof sketch. We can restrict to tables of height $k = b$ (rather than $k \leq b$), since checking satisfiability for every height separately does not change the space complexity. We reduce the problem to words: A table $T = (n,k,\lambda)$ is considered as the word $T[1] \cdots T[n] \in \Delta^+$. Thus, the columns are written horizontally rather than vertically. When translating an LCPDL formula over tables into an LCPDL formula over words, going to the left or right involves some modulo counting: $\leftarrow$ is translated to $\leftarrow^{k+1}$, and $\rightarrow$ is translated to $\rightarrow^{k+1}$. We then follow the construction of [13] to obtain, in polynomial time, an alternating two-way automaton (A2A) of polynomial size corresponding to the LCPDL formula (since formulas from LCPDL have bounded intersection width). Though [13] uses an exponential sized alphabet (subsets of propositions), our alphabet is the (linear-sized) set of transitions Δ, ensuring that the transition relation has only polynomial size. We allow automata as path expressions, but it is straightforward to integrate them into the construction of the A2A. Finally, satisfiability checking amounts to emptiness checking of the A2A. Emptiness checking of A2A over words can be done in PSPACE (cf. [18, 19]). ◀

From Distributed Algorithms to LCPDL. Without loss of generality, we assume that Δ contains $\mathsf{t} = \langle \mathsf{s}\colon \mathbf{skip}\,;\mathbf{skip}\,;\mathbf{skip}\,;\mathbf{skip}\,;\mathbf{goto}\ s_0\rangle$ where $\mathsf{s} \neq s_0$ does not occur in any other transition.

Let $\mathcal{R} = (n : p_1, \ldots, p_n)$ be a ring and $\chi = C_0 \overset{t^1}{\rightsquigarrow} C_1 \overset{t^2}{\rightsquigarrow} \cdots \overset{t^k}{\rightsquigarrow} C_k$ be an $\mathcal{R}$-run of $\mathcal{D}$, where $t^j = (t^j_1, \ldots, t^j_n) \in \Delta^n$ for all $j \in [k]$. From χ, we extract the *symbolic run*

language. It is also important for the complexity since an automaton may be exponentially smaller than a regular expression.

$$loc^{0,1}_{r,r'} = \begin{cases} \{\bigwedge_{\bar{r}\in Reg} \neg\langle (msg^{0,1}_{\bar{r},r})^{-1}\rangle\}? & \text{if } r = r' \\ \{false\}? & \text{if } r \neq r' \end{cases} \qquad upd^{1,2}_{r,r'} = \begin{cases} \{\bigwedge_{\bar{r}\neq r} \neg(r := \bar{r})\}? & \text{if } r = r' \\ \{r' := r\}? & \text{if } r \neq r' \end{cases}$$

$$msg^{0,1}_{r,r'} = \left(\begin{array}{ll} & \{\mathbf{right}!r\}? \cdot (\hookrightarrow \cdot \{\mathbf{fwd}\}?)^* \cdot \hookrightarrow \cdot \{\mathbf{left}?r'\}? \\ + & \{\mathbf{left}!r\}? \cdot (\hookleftarrow \cdot \{\mathbf{fwd}\}?)^* \cdot \hookleftarrow \cdot \{\mathbf{right}?r'\}? \end{array} \right) \qquad next^{2,0}_{r,r'} = \begin{cases} \downarrow & \text{if } r = r' \\ \{false\}? & \text{if } r \neq r' \end{cases}$$

Figure 4 Path formulas to trace back transmission of pids.

$T_\chi = (n, k, \lambda) \in \Delta^{++}$ given by its columns $T_\chi[i] = \mathtt{t}\, t_i^1 \cdots t_i^k$. The purpose of the dummy transition $\mathtt{t}$ at the beginning of a column is to match the number of configurations in a run.

We will construct, in polynomial time, a formula $\psi_{\mathcal{D}} \in \mathrm{LCPDL}(\mathcal{D})$ such that $L(\psi_{\mathcal{D}}) = \{T_\chi \mid \chi \text{ is a run of } \mathcal{D}\}$. In particular, $\psi_{\mathcal{D}}$ will verify that (i) there are no cyclic dependencies that arise from $<$-guards, and (ii) registers in equality guards can be traced back to the same origin. In that case, the symbolic run is consistent and corresponds to a "real" run of $\mathcal{D}$.

The main ingredients of $\psi_{\mathcal{D}}$ are some path formulas that describe the transmission of pids in a symbolic run. They are depicted in Figure 4. For $\theta \in \{loc, msg, upd, next\}$ and $h \in \{0, 1, 2\}$, the meaning of $(x, y) \in [\![\theta^{h,h'}_{r,r'}]\!]_T$ is that the pid stored in r at *stage* h of position/transition x has been propagated to register r' at stage h' of y. Here, $h = 0$ means "after sending", $h = 1$ "after receiving", and $h = 2$ "after register update". The interpretation of "propagated" depends on θ. Formula $loc^{0,1}_{r,r'}$ says that the value of register r is not affected by reception. Similarly, $upd^{1,2}_{r,r'}$ takes care of updates. Formula $next^{2,0}_{r,r'}$ allows us to switch to the next transition of a process, preserving the value of $r(= r')$. The most interesting case is $msg^{0,1}_{r,r'}$, which describes paths across several processes. It relates the sending of r and a corresponding receive in r', which requires that all intermediate transitions are forward transitions. All path formulas are illustrated in Figure 3.

Since pids can be transmitted along several transitions and messages, the formulas $\theta^{h,h'}_{r,r'}$ will be composed by path automata. For $\mathrm{h} \in \{1, 2\}$ and $\mathrm{r} \in Reg$, we define a path automaton $\mathcal{A}^{\mathrm{h}}_{\mathrm{r}}$ that, in T_χ, connects some positions $(i, 0)$ and (i', j') iff, in χ, register r stores p_i at stage h of position (i', j'). Its set of states is $\iota \cup (\{0, 1, 2\} \times Reg)$. For all $r \in Reg$, there is a transition from the initial state ι to $(0, r)$ with transition label $\{\neg\langle\uparrow\rangle\}?$. Thus, the automaton starts at the top row and non-deterministically chooses some register r. From state (h, r), it can read any transition label $\theta^{h,h'}_{r,r'}$ and move to (h', r'). The only final state is (h, r). Figure 3 describes (partial) runs of $\mathcal{A}^1_{r'}$ and $\mathcal{A}^1_{r''}$, which allow us to identify the origin of r' and r'' when applying the guard $r' < r''$.

Now, consistency of equality guards can indeed be verified by an LCPDL formula. It says that, whenever an equality check $r = r'$ occurs in the symbolic run, then the pids stored in r and r' have a common origin. This can be conveniently expressed in terms of loop and converse. Note that guards are checked at stage $h = 1$ of the corresponding transition:

$$\psi_= \;=\; [(\rightarrow + \downarrow)^*] \bigwedge_{r,r'\in Reg} \Big(r = r' \;\Rightarrow\; \mathsf{loop}((\mathcal{A}^1_r)^{-1} \cdot \mathcal{A}^1_{r'}) \Big)\,.$$

The next path formula connects the first coordinate of a process i with the first coordinate of another process i' if some guard forces the pid of i to be smaller than that of i':

$$\pi_< \;= \Big(\textstyle\sum_{r,r'\in Reg} \mathcal{A}^1_r \cdot \{r < r'\}? \cdot (\mathcal{A}^1_{r'})^{-1} \Big)^+ .$$

Note that, here, we use the (strict) transitive closure. Consistency of $<$-guards now reduces to saying that there is no $\pi_<$-loop: $\psi_< \;=\; \neg\langle\rightarrow^*\rangle\mathsf{loop}(\pi_<)$.

$\widetilde{\mathsf{m}} = \neg\langle\leftarrow\rangle \qquad \widetilde{s} = \mathbf{goto}\ s \ \text{ for all } s \in S$

$\widetilde{\neg\varphi} = \neg\widetilde{\varphi} \quad \widetilde{\varphi_1 \wedge \varphi_2} = \widetilde{\varphi_1} \wedge \widetilde{\varphi_2} \quad \widetilde{\varphi_1 \Rightarrow \varphi_2} = \widetilde{\varphi_1} \Rightarrow \widetilde{\varphi_2} \quad \widetilde{[\pi]\varphi} = [\widetilde{\pi}]\widetilde{\varphi}$

$\widetilde{\langle\pi\rangle r < \langle\pi'\rangle r'} = \mathsf{loop}(\widetilde{\pi} \cdot (\mathcal{A}^2_r)^{-1} \cdot \pi_< \cdot \mathcal{A}^2_{r'} \cdot (\widetilde{\pi}')^{-1})$

$\widetilde{\langle\pi\rangle r \leq \langle\pi'\rangle r'} = \mathsf{loop}(\widetilde{\pi} \cdot (\mathcal{A}^2_r)^{-1} \cdot (\pi_< + \epsilon) \cdot \mathcal{A}^2_{r'} \cdot (\widetilde{\pi}')^{-1})$

$\widetilde{\langle\pi\rangle r = \langle\pi'\rangle r'} = \mathsf{loop}(\widetilde{\pi} \cdot (\mathcal{A}^2_r)^{-1} \cdot \mathcal{A}^2_{r'} \cdot (\widetilde{\pi}')^{-1})$

$\widetilde{\langle\pi\rangle r \neq \langle\pi'\rangle r'} = \mathsf{loop}(\widetilde{\pi} \cdot (\mathcal{A}^2_r)^{-1} \cdot (\leftarrow^+ + \rightarrow^+) \cdot \mathcal{A}^2_{r'} \cdot (\widetilde{\pi}')^{-1})$

$\widetilde{\pi}$ is inductively obtained from π by replacing tests $\{\varphi\}?$ by $\{\widetilde{\varphi}\}?$, $\rightarrow$ by $\hookrightarrow$, and $\leftarrow$ by $\hookleftarrow$

Figure 5 From DataPDL$^\ominus$ to LCPDL.

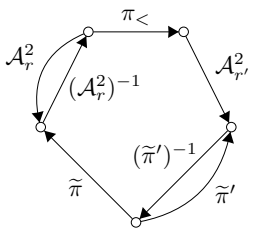


Figure 6 $\widetilde{\langle\pi\rangle r < \langle\pi'\rangle r'}$.

Finally, we can easily write an LCPDL formula ψ_{col} that checks whether every column $T[i] \in \Delta^+$ (ignoring $\mathtt{t}$) is a valid transition sequence of $\mathcal{D}$. Finally, let $\psi_{\mathcal{D}} = \psi_= \wedge \psi_< \wedge \psi_{\mathsf{col}}$.

▶ **Lemma 12.** *We have* $L(\psi_{\mathcal{D}}) = \{T_\chi \mid \chi \text{ is a run of } \mathcal{D}\}$.

From DataPDL$^\ominus$ to LCPDL. Next, we inductively translate every local DataPDL$^\ominus(\mathcal{D})$ formula φ into an LCPDL($\mathcal{D}$) formula $\widetilde{\varphi}$. The translation is given in Figure 5. As mentioned before, the first column in a table plays the role of a marked process so that $\widetilde{\mathsf{m}} = \neg\langle\leftarrow\rangle$. The standard formulas are translated as expected. Now, consider $\widetilde{\langle\pi\rangle r < \langle\pi'\rangle r'}$ (the remaining cases are similar). To "prove" $\langle\pi\rangle r < \langle\pi'\rangle r'$ at a given position in a symbolic run, we require that there are a $\widetilde{\pi}$-path and a $\widetilde{\pi}'$-path to coordinates x and x', respectively, whose registers r and r' satisfy $r < r'$. To guarantee the latter, the pids stored in r and r' have to go back to coordinates that are connected by a $\pi_<$-path. Again, using converse, this can be expressed as a loop (cf. Figure 6). Note that, hereby, $\mathcal{A}^2_r$ and $\mathcal{A}^2_{r'}$ refer to stage $h = 2$, which reflects the fact that DataPDL speaks about *configurations* (determined after updates).

▶ **Lemma 13.** *Let* $T \in \{T_\chi \mid \chi \text{ is a run of } \mathcal{D}\}$ *and* φ *be a local* DataPDL$^\ominus(\mathcal{D})$ *formula. We have* $T, (1,0) \models \widetilde{\varphi} \iff \big(\chi, 1, (1,0) \models \varphi \text{ for all runs } \chi \text{ of } \mathcal{D} \text{ such that } T_\chi = T\big)$.

Using Lemmas 12 and 13, we can now prove Lemma 14 below. Together with Theorem 11, the upper bound of Theorem 10 follows.

▶ **Lemma 14.** *Let* $\mathcal{D}$ *be a distributed algorithm,* $\Phi = \forall_{rings}\forall_{runs}\forall_{\mathsf{m}}\varphi \in$ DataPDL$^\ominus(\mathcal{D})$, *and* $b \geq 1$. *We have (a)* $\mathcal{D} \models \Phi \iff L(\psi_{\mathcal{D}} \wedge \neg\widetilde{\varphi}) = \emptyset$, *and (b)* $\mathcal{D} \models_b \Phi \iff L_b(\psi_{\mathcal{D}} \wedge \neg\widetilde{\varphi}) = \emptyset$.

5 Conclusion

In this paper, we provided a conceptually new approach to the verification of distributed algorithms that is robust against small changes of the model.

Actually, we made some assumptions that simplify the presentation, but are not crucial to the approach and results. For example, we assumed that an algorithm is synchronous, i.e., there is a global clock that, at every clock tick, triggers a round, in which every process participates. This can be relaxed to handle communication via (bounded) channels. Second, messages are pids, but they could contain message contents from a finite alphabet as well. Though the restriction to the class of rings is crucial for the complexity of our algorithm, the logical framework we developed is largely independent of concrete (ring) architectures. Essentially, we could choose any class of architectures for which LCPDL is decidable, for instance trees.

We leave open whether round-bounded model checking can deal with full DataPDL, or with properties of the form $\forall_{rings}\exists_{run}\forall_{\mathsf{m}}\varphi$, which are branching-time in spirit.

References

1 C. Aiswarya, B. Bollig and P. Gastin. An Automata-Theoretic Approach to the Verification of Distributed Algorithms. *CoRR*, abs/1504.06534. 2015.

2 R. Alur and P. Černý. Streaming transducers for algorithmic verification of single-pass list-processing programs. In *POPL'11*, pages 599–610. ACM, 2011.

3 B. Aminof, S. Jacobs, A. Khalimov, and S. Rubin. Parameterized model checking of token-passing systems. In *VMCAI'14*, volume 8318 of *LNCS*, pages 262–281, 2014.

4 M. Bojanczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3), 2009.

5 B. Bollig, A. Cyriac, P. Gastin, and K. Narayan Kumar. Model checking languages of data words. In *FoSSaCS'12*, volume 7213 of *LNCS*, pages 391–405. Springer, 2012.

6 M. Chaouch-Saad, B. Charron-Bost, and S. Merz. A reduction theorem for the verification of round-based distributed algorithms. In *RP'09*, volume 5797 of *LNCS*, pages 93–106. Springer, 2009.

7 D. Dolev, M. M. Klawe, and M. Rodeh. An O(n log n) unidirectional distributed algorithm for extrema finding in a circle. *J. Algorithms*, 3(3):245–260, 1982.

8 E. A. Emerson and V. Kahlon. Parameterized Model Checking of Ring-Based Message Passing Systems. In *CSL'05*, volume 3210 of *LNCS*, pages 325-339. Springer, 2004.

9 E. A. Emerson and K. S. Namjoshi. On reasoning about rings. *Int. J. Found. Comput. Sci.*, 14(4):527–550, 2003.

10 J. Esparza. Keeping a crowd safe: On the complexity of parameterized verification. In *STACS'14*, volume 25 of *LIPIcs*, pages 1–10, 2014.

11 D. Figueira and L. Segoufin. Bottom-up automata on data trees and vertical XPath. In *STACS'11*, volume 9 of *LIPIcs*, pages 93–104, 2011.

12 R. Franklin. On an improved algorithm for decentralized extrema finding in circular configurations of processors. *Commun. ACM*, 25(5):336–337, 1982.

13 S. Göller, M. Lohrey, and C. Lutz. PDL with intersection and converse: satisfiability and infinite-state model checking. *J. Symb. Log.*, 74(1):279–314, 2009.

14 I. Konnov, H. Veith, and J. Widder. Who is afraid of model checking distributed algorithms? In *CAV'12 Workshop (EC)^2*. 2012.

15 I. Konnov, H. Veith, and J. Widder. On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. In *CONCUR'14*, volume 8704 of *LNCS*, pages 125–140. Springer, 2014.

16 M. Lange. Model checking propositional dynamic logic with all extras. *J. Applied Logic*, 4(1):39–49, 2006.

17 N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.

18 R. Mennicke. Propositional Dynamic Logic with Converse and Repeat for Message-Passing Systems. *LMCS* 9(2:12) 2013.

19 O. Serre. Parity Games Played on Transition Graphs of One-Counter Processes. In *FOSSACS'06*, LNCS, pages 337–351. Springer, 2006.

20 M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency*, volume 1043 of *LNCS*, pages 238–266. Springer, 1996.

21 M. Y. Vardi. Reasoning about the past with two-way automata. In *ICALP'98*, LNCS, pages 628–641. Springer, 1998.

Lazy Probabilistic Model Checking without Determinisation

Ernst Moritz Hahn[1], Guangyuan Li[1], Sven Schewe[2], Andrea Turrini[1], and Lijun Zhang[1]

1 **State Key Laboratory of Computer Science, Institute of Software, CAS, China**
2 **University of Liverpool, UK**

Abstract

The bottleneck in the quantitative analysis of Markov chains and Markov decision processes against specifications given in LTL or as some form of nondeterministic Büchi automata is the inclusion of a determinisation step of the automaton under consideration. In this paper, we show that full determinisation can be avoided: subset and breakpoint constructions suffice. We have implemented our approach – both explicit and symbolic versions – in a prototype tool. Our experiments show that our prototype can compete with mature tools like PRISM.

1998 ACM Subject Classification G.3 Probability and Statistics, D.2.4 Software/Program Verification

Keywords and phrases Markov Decision Processes, Model Checking, PLTL, Determinisation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.354

1 Introduction

Markov chains (MCs) and Markov decision processes (MDPs) are widely used to study systems that exhibit both, probabilistic and nondeterministic choices. Properties of these systems are often specified by temporal logic formulas, such as the branching time logic PCTL [11], the linear time logic PLTL [3], or their combination PCTL* [3]. While model checking is tractable for PCTL [3], it is more expensive for PLTL: PSPACE-complete for Markov chains and 2EXPTIME-complete for MDPs [6].

In classical model checking, one checks whether a model $\mathcal{M}$ satisfies an LTL formula φ by first constructing a nondeterministic Büchi automaton $\mathcal{B}_{\neg\varphi}$ [20], which recognises the models of its negation $\neg\varphi$. The model checking problem then reduces to an emptiness test for the product $\mathcal{M} \otimes \mathcal{B}_{\neg\varphi}$. The translation to Büchi automata may result in an exponential blow-up compared to the length of φ. However, this translation is mostly very efficient in practice, and highly optimised off-the-shelf tools like LTL3BA [1] or SPOT [7] are available.

The quantitative analysis of a probabilistic model $\mathcal{M}$ against an LTL specification φ is more involved. To compute the maximal probability $\mathfrak{P}^{\mathcal{M}}(\varphi)$ that φ is satisfied in $\mathcal{M}$, the classic automata-based approach includes the determinisation of an intermediate Büchi automaton $\mathcal{B}_{\varphi}$. If such a deterministic automaton $\mathcal{A}$ is constructed for $\mathcal{B}_{\varphi}$, then determining the probability $\mathfrak{P}^{\mathcal{M}}(\varphi)$ reduces to solving an equation system for Markov chains, and a linear programming problem for MDPs [3], both in the product $\mathcal{M} \otimes \mathcal{A}$. Such a determinisation step usually exploits a variant of Safra's [17] determinisation construction, such as the techniques presented in [16, 18].

Kupferman, Piterman, and Vardi point out in [14] that "Safra's determinization construction has been notoriously resistant to efficient implementations." Even though analysing long LTL formulas would surely be useful as they allow for the description of more complex

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 354–367

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

requirements on a system's behaviour, model checkers that employ determinisation to support LTL, such as LiQuor [5] or PRISM [15], might fail to verify such properties.

In this paper we argue that applying the Safra determinisation step in full generality is only required in some cases, while simpler subset and breakpoint constructions often suffice. Moreover, where full determinisation is required, it can be replaced by a combination of the simpler constructions, and it suffices to apply it locally on a small share of the places.

A subset construction is known to be sufficient to determinise finite automata, but it fails for Büchi automata. Our first idea is to construct an under- and an over-approximation starting from the subset construction. That is, we construct two (deterministic) subset automata $\mathcal{S}^u$ and $\mathcal{S}^o$ such that $\mathcal{L}(\mathcal{S}^u) \subseteq \mathcal{L}(\mathcal{B}_\varphi) \subseteq \mathcal{L}(\mathcal{S}^o)$ where $\mathcal{L}(\mathcal{B}_\varphi)$ denotes the language defined by the automaton $\mathcal{B}_\varphi$ for φ. The subset automata $\mathcal{S}^u$ and $\mathcal{S}^o$ are the same automaton $\mathcal{S}$ except for their accepting conditions. We build a product Markov chain with the subset automata. We establish the useful property that the probability $\mathfrak{P}^{\mathcal{M}}(\varphi)$ equals the probability of reaching some *accepting* bottom strongly connected components (SCCs) in this product: for each bottom SCC $\mathtt{S}$ in the product, we can first use the accepting conditions in $\mathcal{S}^u$ or $\mathcal{S}^o$ to determine whether $\mathtt{S}$ is accepting or rejecting, respectively. The challenge remains when the test is inconclusive. In this case, we first refine $\mathtt{S}$ using a breakpoint construction. Finally, if the breakpoint construction fails as well, we have two options: we can either perform a Rabin-based determinisation for the part of the model where it is required, thus avoiding to construct the larger complete Rabin product. Alternatively, a refined multi-breakpoint construction is used. An important consequence is that we no longer need to implement a Safra-style determinisation procedure: subset and breakpoint constructions are enough. From a theoretical point of view, this reduces the cost of the automata transformations involved from $n^{\mathcal{O}(k \cdot n)}$ to $\mathcal{O}(k \cdot 3^n)$ for generalised Büchi automata with n states and k accepting sets. From a practical point of view, the easy symbolic encoding admitted by subset and breakpoint constructions is of equal value. We discuss that (and how) the framework can be adapted to MDPs– with the same complexity – by analysing the end components [6, 3].

We have implemented our approach, both explicit and symbolic versions, in our IscasMC tool [10], which we applied on various Markov chain and MDP case studies (for space reasons we report on only two in this paper, cf. [9] for others). Our experimental results confirm that our new algorithm outperforms the Rabin-based approach in most of the properties considered. However, there are some cases in which the Rabin determinisation approach performs better when compared to the multi-breakpoint construction: the construction of a single Rabin automaton suffices to decide a given connected component, while the breakpoint construction may require several iterations. Our experiments also show that our prototype can compete with mature tools like PRISM.

Due to the lack of space, detailed proofs and additional case studies are provided in [9].

2 Preliminaries

2.1 ω-Automata

Nondeterministic Büchi automata are used to represent ω-regular languages $\mathcal{L} \subseteq \Sigma^\omega = \omega \to \Sigma$ over a finite alphabet Σ. In this paper, we use automata with trace-based acceptance mechanisms. We denote by $[1..k]$ the set $\{1, 2, \ldots, k\}$ and by $j \oplus_k 1$ the successor of j in $[1..k]$. I.e., $j \oplus_k 1 = j + 1$ if $j < k$ and $j \oplus_k 1 = 1$ if $j = k$.

▶ **Definition 1.** A *nondeterministic generalised Büchi automaton* (NGBA) is a quintuple $\mathcal{B} = (\Sigma, Q, I, \mathrm{T}, \mathbf{F}_k)$, consisting of a finite alphabet Σ of input letters, a finite set Q of states

with a non-empty subset $I \subseteq Q$ of initial states, a set $\mathrm{T} \subseteq Q \times \Sigma \times Q$ of transitions from states through input letters to successor states, and a family $\mathbf{F}_k = \{ \mathrm{F}_j \subseteq \mathrm{T} \mid j \in [1..k] \}$ of accepting (final) sets.

Nondeterministic Büchi automata are interpreted over infinite sequences $\alpha\colon \omega \to \Sigma$ of input letters. An infinite sequence $\rho\colon \omega \to Q$ of states of $\mathcal{B}$ is called a *run* of $\mathcal{B}$ on an input word α if $\rho(0) \in I$ and, for each $i \in \omega$, $\big(\rho(i), \alpha(i), \rho(i+1)\big) \in \mathrm{T}$. We denote by $\mathrm{Run}(\alpha)$ the set of all runs ρ on α. For a run $\rho \in \mathrm{Run}(\alpha)$, we denote with $\mathrm{tr}(\rho)\colon i \mapsto \big(\rho(i), \alpha(i), \rho(i+1)\big)$ the *transitions of* ρ. We sometimes denote a run ρ by the associated states, that is, $\rho = q_0 \cdot q_1 \cdot q_2 \cdot \ldots$ where $\rho(i) = q_i$ for each $i \in \omega$ and we call a finite prefix $q_0 \cdot q_1 \cdot q_2 \cdot \ldots \cdot q_n$ of ρ a *pre-run*. A run ρ of a NGBA is *accepting* if its transitions $\mathrm{tr}(\rho)$ contain infinitely many transitions from all final sets, i.e., for each $j \in [1..k]$, $\mathrm{Inf}(\mathrm{tr}(\rho)) \cap \mathrm{F}_j \neq \emptyset$, where $\mathrm{Inf}(\mathrm{tr}(\rho)) = \{ t \in \mathrm{T} \mid \forall i \in \omega\ \exists j > i$ such that $\mathrm{tr}(\rho)(j) = t \}$. A word $\alpha\colon \omega \to \Sigma$ is *accepted* by $\mathcal{B}$ if $\mathcal{B}$ has an accepting run on α, and the set $\mathcal{L}(\mathcal{B}) = \{ \alpha \in \Sigma^\omega \mid \alpha$ is accepted by $\mathcal{B} \}$ of words accepted by $\mathcal{B}$ is called its *language*.

Figure 1 shows an example of Büchi automaton. The number j after the label as in the transition (x, a, y), when present, indicates that the transition belongs to the accepting set F_j, i.e., (x, a, y) belongs to F_1. The language generated by $\mathcal{B}_\mathcal{E}$ is a subset of $(ab|ac)^\omega$ and a word α is accepted if each b (and c) is eventually followed by a c (by a b, respectively).

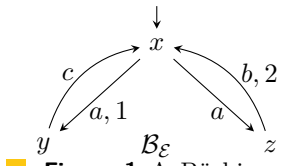


Figure 1 A Büchi automaton.

We call the automaton $\mathcal{B}$ a *nondeterministic Büchi automaton* (NBA) whenever $|\mathbf{F}_k| = 1$ and we denote it by $\mathcal{B} = (\Sigma, Q, I, \mathrm{T}, \mathrm{F})$. For technical convenience we also allow for finite runs $q_0 \cdot q_1 \cdot q_2 \cdot \ldots \cdot q_n$ with $\mathrm{T} \cap \{q_n\} \times \{\alpha(n)\} \times Q = \emptyset$. In other words, a run may end with q_n if action $\alpha(n)$ is not enabled from q_n. Naturally, no finite run satisfies the accepting condition, thus it is not accepting and has no influence on the language of an automaton.

To simplify the notation, the transition set T can also be seen as a function $\mathrm{T}\colon Q \times \Sigma \to 2^Q$ assigning to each pair $(q, \sigma) \in Q \times \Sigma$ the set of successors according to T, i.e., $\mathrm{T}(q, \sigma) = \{ q' \in Q \mid (q, \sigma, q') \in \mathrm{T} \}$. We extend T to sets of states in the usual way, i.e., by defining $\mathrm{T}(S, \sigma) = \bigcup_{q \in S} \mathrm{T}(q, \sigma)$.

▶ **Definition 2.** A *(transition-labelled) nondeterministic Rabin automaton* (NRA) with k accepting pairs is a quintuple $\mathcal{A} = (\Sigma, Q, I, \mathrm{T}, (\mathbf{A}_k, \mathbf{R}_k))$ where Σ, Q, I, and T are as in Definition 1 and $(\mathbf{A}_k, \mathbf{R}_k) = \{ (\mathrm{A}_i, \mathrm{R}_i) \mid i \in [1..k],\ \mathrm{A}_i, \mathrm{R}_i \subseteq \mathrm{T} \}$ is a finite family of Rabin pairs. (For convenience, we sometimes use other finite sets of indices rather than $[1..k]$.)

A run ρ of a NRA is accepting if there exists $i \in [1..k]$ such that $\mathrm{Inf}(\mathrm{tr}(\rho)) \cap \mathrm{A}_i \neq \emptyset$ and $\mathrm{Inf}(\mathrm{tr}(\rho)) \cap \mathrm{R}_i = \emptyset$.

An automaton $\mathcal{A} = (\Sigma, Q, I, \mathrm{T}, \mathbf{ACC})$, where $\mathbf{ACC}$ is the acceptance condition (Rabin, Büchi, or generalised Büchi), is called *deterministic* if, for each $(q, \sigma) \in Q \times \Sigma$, $|\mathrm{T}(q, \sigma)| \leq 1$, and $I = \{q_0\}$ for some $q_0 \in Q$. For notational convenience, we denote a deterministic automaton $\mathcal{A}$ by the tuple $(\Sigma, Q, q_0, \mathrm{T}, \mathbf{ACC})$ and $\mathrm{T}\colon Q \times \Sigma \to Q$ is the partial function, which is defined at (q, σ) if, and only if, σ is enabled at q. For a given deterministic automaton $\mathcal{D}$, we denote by $\mathcal{D}_d$ the otherwise similar automaton with initial state d. Similarly, for a NGBA $\mathcal{B}$, we denote by $\mathcal{B}_R$ the NGBA with R as set of initial states.

2.2 Markov Chains and Product

A *distribution* μ over a set X is a function $\mu\colon X \to [0, 1]$ such that $\sum_{x \in X} \mu(x) = 1$. A *Markov chain (MC)* is a tuple $\mathcal{M} = (M, L, \mu_0, \mathrm{P})$, where M is a finite set of states, $L\colon M \to \Sigma$ is

a *labelling function*, μ_0 is the *initial distribution*, and $\mathrm{P}\colon M \times M \to [0,1]$ is a *probabilistic transition matrix* satisfying $\sum_{m' \in M} \mathrm{P}(m,m') \in \{0,1\}$ for all $m \in M$. A state m is called *absorbing* if $\sum_{m' \in M} \mathrm{P}(m,m') = 0$. We write $(m,m') \in \mathrm{P}$ for $\mathrm{P}(m,m') > 0$.

A *maximal path* of $\mathcal{M}$ is an infinite sequence $\xi = m_0 m_1 \ldots$ satisfying $\mathrm{P}(m_i, m_{i+1}) > 0$ for all $i \in \omega$, or a finite one if the last state is absorbing. We denote by $Paths^{\mathcal{M}}$ the set of all maximal paths of $\mathcal{M}$. An infinite path $\xi = m_0 m_1 \ldots$ defines the word $\alpha(\xi) = w_0 w_1 \ldots \in \Sigma^\omega$ with $w_i = L(m_i)$, $i \in \omega$.

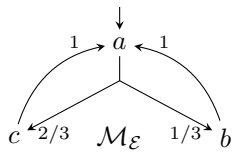


Figure 2 A MC with $L(m) = m$ for each m.

Given a finite sequence $\xi = m_0 m_1 \ldots m_k$, the *cylinder* of ξ, denoted by $Cyl(\xi)$, is the set of maximal paths starting with prefix ξ. We define the probability of the cylinder set by $\mathfrak{P}^{\mathcal{M}}\big(Cyl(m_0 m_1 \ldots m_k)\big) \stackrel{\text{def}}{=} \mu_0(m_0) \cdot \prod_{i=0}^{k-1} \mathrm{P}(m_i, m_{i+1})$. For a given MC $\mathcal{M}$, $\mathfrak{P}^{\mathcal{M}}$ can be uniquely extended to a probability measure over the σ-algebra generated by all cylinder sets.

In this paper we are interested in ω-regular properties $\mathcal{L} \subseteq \Sigma^\omega$ and the probability $\mathfrak{P}^{\mathcal{M}}(\mathcal{L})$ for some measurable set $\mathcal{L}$. Further, we define $\mathfrak{P}^{\mathcal{M}}(\mathcal{B}) \stackrel{\text{def}}{=} \mathfrak{P}^{\mathcal{M}}(\{\, \xi \in Paths^{\mathcal{M}} \mid \alpha(\xi) \in \mathcal{L}(\mathcal{B}) \,\})$ for an automaton $\mathcal{B}$. We write $\mathfrak{P}^{\mathcal{M}}_m$ to denote the probability function when assuming that m is the initial state. Moreover, we omit the superscript $\mathcal{M}$ whenever it is clear from the context. We follow the standard way of computing this probability in the product of $\mathcal{M}$ and a deterministic automaton for $\mathcal{L}$.

▶ **Definition 3.** Given a MC $\mathcal{M} = (M, L, \mu_0, \mathrm{P})$ and a deterministic automaton $\mathcal{A} = (\Sigma, Q, q_0, \mathrm{T}, \mathbf{ACC})$, the *product Markov chain* is defined by $\mathcal{M} \times \mathcal{A} \stackrel{\text{def}}{=} (M \times Q, L', \mu_0', \mathrm{P}')$ where $L'\big((m,d)\big) = L(m)$; $\mu_0'\big((m,d)\big) = \mu_0(m)$ if $d = \mathrm{T}(q_0, L(m))$, 0 otherwise; and $\mathrm{P}'\big((m,d),(m',d')\big)$ equals $\mathrm{P}(m,m')$ if $d' = \mathrm{T}(d, L(m'))$, and is 0 otherwise.

We denote by $\pi_{\mathcal{A}}((m,d),(m',d'))$ the *projection* on $\mathcal{A}$ of the given $((m,d),(m',d')) \in \mathrm{P}'$, i.e., $\pi_{\mathcal{A}}((m,d),(m',d')) = (d, L(m'), d')$, and by $\pi_{\mathcal{A}}(\mathrm{B})$ its extension to a set of transitions $\mathrm{B} \subseteq \mathrm{T}'$, i.e., $\pi_{\mathcal{A}}(\mathrm{B}) = \{\, \pi_{\mathcal{A}}(p,p') \mid (p,p') \in \mathrm{B} \,\}$.

As we have accepting transitions on the edges of the automata, we propose product Markov chains with accepting conditions on their edges.

▶ **Definition 4.** Given a MC $\mathcal{M}$ and a deterministic automaton $\mathcal{A}$ with accepting set $\mathbf{ACC}$, the product automaton is $\mathcal{M} \otimes \mathcal{A} \stackrel{\text{def}}{=} (\mathcal{M} \times \mathcal{A}, \mathbf{ACC}')$ where

- if $\mathbf{ACC} = \mathbf{F}_k$, then $\mathbf{ACC}' \stackrel{\text{def}}{=} \mathbf{F}'_k$ where $\mathrm{F}'_i = \{\, (p,p') \in \mathrm{P}' \mid \pi_{\mathcal{A}}(p,p') \in \mathrm{F}_i \,\} \in \mathbf{F}'_k$ for each $i \in [1..k]$ (Generalised Büchi Markov chain, GMC); and
- if $\mathbf{ACC} = (\mathbf{A}_k, \mathbf{R}_k)$, then $\mathbf{ACC}' \stackrel{\text{def}}{=} (\mathbf{A}'_k, \mathbf{R}'_k)$ where $\mathrm{A}'_i = \{\, (p,p') \in \mathrm{P}' \mid \pi_{\mathcal{A}}(p,p') \in \mathrm{A}_i \,\} \in \mathbf{A}'_k$ and $\mathrm{R}'_i = \{\, (p,p') \in \mathrm{P}' \mid \pi_{\mathcal{A}}(p,p') \in \mathrm{R}_i \,\} \in \mathbf{R}'_k$ for each $i \in [1..k]$ (Rabin Markov chain, RMC).

Thus, RMC and GMC are Markov chains extended with the corresponding accepting conditions. We remark that the labelling of the initial states of the Markov chain is taken into account in the definition of μ_0'.

▶ **Definition 5.** A *bottom strongly connected component* (BSCC) $\mathtt{S} \subseteq V$ is an SCC in the underlying digraph (V,E) of a MC $\mathcal{M}$, where all edges with source in $\mathtt{S}$ have only successors in $\mathtt{S}$ (i.e., for each $(v,v') \in E$, $v \in \mathtt{S}$ implies $v' \in \mathtt{S}$). We assume that a (bottom) SCC does not contain any absorbing state. Given an SCC $\mathtt{S}$, we denote by $\mathrm{P}_{\mathtt{S}}$ the transitions of $\mathcal{M}$ in $\mathtt{S}$, i.e., $\mathrm{P}_{\mathtt{S}} = \{\, (m,m') \in \mathrm{P} \mid m,m' \in \mathtt{S} \,\}$.

3 Lazy Determinisation

We fix an input MC $\mathcal{M}$ and a NGBA $\mathcal{B} = (\Sigma, Q, I, \mathrm{T}, \mathbf{F}_k)$ as a specification. Further, let $\mathcal{A} = \det(\mathcal{B})$ be the deterministic Rabin automaton (DRA) constructed for $\mathcal{B}$ (cf. [17, 18, 19]), and let $\mathcal{M} \otimes \mathcal{A} = (M \times Q, L, \mu_0, \mathrm{P}, \mathbf{ACC})$ be the product RMC. We consider the problem of computing $\mathfrak{P}^{\mathcal{M}}_{m_0}(\mathcal{B})$, i.e., the probability that a run of $\mathcal{M}$ is accepted by $\mathcal{B}$.

3.1 Outline of our Methodology

We first recall the classical approach for computing $\mathfrak{P}^{\mathcal{M}}(\mathcal{B})$, see [2] for details. It is well known [3] that the computation of $\mathfrak{P}^{\mathcal{M}}(\mathcal{B})$ reduces to the computation of the probabilistic reachability in the product RMC $\mathcal{M} \otimes \mathcal{A}$ with $\mathcal{A} = \det(\mathcal{B})$. We first introduce the notion of accepting SCCs:

▶ **Definition 6.** Given a MC $\mathcal{M}$ and the DRA $\mathcal{A} = \det(\mathcal{B})$, let $\mathtt{S}$ be a bottom SCC of the product RMC $\mathcal{M} \otimes \mathcal{A}$. We say that $\mathtt{S}$ is accepting if there exists an index $i \in [1..k]$ such that $\mathrm{A}_i \cap \pi_{\mathcal{A}}(\mathrm{P}_{\mathtt{S}}) \neq \emptyset$ and $\mathrm{R}_i \cap \pi_{\mathcal{A}}(\mathrm{P}_{\mathtt{S}}) = \emptyset$; we call each $s \in \mathtt{S}$ an *accepting state*. Moreover, we call the union of all accepting BSCCs the *accepting region*.

Essentially, since a BSCC is an ergodic set, once a path enters an accepting BSCC $\mathtt{S}$, with probability 1 it will take transitions from A_i infinitely often; since A_i is finite, at least one transition from A_i is taken infinitely often. Now we have the following reduction:

▶ **Theorem 7 (**3**).** *Given a MC $\mathcal{M}$ and a Büchi automaton $\mathcal{B}$, consider $\mathcal{A} = \det(\mathcal{B})$. Let $U \subseteq M \times Q$ be the accepting region and let $\Diamond U$ denote the set of paths containing a state of U. Then, $\mathfrak{P}^{\mathcal{M}}(\mathcal{B}) = \mathfrak{P}^{\mathcal{M}\otimes\mathcal{A}}(\Diamond U)$.*

When all bottom SCCs are evaluated, the evaluation of the Rabin MC is simple: we abstract all accepting bottom SCCs to an absorbing goal state and perform a reachability analysis, which can be solved in polynomial time [3, 2]. Thus, the outline of the traditional probabilistic model checking approach for LTL specifications is as follows: (1.) translate the NGBA $\mathcal{B}$ into an equivalent DRA $\mathcal{A} = \det(\mathcal{B})$; (2.) build (the reachable fragment of) the product automaton $\mathcal{M} \otimes \mathcal{A}$; (3.) for each BSCC $\mathtt{S}$, check whether $\mathtt{S}$ is accepting. Let U be the union of these accepting SCCs; (4.) infer the probability $\mathfrak{P}^{\mathcal{M}\otimes\mathcal{A}}(\Diamond U)$.

The construction of the deterministic Rabin automaton used in the classical approach is often the bottleneck of the approach, as one exploits some variant of the approach proposed by Safra [17], which is rather involved. The lazy determinisation technique we suggest in this paper follows a different approach. We first transform the high-level specification (e.g., given in the PRISM language [15]) into its MDP or MC semantics. We then employ some tool (e.g., LTL3BA [1] or SPOT [7]) to construct a Büchi automaton equivalent to the LTL specification. This nondeterministic automaton is used to obtain the deterministic Büchi over- and under-approximation subset automata $\mathcal{S}^u$ and $\mathcal{S}^o$, as described in Subsection 3.3. The languages recognised by these two deterministic Büchi automata are such that $\mathcal{L}(\mathcal{S}^u) \subseteq \mathcal{L}(\mathcal{B}) \subseteq \mathcal{L}(\mathcal{S}^o)$. We build the product of these subset automata with the model MDP or MC (cf. Lemma 13). We then compute the maximal end components or bottom strongly connected components. According to Lemma 14, we try to decide these components of the product by using the acceptance conditions F^o_i and F^u_i of $\mathcal{S}^u$ and $\mathcal{S}^o$, respectively.

For each of those components where over- and under-approximation do not agree (and which we therefore cannot decide), we employ the breakpoint construction (cf. Corollary 16), involving the deterministic Rabin over- and under-approximation breakpoint automata $\mathcal{BP}^u$ and $\mathcal{BP}^o$, such that $\mathcal{L}(\mathcal{BP}^u) \subseteq \mathcal{L}(\mathcal{B}) \subseteq \mathcal{L}(\mathcal{BP}^o)$. For this, we take one state of the component

under consideration and start the breakpoint construction with this state as initial state. This way, we obtain a product of a breakpoint automaton with parts of the model. If the resulting product contains an accepting component (using the under-approximation), then the original component must be accepting, and if the resulting product contains a rejecting component (using the over-approximation), then the original component must be rejecting.

The remaining undecided components are decided either by using a Rabin-based construction, restricted to the undecided component, or only by using $\mathcal{BP}^u$, where we start from possibly different states of the subset product component under consideration; this approach always decides the remaining components, and we call it the multi-breakpoint construction.

For the model states that are part of an accepting component, or from which no accepting component is reachable, the probability to fulfil the specification is now already known to be 1 or 0, respectively. To obtain the remaining state probabilities, we construct and solve a linear programming (LP) problem (or a linear equation system when we start with MCs).

Note that, even in case the multi-breakpoint procedure is necessary in some places, our method is usually still more efficient than direct Rabin determinisation, for instance based on some variation of [19]. The reason for this is twofold. First, when starting the determinisation procedure from a component rather than from the initial state of the model, the number of states in the Rabin product will be smaller, and second, we only need the multi-breakpoint determinisation to decide MECs or bottom SCCs, such that the computation of transient probabilities can still be done in the smaller subset product.

In the remainder of this section, we detail the proposed approach: we first introduce the theoretical background, and then present the incremental evaluation of the bottom SCCs.

3.2 Acceptance Equivalence

In order to be able to apply our lazy approach, we exploit a number of acceptance equivalences in the RMC. Given the DRA $\mathcal{A} = \det(\mathcal{B})$ and a state d of $\mathcal{A}$, we denote by $\mathsf{rchd}(d)$ the label of the root node ε of the labelled ordered tree associated to d (cf. [17, 18, 19]).

▶ **Proposition 8.** *Given a NGBA $\mathcal{B}$, a MC $\mathcal{M}$, and the DRA $\mathcal{A} = \det(\mathcal{B})$, (1.) a path ρ in $\mathcal{M} \otimes \mathcal{A}$ that starts from a state (m, d) is accepted if, and only if, the word it defines is accepted by $\mathcal{B}_{\mathsf{rchd}(d)}$; and (2.) if $\mathsf{rchd}(d) = \mathsf{rchd}(d')$, then the probabilities of acceptance from a state (m, d) and a state (m, d') are equal, i.e., $\mathfrak{P}^{\mathcal{M}\otimes\mathcal{A}}_{(m,d)}(\mathcal{B}) = \mathfrak{P}^{\mathcal{M}\otimes\mathcal{A}}_{(m,d')}(\mathcal{B})$.*

This property allows us to work on quotients *and* to swap between states with the same reachability set. If we ignore the accepting conditions, we have a product MC, and we can consider the quotient of such a product MC as follows.

▶ **Definition 9** (Quotient MC). Given a MC $\mathcal{M}$ and a DRA $\mathcal{A} = \det(\mathcal{B})$, the *quotient MC* $[\mathcal{M} \times \mathcal{A}]$ of $\mathcal{M} \times \mathcal{A}$ is the MC $([M \times Q], [L], [\mu_0], [\mathrm{P}])$ where

- $[M \times Q] = \{\, (m, [d]) \mid (m, d) \in M \times Q,\ [d] = \{\, d' \in Q \mid \mathsf{rchd}(d') = \mathsf{rchd}(d) \,\} \,\}$,
- $[L](m, [d]) = L(m, d)$,
- $[\mu_0](m, [d]) = \mu_0(m, d)$, and
- $[\mathrm{P}]\big((m, [d]), (m', [d'])\big) = \mathrm{P}\big((m, d), (m', d')\big)$.

By abuse of notation, we define $[(m, d)] = (m, [d])$ and $[C] = \{\, [s] \mid s \in C \,\}$. It is easy to see that, for each $d \in Q$, $d \in [d]$ holds and that [P] is well defined: for $(m, d_1), (m, d_2) \in [(m, d)]$, $\mathrm{P}\big((m, d_1), (m', [d'])\big) = \mathrm{P}\big((m, d), (m', d')\big) = \mathrm{P}\big((m, d_2), (m', [d'])\big)$ holds.

▶ **Theorem 10.** *For a MC $\mathcal{M}$ and DRA $\mathcal{A} = \det(\mathcal{B})$, it holds that*

1. *if $\mathtt{S}$ is a bottom SCC of $\mathcal{M} \times \mathcal{A}$ then $[\mathtt{S}]$ is a bottom SCC of $[\mathcal{M} \times \mathcal{A}]$,*
2. *if $\mathtt{S}'$ is a bottom SCC of $[\mathcal{M} \times \mathcal{A}]$, then there is a bottom SCC $\mathtt{S}$ of $\mathcal{M} \times \mathcal{A}$ with $\mathtt{S}' = [\mathtt{S}]$.*

Together with Definition 6 and Proposition 8, Theorem 10 provides:

▶ **Corollary 11.** *Let* $\mathtt{S}$ *be a bottom SCC of* $[\mathcal{M} \times \mathcal{A}]$*. Then, either all states* s *of* $\mathcal{M} \otimes \mathcal{A}$ *with* $[s] \in \mathtt{S}$ *are accepting, or all states* s *of* $\mathcal{M} \otimes \mathcal{A}$ *with* $[s] \in \mathtt{S}$ *are rejecting.*

Once all bottom SCCs are evaluated, we only need to perform a standard probabilistic reachability analysis on the quotient MC.

3.3 Incremental Evaluation of Bottom SCCs

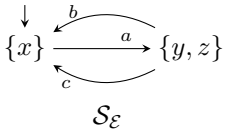
To evaluate each bottom SCC of the RMC, we use three techniques: the first one is based on evaluating the subset construction directly. We get two deterministic NGBAs that provide over- and under-approximations. If this fails, we refine the corresponding bottom SCC by a breakpoint construction. Only if both fail, a precise construction follows.

3.3.1 Subset Construction

For a given NGBA $\mathcal{B} = (\Sigma, Q, I, \mathrm{T}, \mathbf{F}_k)$, a simple way to over- and under-approximate its language by a subset construction is as follows. We build two NGBAs $\mathcal{S}^o = (\Sigma, 2^Q, \{I\}, \mathrm{T}', \mathbf{F}^o_k)$ and $\mathcal{S}^u = (\Sigma, 2^Q, \{I\}, \mathrm{T}', \mathbf{F}^u_k)$, differing only for the accepting condition, where

- $\mathrm{T}' = \{\, (R, \sigma, C) \mid \emptyset \neq R \subseteq Q, C = \mathrm{T}(R, \sigma) \,\}$,
- $\mathrm{F}^o_i = \{\, (R, \sigma, C) \in \mathrm{T}' \mid \exists (q, q') \in R \times C.\ (q, \sigma, q') \in \mathrm{F}_i \,\} \in \mathbf{F}^o_k$ for each $i \in [1..k]$, and
- $\mathrm{F}^u_i = \{\, (R, \sigma, C) \in \mathrm{T}' \mid \forall (q, q') \in R \times C.\ (q, \sigma, q') \in \mathrm{F}_i \,\} \in \mathbf{F}^u_k$ for each $i \in [1..k]$.

Essentially, $\mathcal{S}^o$ and $\mathcal{S}^u$ are the subset automata that we use to over- and under-approximate the accepting conditions, respectively. Figure 3 shows the reachable fragment of the subset construction for the NGBA $\mathcal{B}_\mathcal{E}$ depicted in Figure 1. The final sets of the two subset automata are $\mathrm{F}^o_1 = \{(\{x\}, a, \{yz\})\}$ and $\mathrm{F}^o_2 = \{(\{yz\}, b, \{x\})\}$ for $\mathcal{S}^o$ and $\mathrm{F}^u_1 = \mathrm{F}^u_2 = \emptyset$ for $\mathcal{S}^u$. The following lemma holds:



$\mathcal{S}_\mathcal{E}$

Figure 3 The subset construction for $\mathcal{B}_\mathcal{E}$.

▶ **Lemma 12.** $\mathcal{L}(\mathcal{S}^u_{[d]}) \subseteq \mathcal{L}(\mathcal{A}_d) \subseteq \mathcal{L}(\mathcal{S}^o_{[d]})$.

The proof is easy as, in each F^o_i and F^u_i, the accepting transitions are over- and under-approximated. With this lemma, we are able to identify some accepting and rejecting bottom SCCs in the product.

We remark that $\mathcal{S}^o$ and $\mathcal{S}^u$ differ only in their accepting conditions. Thus, the corresponding GMCs $\mathcal{M} \otimes \mathcal{S}^u$ and $\mathcal{M} \otimes \mathcal{S}^o$ also differ only for their accepting conditions. If we ignore the accepting conditions, we have the following result:

▶ **Lemma 13.** *Let* $\mathcal{M}$ *be a MC,* $\mathcal{B}$ *a NGBA,* $\mathcal{A} = \det(\mathcal{B})$*, and* $\mathcal{S}^u$ *as defined above; let* $\mathcal{S}$ *be* $\mathcal{S}^u$ *without the accepting conditions. Then,* $\mathcal{M} \times \mathcal{S}$ *and* $[\mathcal{M} \times \mathcal{A}]$ *are isomorphic.*

The proof is rather easy – it is based on the isomorphism identifying a state (m, R) of $\mathcal{M} \times \mathcal{S}$ with the state $(m, [d])$ of $[\mathcal{M} \times \mathcal{A}]$ such that $\mathsf{rchd}(d) = R$.

Considering the accepting conditions, we can classify some bottom SCCs.

▶ **Lemma 14.** *Let* $\mathcal{M}$ *be a MC and* $\mathcal{B}$ *a NGBA. Let* $\mathcal{S}^o$ *and* $\mathcal{S}^u$ *be as defined above. Let* $\mathtt{S}$ *be a bottom SCC of* $\mathcal{M} \otimes \mathcal{S}^u$*. Then,*

- $\mathtt{S}$ *is accepting if* $\mathrm{F}^u_i \cap \pi_{\mathcal{S}^u}(\mathrm{P}_\mathtt{S}) \neq \emptyset$ *holds for all* $i \in [1..k]$*;*
- $\mathtt{S}$ *is rejecting if* $\mathrm{F}^o_i \cap \pi_{\mathcal{S}^u}(\mathrm{P}_\mathtt{S}) = \emptyset$ *holds for some* $i \in [1..k]$.

The above result directly follows by Lemma 12. Figure 4 shows the product of the MC $\mathcal{M}_\mathcal{E}$ depicted in Figure 2 and the subset automaton $\mathcal{S}_\mathcal{E}$ in Figure 3. It is easy to check that the only bottom SCC is neither accepting nor rejecting.

For the bottom SCCs, for which we cannot conclude whether they are accepting or rejecting, we continue with the breakpoint construction.

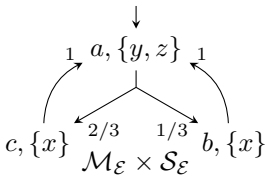


Figure 4 The product of $\mathcal{M}_\mathcal{E}$ and $\mathcal{S}_\mathcal{E}$.

3.3.2 Breakpoint Construction

For a given NGBA $\mathcal{B} = (\Sigma, Q, I, \mathrm{T}, \mathbf{F}_k)$, we denote with $\mathsf{bp}(Q,k) = \{\,(R,j,C) \mid C \subsetneq R \subseteq Q,\ j \in [1..k]\,\}$ the breakpoint set. Intuitively, for a state d of the DRA $\mathcal{A} = \det(\mathcal{B})$, the corresponding breakpoint state is $\langle d \rangle = (R,j,C)$ where R contains the states labelling the root ε of labelled ordered tree associated to d, C subsumes the states labelling the lower levels of the tree, and j is the index of the accepting set F_j considered at the root of the tree.

We build two DRAs $\mathcal{BP}^o = (\Sigma, \mathsf{bp}(Q,k), (I,1,\emptyset), \mathrm{T}', \{(\mathrm{A}_\varepsilon, \emptyset), (\mathrm{T}', \mathrm{R}_0)\})$ and $\mathcal{BP}^u = (\Sigma, \mathsf{bp}(Q,k), (I,1,\emptyset), \mathrm{T}', \{(\mathrm{A}_\varepsilon, \emptyset)\})$, called the *breakpoint automata*, as follows.

From the breakpoint state (R,j,C), let $R' = \mathrm{T}(R,\sigma)$ and $C' = \mathrm{T}(C,\sigma) \cup \mathrm{F}_j(R,\sigma)$. Then an accepting transition with letter σ reaches $(R', j \oplus_k 1, \emptyset)$ if $C' = R'$. Formally,

$$\mathrm{A}_\varepsilon = \{\,((R,j,C),\sigma,(R', j \oplus_k 1, \emptyset)) \mid (R,j,C) \in \mathsf{bp}(Q,k),\ \sigma \in \Sigma,\ \emptyset \neq R' = \mathrm{T}(R,\sigma),\ C' = \mathrm{T}(C,\sigma) \cup \mathrm{F}_j(R,\sigma),\ C' = R'\,\}.$$

The remaining transitions, for which $C' \neq R'$, are obtained in a similar way, but now the transition reaches (R', j, C'), where j remains unchanged; formally,

$$\mathrm{T}'' = \{\,((R,j,C),\sigma,(R', j, C')) \mid (R,j,C) \in \mathsf{bp}(Q,k),\ \sigma \in \Sigma,\ \emptyset \neq R' = \mathrm{T}(R,\sigma),\ C' = \mathrm{T}(C,\sigma) \cup \mathrm{F}_j(R,\sigma),\ C' \neq R'\,\}.$$

The transition relation T' is just $\mathrm{T}'' \cup \mathrm{A}_\varepsilon$. Transitions that satisfy $C' = \emptyset$ are rejecting:

$$\mathrm{R}_0 = \{\,((R,j,C),\sigma,d) \in \mathrm{T}'' \mid \mathrm{T}(C,\sigma) = \emptyset\,\}.$$

Figure 5 shows the reachable fragment of the breakpoint construction for the NGBA $\mathcal{B}_\mathcal{E}$ depicted in Figure 1. The double arrow transitions are in A_ε while the remaining transitions are in R_0.

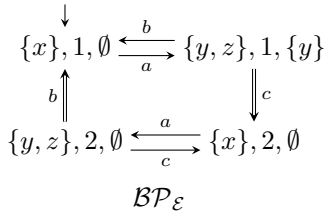


Figure 5 The breakpoint construction for $\mathcal{B}_\mathcal{E}$ (fragment reachable from $(\{x\}, 1, \emptyset)$).

▶ **Theorem 15.** *The following inclusions hold:*

$$\mathcal{L}(\mathcal{S}^u_{[d]}) \subseteq \mathcal{L}(\mathcal{BP}^u_{\langle d \rangle}) \subseteq \mathcal{L}(\mathcal{A}_d) \subseteq \mathcal{L}(\mathcal{BP}^o_{\langle d \rangle}), \mathcal{L}(\mathcal{S}^o_{[d]}).$$

We remark that the breakpoint construction can be refined further such that it is finer than $\mathcal{L}(\mathcal{S}^o_{[d]})$. However we leave it as future work to avoid heavy technical preparations. Exploiting the above theorem, the following becomes clear.

▶ **Corollary 16.** *Let* $\mathtt{S}$ *be a bottom SCC of the quotient MC. Let* $(m,d) \in \mathtt{S}$ *be an arbitrary state of* $\mathtt{S}$*. Moreover, let* $\mathcal{BP}^o$*,* $\mathcal{BP}^u$ *be the breakpoint automata. Then,*

- $\mathtt{S}$ *is accepting if there exists a bottom SCC* $\mathtt{S}'$ *in* $\mathcal{M} \otimes \mathcal{BP}^u_{\langle d \rangle}$ *with* $\mathtt{S} = [\mathtt{S}']$*, which is accepting (i.e.,* $\mathtt{S}'$ *contains some transition in* A_ε*).*
- $\mathtt{S}$ *is rejecting if there exists a bottom SCC* $\mathtt{S}'$ *in* $\mathcal{M} \otimes \mathcal{BP}^o_{\langle d \rangle}$ *with* $\mathtt{S} = [\mathtt{S}']$*, which is rejecting (i.e.,* $\mathtt{S}'$ *contains no transition in* A_ε*, but some transition in* R_0*).*

Note that the products $\mathcal{M} \otimes \mathcal{BP}^u_{\langle d \rangle}$ and $\mathcal{M} \otimes \mathcal{BP}^o_{\langle d \rangle}$ are the same RMCs except for their accepting conditions. Figure 6 shows the product of the MC $\mathcal{M}_\mathcal{E}$ depicted in Figure 2 and the breakpoint automaton $\mathcal{BP}^u_\mathcal{E}$ in Figure 5. It is easy to see that the only bottom SCC is accepting.

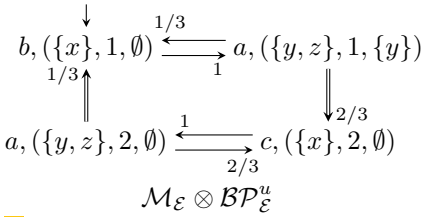


Figure 6 The product of $\mathcal{M}_\mathcal{E}$ and $\mathcal{BP}^u_\mathcal{E}$.

Together with Corollary 11, Lemma 14 and Corollary 16 immediately provide the following result, which justifies the incremental evaluations of the bottom SCCs.

▶ **Corollary 17.** *Given a MC $\mathcal{M}$, a NGBA $\mathcal{B}$, and $\mathcal{A} = \det(\mathcal{B})$, if $[(m, d)]$ is a state in a bottom SCC of the quotient MC and $[d] = [d']$, then*

- $\mathfrak{P}^{\mathcal{M}\otimes\mathcal{A}_d}_{(m,d)}(\mathcal{B}) = 1$ *if* $\mathfrak{P}^{\mathcal{M}\otimes\mathcal{S}^u_{[d']}}_{(m,[d])}(\mathcal{B}) > 0$ *or* $\mathfrak{P}^{\mathcal{M}\otimes\mathcal{BP}^u_{\langle d' \rangle}}_{(m,\langle d \rangle)}(\mathcal{B}) > 0$, *and*
- $\mathfrak{P}^{\mathcal{M}\otimes\mathcal{A}_d}_{(m,d)}(\mathcal{B}) = 0$ *if* $\mathfrak{P}^{\mathcal{M}\otimes\mathcal{S}^o_{[d']}}_{(m,[d])}(\mathcal{B}) < 1$ *or* $\mathfrak{P}^{\mathcal{M}\otimes\mathcal{BP}^o_{\langle d' \rangle}}_{(m,\langle d \rangle)}(\mathcal{B}) < 1$.

In case there are remaining bottom SCCs, for which we cannot conclude whether they are accepting or rejecting, we continue with a multi-breakpoint construction that is language-equivalent to the Rabin construction.

3.3.3 Multi-Breakpoint Construction

The multi-breakpoint construction we propose to decide the remaining bottom SCCs makes use of a combination of the subset and breakpoint constructions we have seen in the previous steps, but with different accepting conditions: for the subset automaton $\mathcal{S} = \mathcal{S}(\mathcal{B}) = (\Sigma, Q_{ss}, q_{ss}, \mathrm{T}_{ss}, \mathrm{F}_{ss})$, we use the accepting condition $\mathrm{F}_{ss} = \emptyset$, i.e., the automaton accepts no words; for the breakpoint automaton $\mathcal{BP} = \mathcal{BP}(\mathcal{B}) = (\Sigma, Q_{bp}, q_{bp}, \mathrm{T}_{bp}, \mathrm{F}_{bp})$, we consider $\mathrm{F}_{bp} = \mathrm{A}_\varepsilon$. Note that the Büchi acceptance condition $\mathrm{F}_{bp} = \mathrm{A}_\varepsilon$ is trivially equivalent to the Rabin acceptance condition $\{(\mathrm{A}_\varepsilon, \emptyset)\}$, so $\mathcal{BP}$ is essentially $\mathcal{BP}^u$. We remark that in general the languages accepted by $\mathcal{S}$ and $\mathcal{BP}$ are different from $\mathcal{L}(\mathcal{B})$: $\mathcal{L}(\mathcal{S}) = \emptyset$ by construction while $\mathcal{L}(\mathcal{BP}) \subseteq \mathcal{L}(\mathcal{B})$, as shown in Theorem 15. To generate an automaton accepting the same language of $\mathcal{B}$, we construct a *semi-deterministic* automaton $\mathcal{SD} = \mathcal{SD}(\mathcal{B}) = (\Sigma, Q_{sd}, q_{sd}, \mathrm{T}_{sd}, \mathrm{F}_{sd})$ by merging $\mathcal{S}$ and $\mathcal{BP}$ as follows: $Q_{sd} = Q_{ss} \cup Q_{bp}$, $q_{sd} = q_{ss}$, $\mathrm{T}_{sd} = \mathrm{T}_{ss} \cup \mathrm{T}_t \cup \mathrm{T}_{bp}$, and $\mathrm{F}_{sd} = \mathrm{F}_{bp}$, where $\mathrm{T}_t = \{\, (R, \sigma, (R', j', C')) \mid R \in Q_{ss}, (R', j', C') \in Q_{bp}, \text{ and } R' \subseteq \mathrm{T}_{ss}(R, \sigma) \,\}$. $\mathcal{B}$ and $\mathcal{SD}$ accept the same language:

▶ **Proposition 18.** *Given a NGBA $\mathcal{B}$, let $\mathcal{SD}$ be constructed as above. Then, $\mathcal{L}(\mathcal{SD}) = \mathcal{L}(\mathcal{B})$.*

For $\mathcal{A} = \det(\mathcal{B})$, it is known by Lemma 13 that $\mathcal{M} \times \mathcal{S}$ and $\mathcal{M} \times \mathcal{A}$ are strictly related, so we can define the accepting SCC of $\mathcal{M} \times \mathcal{S}$ by means of the accepting states of $\mathcal{M} \times \mathcal{A}$.

▶ **Definition 19.** Given a MC $\mathcal{M}$ and a NGBA $\mathcal{B}$, for $\mathcal{S} = \mathcal{S}(\mathcal{B})$ and $\mathcal{A} = \det(\mathcal{B})$, we say that a bottom SCC $\mathtt{S}$ of $\mathcal{M} \times \mathcal{S}$ is accepting if, and only if, there exists a state $s = (m, d)$ in an accepting bottom SCC $\mathtt{S}'$ of $\mathcal{M} \times \mathcal{A}$ such that $(m, \mathsf{rchd}(d)) \in \mathtt{S}$.

Note that Proposition 8 ensures that the accepting SCCs of $\mathcal{M} \times \mathcal{S}$ are well defined.

▶ **Theorem 20.** *Given a MC $\mathcal{M}$ and a NGBA $\mathcal{B}$, for $\mathcal{SD} = \mathcal{SD}(\mathcal{B})$ and $\mathcal{S} = \mathcal{S}(\mathcal{B})$, the following facts are equivalent:*

1. *$\mathtt{S}$ is an accepting bottom SCC of $\mathcal{M} \times \mathcal{S}$;*
2. *there exist $(m, R) \in \mathtt{S}$ and $R' \subseteq R$ such that $(m, (R', j, \emptyset))$ belongs to an accepting SCC of $\mathcal{M} \otimes \mathcal{SD}_{(m,(R',j,\emptyset))}$ for some $j \in [1..k]$;*
3. *there exist $(m, R) \in \mathtt{S}$ and $q \in R$ such that $(m, (\{q\}, 1, \emptyset))$ reaches with probability 1 an accepting SCC of $\mathcal{M} \otimes \mathcal{SD}_{(m,(\{q\},1,\emptyset))}$.*

Theorem 20 provides a practical way to check whether an SCC $\mathtt{S}$ of $\mathcal{M} \times \mathcal{S}$ is accepting: it is enough to check whether some state (m, R) of $\mathtt{S}$ has $R \supseteq R'$ for some $(m, (R', j, \emptyset))$ in the accepting region of $\mathcal{M} \otimes \mathcal{SD}$, or whether, for a state $q \in R$, $(m, (\{q\}, 1, \emptyset))$ reaches with probability 1 the accepting region. We remark that, by construction of $\mathcal{SD}$, if we change the initial state of $\mathcal{SD}$ to (R, j, C) – i.e., if we consider $\mathcal{SD}_{(R,j,C)}$ – then the run can only visit breakpoint states; i.e., it is actually a run of $\mathcal{BP}_{(R,j,C)}$.

4 Markov Decision Processes

The lazy determinisation approach proposed in this paper extends to Markov decision processes (MDPs) after minor adaptation; Markov chains have mainly been used for ease of notation. We give here an outline of the adaptation with a focus on the differences and particularities that need to be taken into consideration when we are dealing with MDPs.

An MDP is a tuple $\mathcal{M} = (M, L, \mathit{Act}, \mu_0, \mathrm{P})$ where M, L, and μ_0 are as for Markov chains, Act is a finite set of actions, and $\mathrm{P} \colon M \times \mathit{Act} \to \mathit{Dist}(M)$ is the transition probability function where $\mathit{Dist}(M)$ is the set of distributions over M. The nondeterministic choices are resolved by a scheduler υ that chooses the next action to be executed depending on a finite path. Like for Markov chains, the principal technique to analyse MDPs against a specification φ is to construct a deterministic Rabin automaton $\mathcal{A}$, build the product $\mathcal{M} \otimes \mathcal{A}$, and analyse it. This product will be referred to as a *Rabin MDP* (RMDP). According to [3], for a RMDP, it suffices to consider memoryless deterministic schedulers of the form $\upsilon \colon M \times Q \to \mathit{Act}$, where Q is the set of states of $\mathcal{A}$. Given a NGBA specification $\mathcal{B}_\varphi$, we are interested in $\sup_\upsilon \mathfrak{P}^{\mathcal{M},\upsilon}(\mathcal{B}_\varphi)$. In particular, one can use finite memory schedulers on $\mathcal{M}$. (Schedulers that control $\mathcal{M}$ can be used to control $\mathcal{M} \otimes \mathcal{A}$ for all deterministic automata $\mathcal{A}$.) The superscript $\mathcal{M}$ is omitted when it is clear from the context. We remark that the infimum can be treated accordingly, as $\inf_\upsilon \mathfrak{P}^\upsilon(\mathcal{B}_\varphi) = 1 - \sup_\upsilon \mathfrak{P}^\upsilon(\mathcal{B}_{\neg\varphi})$.

As Proposition 8 operates on words, it immediately extends to MDPs. Under the corresponding equivalence relation we obtain a *quotient MDP*. From here, it is clear that we can use the estimation of the word languages provided in Theorem 15 to estimate $\sup_\upsilon \mathfrak{P}^\upsilon(\mathcal{B}_\varphi)$.

▶ **Corollary 21.** *Given an MDP $\mathcal{M}$ and a NGBA $\mathcal{B}$, let m be a state of $\mathcal{M}$ and d, d' be states of $\mathcal{A} = \det(\mathcal{B})$ with $[d] = [d']$. Then $\sup_\upsilon \mathfrak{P}^\upsilon_{(m,[d])}(\mathcal{S}^u_{[d]}) \leq \sup_\upsilon \mathfrak{P}^\upsilon_{(m,\langle d\rangle)}(\mathcal{BP}^u_{\langle d\rangle}) \leq \sup_\upsilon \mathfrak{P}^\upsilon_{(m,d)}(\mathcal{A}_d) = \sup_\upsilon \mathfrak{P}^\upsilon_{(m,d')}(\mathcal{A}_{d'}) \leq \sup_\upsilon \mathfrak{P}^\upsilon_{(m,\langle d\rangle)}(\mathcal{BP}^o_{\langle d\rangle})$, $\sup_\upsilon \mathfrak{P}^\upsilon_{(m,[d])}(\mathcal{S}^o_{[d]})$ holds.*

In the standard evaluation of RMDP, the *end components* of the product $\mathcal{M} \otimes \mathcal{A}$ play a role comparable to the one played by bottom SCCs in MCs. An end component (EC) is simply a sub-MDP, which is closed in the sense that there exists a memoryless scheduler υ such that the induced Markov chain is a bottom SCC. If there is a scheduler that additionally guarantees that a run that contains all possible transitions infinitely often is accepting, then the EC is *accepting*. Thus, one can stay in the EC and traverse all of its transitions (that the scheduler allows) infinitely often, where acceptance is defined as for BSCCs in MCs.

▶ **Theorem 22.** *Given an MDP $\mathcal{M}$ and a NGBA $\mathcal{B}$, for $\mathcal{A} = \det(\mathcal{B})$, $\mathcal{SD} = \mathcal{SD}(\mathcal{B})$, and $\mathcal{S} = \mathcal{S}(\mathcal{B})$, if $\mathtt{C}$ is an accepting EC of $\mathcal{M}\otimes\mathcal{A}$, then* (1.) *$[\mathtt{C}]$ is an EC of $\mathcal{M}\times\mathcal{S}$ and* (2.) *$\mathtt{C}' = \langle \mathtt{C}\rangle$ is an accepting EC of $\mathcal{M}\otimes\mathcal{SD}$. $\mathtt{C}'$ contains a state $(m, (R, 1, \emptyset))$ with $R \subseteq [d]$ and $(m, d) \in \mathtt{C}$.*

Note that, since each EC $\mathtt{C}$ of $\mathcal{M} \otimes \mathcal{A}$ is either accepting or rejecting, finding an accepting EC $\mathtt{C}' = \langle \mathtt{C}\rangle$ of $\mathcal{M} \otimes \mathcal{SD}$ allows us to derive that $\mathtt{C}$ is accepting as well.

For RMDPs, it suffices to analyse maximal end components (MEC). We define a MEC as accepting if it contains an accepting EC. MECs are easy to construct and, for each

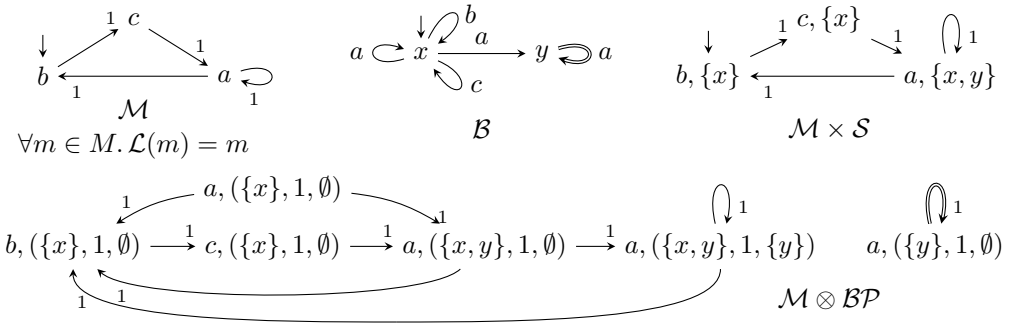


Figure 7 Finding accepting ECs in MDPs: MDP $\mathcal{M}$, NGBA $\mathcal{B}$, $\mathcal{S} = \mathcal{S}(\mathcal{B})$, $\mathcal{BP} = \mathcal{BP}(\mathcal{B})$.

accepting pair, they are easy to evaluate: it suffices to remove the rejecting transitions, repeat the construction of MECs on the remainder, and check if there is any that contains an accepting transition. Once accepting MECs are determined, their states are assigned a winning probability of 1, and evaluating the complete MDP reduces to a maximal reachability analysis, which reduces to solving an LP problem. It can therefore be solved in polynomial time.

These two theorems allow us to use a layered approach of lazy determinisation for MDPs, which is rather similar to the one described for Markov chains. We start with the quotient MDP, and consider an arbitrary MEC $\mathtt{C}$. By using the accepting conditions of the subset automata $\mathcal{S}^u$ and $\mathcal{S}^o$, we check whether $\mathtt{C}$ is accepting or rejecting, respectively. If this test is inconclusive, we first refine $\mathtt{C}$ by a breakpoint construction, and finally by a multi-breakpoint construction. We remark that, as for Markov chains, the breakpoint and multi-breakpoint constructions can be considered as oracles: when we have identified the accepting MECs, a plain reachability analysis is performed on the quotient MDP.

Theorem 22 makes clear what needs to be calculated in order to classify an EC – and thus a MEC – as accepting, while Corollary 21 allows for applying this observation in the quantitative analysis of an MDP, and also to smoothly combine this style of reasoning with the lazy approach. This completes the picture of [6] for the quantitative analysis of MDPs, which is technically the same as their analysis of concurrent probabilistic programs [6]. It is worthwhile to point out that, in principle, the qualitative analysis from [6] could replace Theorem 22 when starting with a Büchi automaton that recognises the complement of the models of φ and minimising instead of maximising. This detour would, however, not allow us to restrict the analysis to (M)ECs, which would, in turn, lead to a significant overhead.

For MDPs, differently from the subset and breakpoint construction, for the multi-breakpoint case testing only one $(m, R) \in \mathtt{C}$ in general is not sufficient; consider the MDP $\mathcal{M}$ and the NGBA $\mathcal{B}$ depicted in Figure 7. We first consider the product MDP $\mathcal{M} \times \mathcal{S}$, containing one MEC. We first try to decide whether it is accepting by considering the state $(c, \{x\})$. The only nonempty subset of $\{x\}$ is the set itself, thus we look for accepting MECs in $\mathcal{M} \otimes \mathcal{BP}_{(c,(\{x\},1,\emptyset))}$. It is clear that from $(c, (\{x\}, 1, \emptyset))$ no accepting MECs can be reached. In contrast to the MC setting, we cannot conclude that the original MEC is not accepting. Instead, we remove $(c, \{x\})$ from the set of states to consider, as well as $(b, \{x\})$, from which we cannot avoid reaching $(c, \{x\})$. The state left to try is $(a, \{x, y\})$, where we have two transitions available. Indeed, in $\mathcal{M} \otimes \mathcal{BP}$ the singleton MEC $\{(a, (\{y\}, 1, \emptyset))\}$ is accepting. Thus the MEC of $\mathcal{M} \times \mathcal{S}$ is accepting, though only one of its states – $\{(a, \{x, y\})\}$ – allows us to conclude this, and we need to select the correct subset, $\{y\}$, to start with.

Table 1 Runtime comparison for the randomised mutual exclusion protocol.

property		n	time BP expl.	BP BDD	RB expl.	RB BDD	PRISM	Rabinizer3	scaled [4]
$\mathbb{P}_{\min=?}(\mathbf{GF}p_1{=}10 \wedge \mathbf{GF}p_2{=}10 \wedge \mathbf{GF}\,p_3{=}10 \wedge \mathbf{GF}p_4{=}10)$	(3)	4	**3**	5	15	28	–	104	23
		5	**19**	21	–	104	–	1478	380
$\mathbb{P}_{\max=?}((\mathbf{GF}p_1{=}0 \vee \mathbf{FG}p_2{\neq}0) \wedge(\mathbf{GF}p_2{=}0 \vee \mathbf{FG}p_3{\neq}0))$	(4)	3	**1**	2	2	4	138	2	**1**
		4	**3**	7	4	15	–	20	18
		5	**19**	32	35	76	–	319	299
$\mathbb{P}_{\max=?}((\mathbf{GF}p_1{=}0 \vee \mathbf{FG}p_1{\neq}0) \wedge(\mathbf{GF}p_2{=}0 \vee \mathbf{FG}p_2{\neq}0))$	(5)	3	2	2	2	4	41	2	**1**
		4	**3**	8	4	17	336	19	18
		5	**19**	34	45	68	–	314	289
$\mathbb{P}_{\max=?}((\mathbf{GF}p_1{=}0 \vee \mathbf{FG}p_2{\neq}0) \wedge(\mathbf{GF}p_2{=}0 \vee \mathbf{FG}p_3{\neq}0) \wedge(\mathbf{GF}p_3{=}0 \vee \mathbf{FG}p_1{\neq}0))$	(6)	3	**1**	2	2	6	–	5	4
		4	**3**	9	7	27	–	52	47
		5	**29**	38	99	124	–	871	762
$\mathbb{P}_{\max=?}((\mathbf{GF}p_1{=}0 \vee \mathbf{FG}p_1{\neq}0) \wedge(\mathbf{GF}p_2{=}0 \vee \mathbf{FG}p_2{\neq}0) \wedge(\mathbf{GF}p_3{=}0 \vee \mathbf{FG}p_3{\neq}0))$	(7)	3	**1**	2	2	9	–	5	5
		4	**3**	9	12	41	–	50	49
		5	**29**	38	–	171	–	849	792
$\mathbb{P}_{\min=?}((\mathbf{GF}p_1{\neq}10 \vee \mathbf{GF}p_1{=}0 \vee \mathbf{FG}p_1{=}1) \wedge \mathbf{GF}\,p_1{\neq}0 \wedge \mathbf{GF}p_1{=}1)$	(8)	3	**1**	2	**1**	3	**1**	**1**	**1**
		4	**3**	6	**3**	10	8	13	6
		5	**17**	25	**17**	41	123	208	91
$\mathbb{P}_{\max=?}((\mathbf{G}p_1{\neq}10 \vee \mathbf{G}p_2{\neq}10 \vee \mathbf{G}p_3{\neq}10) \wedge(\mathbf{FG}p_1{\neq}1 \vee \mathbf{GF}p_2{=}1 \vee \mathbf{GF}p_3{=}1) \wedge(\mathbf{FG}p_2{\neq}1 \vee \mathbf{GF}p_1{=}1 \vee \mathbf{GF}p_3{=}1))$	(9)	3	**2**	6	**2**	4	–	982	50
		4	9	16	**7**	14	–	1718	440
		5	136	60	91	**56**	–	–	–
$\mathbb{P}_{\min=?}((\mathbf{FG}p_1{\neq}0 \vee \mathbf{FG}p_2{\neq}0 \vee \mathbf{GF}p_3{=}0) \vee(\mathbf{FG}p_1{\neq}10 \wedge \mathbf{GF}p_2{=}10 \wedge \mathbf{GF}p_3{=}10)$	(10)	3	**2**	3	**2**	5	169	3	**2**
		4	79	12	**4**	18	–	32	21
		5	–	48	**44**	69	–	480	339

5 Implementation and Results

We have implemented our approach in our IscasMC tool [10] in both explicit and BDD-based symbolic versions. We use LTL formulas to specify properties, and apply SPOT [7] to translate them to NGBAs. Our experimental results suggest that our technique provides a practical approach for checking LTL properties for probabilistic systems. A web interface to IscasMC can be found at `http://iscasmc.ios.ac.cn/`. For our experiments, we used a 3.6 GHz Intel Core i7-4790 with 16GB 1600 MHz DDR3 RAM.

We consider a set of properties analysed previously in [4]. As there, we aborted tool runs when they took more than 30 minutes or needed more than 4GB of RAM. The comparison with the results from [4] cannot be completely accurate: unfortunately, their implementation is not available on request to the authors, and for their results they did not state the exact speed of the machine used. By comparing the runtimes stated for PRISM in [4] with the corresponding runtimes we obtained on our machine, we estimate that our machine is faster than theirs by about a factor of 1.6. Thus, we have included the values from [4] divided by 1.6 to take into account the estimated effect of the machine. In Table 1 we provide the results obtained. Here, "property" and "n" are as in [4] and depict the property and the size of the model under consideration. We report the total runtime in seconds ("time") for the explicit-state ("BP expl.") and the BDD-based symbolic ("BP BDD") implementations of the multi-breakpoint construction, as well as the explicit and symbolic ("RB expl.", "RB BDD") of the Rabin-based implementation. In both BP and RB cases, we first apply the subset and breakpoint steps. We also include the runtimes of PRISM ("PRISM") and of the tool used in [4] ("scaled [4]") developed for a subclass of LTL formulas and its generalisation to full LTL [8] implemented in Rabinizer 3 [13] ("Rabinizer 3"). We mark the best (rounded) runtimes with bold font.

The runtime of our approaches is almost always better than the runtime of the other methods. In many cases, the multi-breakpoint approach performs better than the Rabin-

Table 2 Runtime comparison for the workstation cluster protocol.

	time					
property	BP expl.	BP BDD	RB expl.	RB BDD	PRISM	Rabinizer3
$prop\mathbf{U}_{10}$	**2**	3	**2**	3	23	121
$prop\mathbf{U}_{11}$	**3**	4	**3**	4	95	686
$prop\mathbf{U}_{12}$	**4**	5	**4**	5	–	–
$prop\mathbf{U}_{13}$	**7**	8	**7**	8	–	–
$prop\mathbf{GF}\wedge_3$	**1**	**1**	**1**	**1**	48	**1**
$prop\mathbf{GF}\wedge_4$	2	**1**	**1**	**1**	–	2
$prop\mathbf{GF}\wedge_5$	**1**	**1**	**1**	2	–	14
$prop\mathbf{GF}\wedge_6$	**1**	**1**	**1**	**1**	–	177
$prop\mathbf{GF}\vee_3$	**1**	**1**	2	3	233	**1**
$prop\mathbf{GF}\vee_4$	**1**	**1**	2	3	–	2
$prop\mathbf{GF}\vee_5$	**1**	2	**1**	2	–	14
$prop\mathbf{GF}\vee_6$	**2**	**2**	**2**	4	–	180

based approach (restricted to the single undecided end component), but not always. Broadly speaking, this can happen when the breakpoint construction has to consider many subsets as starting points for one end component, while the Rabin determinisation does not lead to a significant overhead compared to the breakpoint construction. Thus, both methods are of value. Both of them are faster than the specialised algorithm of [4] and Rabinizer 3. We assume that one reason for this is that this method is not based on the evaluation of end components in the subset product, and also its implementation might not involve some of the optimisations we apply. In most cases, the explicit-state implementation is faster than the BDD-based approach, which is, however, more memory-efficient.

As another case study, we consider a model [12] of two clusters of n=16 workstations each, so that the two clusters are connected by a backbone. Each of the workstations may fail with a given rate, as may the backbone. Though this case study is a continuous-time Markov chain, we focused on time-unbounded properties, such that we could use discrete-time Markov chains to analyse them. We give the results in Table 2, where the meaning of the columns is as for the mutual exclusion case in Table 1. The properties $prop\mathbf{U}_k = \mathbb{P}_{=?}(\mathit{left\_n{=}n}\,\mathbf{U}\,(\mathit{left\_n{=}n{-}1}\,\mathbf{U}\,(\ldots\,\mathbf{U}\,(\mathit{left\_n{=}n{-}k}\,\mathbf{U}\,\mathit{right\_n{\neq}n})\ldots)$ are probabilities of the event of component failures with respect to the order (first k failures on left before right) while the properties $prop\mathbf{GF}\wedge_k = \mathbb{P}_{=?}(\mathbf{GF}\mathit{left\_n{=}n} \wedge \bigvee_{i=0}^{k} \mathbf{FG}\mathit{right\_n{=}n{-}i})$ and $prop\mathbf{GF}\vee_k = \mathbb{P}_{=?}(\mathbf{GF}\mathit{left\_n{=}n} \vee \bigvee_{i=0}^{k} \mathbf{FG}\mathit{right\_n{=}n{-}i})$ describe the long-run number of workstations functional. As clearly shown from the results in the table, IscasMC outperforms PRISM and Rabinizer 3 all cases, in particular for large PLTL formulas. It is worthwhile to analyse in details the three properties and how they have been checked: for the $prop\mathbf{U}_k$ case, the subset construction suffices and returns a (rounded) probability value of 0.509642; for $prop\mathbf{GF}\wedge_k$, the breakpoint construction is enough to determine that the property holds with probability 0. This explains why the BP and RB columns are essentially the same (we remark that the reported times are the rounded actual runtimes). Property $prop\mathbf{GF}\vee_k$, instead, requires to use the multi-breakpoint or the Safra-based construction to complete the model checking analysis and obtain a probability value of 1.

Acknowledgement. This work is supported by the Natural Science Foundation of China (NSFC) under grant No. 61472406, 61472473, 61361136002; the Chinese Academy of Sciences Fellowship for International Young Scientists under grant No. 2013Y1GB0006, 2015VTC029; the Research Fund for International Young Scientists (Grant No. 61450110461); the

CAS/SAFEA International Partnership Program for Creative Research Teams; the EPSRC through grant EP/M027287/1. We thank Jan Kretínský for providing us the source code of Rabinizer 3.

References

1. Tomás Babiak, Mojmír Kretínský, Vojtech Rehák, and Jan Strejcek. LTL to Büchi automata translation: Fast and more deterministic. In *TACAS*, volume 7214 of *LNCS*, pages 95–109, 2012.
2. Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.
3. Andrea Bianco and Luca de Alfaro. Model checking of probabalistic and nondeterministic systems. In *FSTTCS*, volume 1026 of *LNCS*, pages 499–513, 1995.
4. Krishnendu Chatterjee, Andreas Gaiser, and Jan Kretínský. Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In *CAV*, volume 8044 of *LNCS*, pages 559–575, 2013.
5. Frank Ciesinski and Christel Baier. LiQuor: A tool for qualitative and quantitative linear time analysis of reactive systems. In *QEST*, pages 131–132, 2006.
6. Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *JACM*, 42(4):857–907, 1995.
7. Alexandre Duret-Lutz. LTL translation improvements in SPOT. In *VECoS*, pages 72–83, 2011.
8. Javier Esparza and Jan Kretínský. From LTL to deterministic automata: A Safraless compositional approach. In *CAV*, volume 8559 of *LNCS*, pages 192–208, 2014.
9. Ernst Moritz Hahn, Guangyuan Li, Sven Schewe, Andrea Turrini, and Lijun Zhang. Lazy determinisation for quantitative model checking. CoRR, available at `http://arxiv.org/abs/1311.2928`, 2014.
10. Ernst Moritz Hahn, Yi Li, Sven Schewe, Andrea Turrini, and Lijun Zhang. IscasMC: A web-based probabilistic model checker. In *FM*, volume 8442 of *LNCS*, pages 312–317, 2014.
11. Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *FAC*, 6(5):512–535, 1994.
12. Boudewijn R. Haverkort, Holger Hermanns, and Joost-Pieter Katoen. On the use of model checking techniques for dependability evaluation. In *SRDS*, pages 228–237, 2000.
13. Zuzana Komárková and Jan Kretínský. Rabinizer 3: Safraless translation of LTL to small deterministic automata. In *ATVA*, volume 8837 of *LNCS*, pages 235–241, 2014.
14. Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. Safraless compositional synthesis. In *CAV*, volume 4144 of *LNCS*, pages 31–44, 2006.
15. Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591, 2011.
16. Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *JLMCS*, 3(3:5), 2007.
17. Shmuel Safra. On the complexity of ω-automata. In *FOCS*, pages 319–327, 1988.
18. Sven Schewe. Tighter bounds for the determinisation of Büchi automata. In *FoSSaCS*, volume 5504 of *LNCS*, pages 167–181, 2009.
19. Sven Schewe and Thomas Varghese. Tight bounds for the determinisation and complementation of generalised Büchi automata. In *ATVA*, volume 7561 of *LNCS*, pages 42–56, 2012.
20. Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344, 1986.

A Modular Approach for Büchi Determinization*

Dana Fisman[1] and Yoad Lustig[2]

1 **University of Pennsylvania, US**
`fisman@seas.upenn.edu`
2 **Google Inc., Israel**
`yoad.lustig@gmail.com`

Abstract

The problem of Büchi determinization is a fundamental problem with important applications in reactive synthesis, multi-agent systems and probabilistic verification. The first asymptotically optimal Büchi determinization (a.k.a. the Safra construction), was published in 1988. While asymptotically optimal, the Safra construction is notorious for its technical complexity and opaqueness in terms of intuition. While some improvements were published since the Safra construction, notably Kähler and Wilke's construction, understanding the constructions remains a non-trivial task.

In this paper we present a modular approach to Büchi determinization, where the difficulties are addressed one at a time, rather than simultaneously, making the solutions natural and easy to understand. We build on the notion of the skeleton trees of Kähler and Wilke. We first show how to construct a deterministic automaton in the case the skeleton's width is one. Then we show how to construct a deterministic automaton in the case the skeleton's width is k (for any given k). The overall construction is obtained by running in parallel the automata for all widths.

1998 ACM Subject Classification F.1.1 Models of Computation, D.2.4 Formal Methods, G.2.2 Graph Algorithms

Keywords and phrases Büchi automata, Determinization, Verification, Games, Synthesis

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.368

1 Introduction

The relationship between deterministic and non-deterministic models of computation is a fundamental question in almost every model of computation. Büchi automata are not an exception. Büchi automata were first introduced, as non-deterministic, in [2], in order to prove the decidability of S1S (second order logic of one successor). The need to consider deterministic automata, however, rose almost immediately. A natural extension of S1S decidability was to prove the decidability of S2S (second order logic of two successors) [27]. (Deciding S2S also allowed solving the fundamental Church problem [3] which is an early formulation of what we now call *synthesis*). To decide S2S, however, one needed automata to run on trees, and while this is natural for deterministic automata, it does not work as expected for non-deterministic automata (see [10] for a comprehensive discussion). Some examples of applications requiring Büchi determization constructions can be found in reactive synthesis [26], multi-agent systems [1] and probabilistic verification [34, 4]. In order to work with deterministic automata, one has to tackle another difficulty: deterministic Büchi automata are less expressive than non-deterministic ones [18]. As a result, the determinization

* The work of the first author was supported by US NSF grant CCF-1138996.

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 368–382

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

of Büchi automata is the following problem: Given a non-deterministic Büchi automaton, find an equivalent deterministic automaton whose acceptance condition may be other than Büchi. (Modern determinization procedures usually use the Parity condition [25, 11].)

Another problem, which is closely related to Büchi determinization is Büchi complementation. Deterministic automata on finite words can be complemented by dualizing the accepting set. Some of the automata on infinite words, namely Muller and Parity, can also be complemented by dualizing the acceptance condition.[1] For this reason determinization constructions yield almost immediately a complementation procedure [28]. In [15] Kupferman and Vardi proposed a complementation construction that actively avoids determinization due to its complexity. Later, in [11], Kähler and Wilke introduced the reduced and skeleton run trees, and both a complementation and a determinization constructions based on them.

The first asymptotically optimal Büchi determinization construction [28] (a.k.a the Safra construction), was published in 1988. While asymptotically optimal, the Safra construction is notorious for its technical complexity and opaqueness in terms of intuition. It took no less than 18 years(!) before Piterman improved upon the Safra construction by modifying it to produce a Parity automaton [25]. A large step forward, in terms of lifting the veil of technical complexity, was made by Kähler and Wilke in [11] who modeled automaton runs by the clear and elegant reduced and skeleton trees. Their complementation construction is as elegant and simple as one can hope for. Their determinization construction is a large step forward, yet it is still non-trivial to understand or implement.

Building on the reduced and skeleton trees of Kähler and Wilke [11], we present in this work a novel Büchi determinization construction which is considerably simpler than previous constructions. The real strength of the construction lies in its modularity. For the first time, the determinization problem is broken down into simpler problems, where the solution of each simple problem is based on one clear idea. Thus the overall solution can be grasped in a gradual manner following several easy steps. This is in contrast to previous solutions in which the correctness reasoning is deferred to the very end, where one needs to reason on a very complex construction.

Overview. In Section 2, we discuss the notions of the *reduced* and *skeleton trees* of [11]. Both trees concisely summarize the runs of a given Büchi automaton $\mathcal{A}$ on given word w, in that these trees have an accepting path iff $\mathcal{A}$ accepts w (where an accepting path is a path with infinitely many accepting nodes). Some of the paths of the *reduced tree* are infinite and some are finite. The *skeleton tree* is the tree obtained from the reduced tree by eliminating the finite paths (i.e. the nodes with finitely many descendants and the edges leading to them). Our constructions conceptually build on the skeleton tree but practically work on the reduced tree. (This is since the skeleton tree, while being easier to reason about, depends on the infinite suffix of the word, and cannot be computed by a deterministic automaton.)

The width of the skeleton tree, formally defined in Section 2, is a central notion in our construction. Roughly speaking, the width of a given level of a tree, is the number of nodes in that level. The width of levels of the skeleton tree are monotonically non-decreasing and bounded by n, the number of states in the given Büchi automaton $\mathcal{A}$. We refer to the maximal width of the skeleton-tree levels by *skel-width*. We use *slice* to refer to the sequence of nodes on a level.

[1] Some, namely Büchi, cannot. Dualization of a Rabin automaton yields a Streett automaton and vice versa.

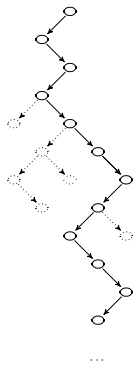


Figure 1 A reduced tree with skel-width one.

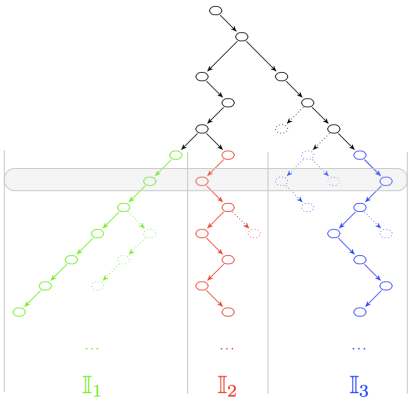


Figure 2 A reduced tree of skel-width 3, marked with the partitions created by the *decide-width* construction. The partitions manage to trap each skeleton path in its own interval in the following sense: Starting from level 5, where the skel-width stabilizes, each slice (see e.g. the marked slice) has three intervals where the i-th interval contains only nodes of the i-th skeleton path, or the finite paths attached to it.

In Section 3, we discuss only words on which the *skel-width* is one, as depicted in Figure 1.[2] For such words, we note that the word is accepted by $\mathcal{A}$ iff it is accepted by the construction of Miyano-Hayashi [22] (henceforth MH) applied to the reduced tree. This is since the MH construction answers if *all* infinite runs of an automaton are accepting, and in the case there exists just *one* infinite run, asking whether *all* infinite runs is the same as asking if there *exists* an infinite run. For the same reason, if we apply it on the reduced tree when the width is greater than one, we may reject if one path accepts but another does not. So in a sense, our mission is to separate the paths of the skeleton tree.

In Section 4 we answer the decision problem: Is the skel-width of the reduced tree smaller than a given k? The solution to this problem creates a partition of a slice of the reduced tree into *intervals*, i.e. consecutive sequences of nodes of the slice, such that each skeleton path (along with the finite paths attached to it) resides in its own interval, as depicted in Figure 2. We capitalize on this separation in consequent constructions.

In Section 5, we show that given a possible width k, we can construct a deterministic automaton that accepts a word of width exactly k iff $\mathcal{A}$ accepts it. This construction essentially runs both previous constructions together, by applying the MH construction on the intervals of the *decide width* construction.

In Section 6, we show how to run in parallel the constructions for all k (upto the number of $\mathcal{A}$'s states), and deduce the correct answer for words of any width. Finally, we show that the complexity of this construction is bounded by $n^{O(n)}$ states, essentially matching the known lower bound of $n!$ given by Michel [21]. In Appendix A, we give a one page illustrative description of the constructions using tokens, bells and buzzers, as a recap. All missing proofs are available in the full version of the paper. We conclude in Section 7.

[2] In this figure (and the rest of the figures in the paper) the reduced tree consists of all nodes and edges, and the skeleton tree of only the solid nodes and edges.

Related Work. The first construction for Büchi determinization is due to McNaughton [20] and dates to 1966. The complexity of this construction was $2^{2^{O(n)}}$, and it took a series of improvements [31, 24] until in 1988, Safra came up with the first asymptotically optimal construction [28] of $n^{O(n)}$. There was a series of works closely studying the exact bounds of Safra, and suggesting improvements obtaining tighter bounds [29, 15, 32, 25, 8, 36, 19].

The works most related to the one presented here were concerned, as we are, in obtaining a determinzation construction simpler than the Safra construction (and the subsequent work), while remaining in $n^{O(n)}$. A state in the determinized automaton of the Safra construction is a labeled tree of sets of states of the given Büchi automaton. The transition between states, and the acceptance condition relies on the nodes labeling. Piterman [25] simplified the labeling and provided a determinized automaton based on Parity acceptance condition, which is simpler and better for many applications [25]. Schewe [30] moved the acceptance condition from states to edges. In 1995, Muller and Schupp [23] proposed a different exponential determinization construction. Kähler and Wilke [11] identified from [23] the so called *skeleton tree* and proposed a construction unifying the determinization, complementation and disambiguation problems of Büchi automata. The most recent works in this thread of research introduce the notion of *profile trees* where a profile is the history of visits to accepting states [33, 6, 7].

A related line of research is that of finding a Büchi complementation construction that does not rely on Büchi determinization [31, 12, 15, 32, 16, 9, 8, 35]. Avoiding determinization, or the Safra construction, was also pursued in other contexts e.g. games and tree automata [17], compositional synthesis [14] and translating linear temporal logic to automata [5].

2 Preliminaries

Buchi and Parity Automata. An automaton $\mathcal{A}$ is a tuple $(\Sigma, Q, I, \delta, \alpha)$ where Σ is the alphabet, Q is a set of states, $I \subseteq Q$ is the set of initial state, $\delta : Q \times \Sigma \to 2^Q$ is the transition relation, and α is the acceptance condition. A run of the automaton on an infinite word $w = \sigma_1\sigma_2\sigma_3 \ldots$ is a sequence of states $q_0, q_1, q_2, \ldots$ such that $q_0 \in I$ and $q_i \in \delta(q_{i-1}, \sigma_i)$ for every $i \geq 1$. For Büchi automata the acceptance condition is a set $F \subseteq Q$ and a run is *accepting* if it visits states in F infinitely often. We use $\overline{F}$ to denote the set of non-accepting states, i.e. $Q \setminus F$. The acceptance condition of a Parity automaton is a coloring (or ranking) function χ that associates with each state a color (or rank) from a given finite set of colors $\{0, 1, 2, \ldots, m\}$. A run of the automaton is considered *accepting* iff the minimal color visited infinitely often is odd. An automaton may have several runs on the same word, and it accepts a word iff one of its runs on that word accepts it. An automaton is *deterministic* if $|I| = 1$ and $|\delta(q, \sigma)| = 1$ for every $q \in Q$ and $\sigma \in \Sigma$. A deterministic automaton has a single run on each word.

We refer to automata by three letter acronyms. The first letter may be D or N signifying if the automaton is deterministic or non-deterministic, the second letter may be B, P or F signifying whether it is a Büchi automaton, a Parity automaton or an automaton on finite words. The third letter signifies the object on which the automaton runs, which is W for words in all automata in this paper. For example, NBW stands for non-deterministic Büchi automaton on words, and DFW stands for deterministic finite automaton on words.

Words and Trees. We use the term word for a finite or infinite sequence of letters. We use $w[j]$ for the j-the letter of w, $w[j..]$ for the suffix of w starting at j, $w[i..j]$ for the infix of w starting at i and ending in j, and $w[..j]$ for the prefix of w up to z. Note that $w[1]$ is the

first letter of w, so is $w[..1]$. We use $w[..0]$ for the empty string. In general, we use $[1..n]$ for the set $\{1, 2, \ldots, n\}$.

We use annotated binary trees. The nodes of such trees are strings in $\{0,1\}^*$. The root node is the empty string ϵ. The left successor of a node t is $t0$ and the right successor is $t1$. An *annotated binary tree* is a function from $\{0,1\}^*$ to some domain D. A node mapped to $\emptyset$ is regarded as absent from the tree. The *depth* of node t, denoted $|t|$, is its distance from the root, and it equals the length of the string t. So the depth of the root is 0, and the depth of its successors are 1. For $t_1, t_2 \in \{0,1\}^*$ we say that $t_1 <_{\text{lex}} t_2$ if the string t_1 is lexicographically smaller than t_2. We say that $t_1 <_{\text{lft}} t_2$ if $|t_1| = |t_2|$ and $t_1 <_{\text{lex}} t_2$. The *i-th level* of an annotated tree T consists of all nodes in depth i. If $t_1 <_{\text{lex}} t_2 <_{\text{lex}} \cdots <_{\text{lex}} t_n$ are all nodes the i-th level of T then the *i-th slice* of T is the sequence $\langle T(t_1), T(t_2), \ldots, T(t_n)\rangle$. The width of the i-th level is the number of nodes in that level.

Summarizing Runs Concisely. The main task of a determinization construction is to find a way to summarize all the information needed on all the runs of a given non-deterministic automaton in the single run of the constructed deterministic automaton. On finite words, it sufficed to record the set of reachable states. It is instructive to see why this approach fails for Büchi automata. The unfamiliar reader is referred to the full version.

Kähler and Wilke [11] introduced the *split tree*, the *reduced tree* and the *skeleton tree*, all of which concisely summarize the information needed on the runs of the non-deterministic Büchi automaton. The split- reduced- and skeleton-trees are defined per a given word w. A key invariant that is maintained is that if there exists an accepting run of the automaton on w then there is an accepting infinite path in all of these trees. The formal definitions of these trees is given Appendix B of the full version. Roughly speaking, the *split tree* refines the subset construction by separating accepting and non-accepting states. From each node of the tree the left son holds the accepting states and the right son the non-accepting. Thus an accepting path has infintely many left turns, and is also referred to as *left recurring*. The width of a slice of the split-tree is generally unbounded. The *reduced tree* bounds the number of nodes on a slice of the tree to n, the number of states of the given Büchi automaton $\mathcal{A}$, by eliminating from a node of the tree all states that appeared in a node to its left. The *skeleton-tree* is the smallest sub-tree of the reduced-tree that contains all its infinite paths.

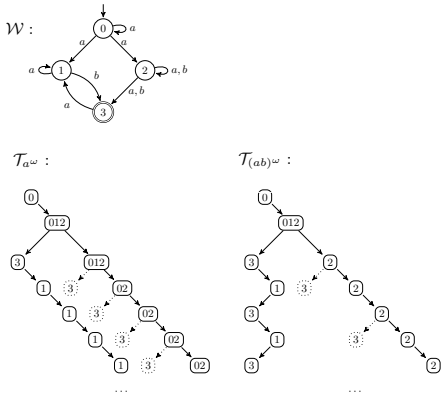


Figure 3 A Büchi automaton $\mathcal{W}$ and its reduced and skeleton trees for a^ω and $(ab)^\omega$ (where a node labeled with multiple digits stands for the subset containing the corresponding states (e.g. 012 stands for $\{0,1,2\}$).

▶ **Example 1.** Figure 3 shows a Büchi automaton $\mathcal{W}$ and the reduced and skeleton trees for a^ω and $(ab)^\omega$ both of which have width 2. The word a^ω is rejected and indeed none of the two skel-paths is left recurring. The word $(ab)^\omega$ is accepted and indeed one of the two skel-paths is left recurring.

Computing the slices of the reduced tree. The skeleton tree thus concisely captures whether the original automaton $\mathcal{A}$ has an accepting path on the given word w. Unfortunately, it requires knowing or guessing which states will eventually have no successors, which seems

a difficult if not impossible task for a deterministic automaton to conduct. Working with the reduced tree, however, is practical.

A *slice* of the reduced tree is a sequence of length at most n of nodes of the reduced tree i.e. an element of $(2^Q)^{\leq n}$. The 0-th slice is $\langle I \rangle$. Given a slice $\mathbb{S} = \langle S_1, S_2, \ldots, S_k \rangle$ we can compute the next slice with respect to a next letter σ as follows [11]. For $1 \leq i \leq k$, let

$$\tilde{S}_i = \cup \{S'_j \mid j < 2i\} \qquad S'_{2i} = \delta(S_i, \sigma) \cap F \setminus \tilde{S}_i \qquad S'_{2i+1} = \delta(S_i, \sigma) \cap \overline{F} \setminus \tilde{S}_i$$

Let $\mathbb{S}' = \langle S'_2, S'_3, \ldots, S'_{2k+1} \rangle$. Let $\mathbb{S}'' = \langle S''_1, S''_2, \ldots, S''_\ell \rangle$ be the sequence obtained from $\mathbb{S}'$ by deleting all the empty sets. Then the next slice of $\mathbb{S}$ with respect to $\mathcal{A}$ and σ, denoted $\delta_{\mathbf{RS}}(\mathbb{S}, \sigma)$ is $\mathbb{S}''$. We define $\delta_{\mathbf{RS}}$ also for a given interval $\mathbb{I} = \langle S_i, ..., S_j \rangle$ of the slices' nodes. Let $\mathbb{I}' = \langle S'_{2i}, S'_{2i+1}, \ldots, S'_{2j+1} \rangle$. Let $\mathbb{I}''$ be the sequence obtained from $\mathbb{I}'$ by deleting all the empty sets. Then $\delta_{\mathbf{RS}}(\mathbb{S}, \mathbb{I}, \sigma) = \mathbb{I}''$.

3 Determinizing assuming the width is one

We first tackle the simplest case where the skeleton's width is one, i.e, there is a single infinite run in the reduced tree. Our automaton then needs to check whether this path visits the accepting set infinitely often. The problem is that we can process the reduced tree, not the skeleton tree, so if we encounter an accepting node we don't know if it is on the skeleton tree or not. Demanding that there exists infinitely many slices where *all* the nodes are accepting is too strong, as can be seen by considering $\mathcal{T}^{\mathcal{A}}_{a^\omega}$ of Fig. 4, which although is accepting all its slices consists at least one non-accepting node. Demanding that there exists infinitely many slices where *there exists* at least one node which is accepting is too weak, as can be seen by considering $\mathcal{T}^{\mathcal{R}}_{a^\omega}$ of Fig. 4, which although is rejecting all its slices consists at least one accepting node.

So we need something a little more sophisticated. The breakpoint construction of Miyano and Hayashi [22] (henceforth, the MH-construction) answers whether all infinite runs of a given non-deterministic Büchi automaton are accepting. Since, in the case considered here, we have just one infinite path, asking whether *all* infinite paths are accepting, is the same as asking whether *the single* infinite path is accepting. Thus, applying the MH-construction on the slices of the reduced tree achieves what we want, when the skeleton width is one.

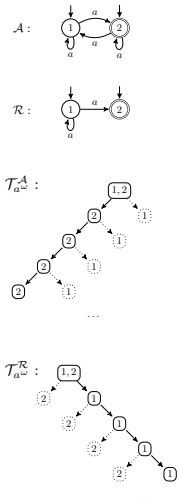


Figure 4 The reduced and skeleton trees of a^ω for $\mathcal{A}$ and $\mathcal{R}$.

The Miyano-Hayashi Construction. The idea of the MH-construction is, in addition to tracking the successors of the current level as in the subset construction, to maintain a bookkeeping about which path has *visited the accepting set recently*. Other states are considered to *owe* a visit to the accepting set. Each state of a layer thus carries with it a bit informing whether it owes a visit or not. When a new layer is constructed the sons of states that owe a visit, are also marked as owing a visit unless they are accepting (in which case the corresponding path has just paid its debt). The other states on the layer are not marked as owing since they are known to have visited the accepting set recently. When none of the states is marked as owing we have found evidence for all of them to visit the accepting set recently. We thus start charging them again for a visit to the accepting set. Such a step is considered a *reset* step. Visiting an accepting state recently thus means visiting an accepting state since the last reset occurred. If there are infinitely many reset steps, we know that between every two adjacent reset points all paths have visited an

accepting state at least once, and therefore all paths have visited the accepting set infinitely often.

Formally, if $\mathcal{A} = (\Sigma, Q, Q_0, \delta, F)$ then the MH-construction results in $\mathcal{E} = (\Sigma, Q_\mathcal{E}, Q^0_\mathcal{E}, \delta_\mathcal{E}, F_\mathcal{E})$ defined as follows. One can think of a state of $\mathcal{E}$ as a mapping $f : Q \to \{\mathsf{a}, \mathsf{o}, \mathsf{v}\}$ where a state of $\mathcal{A}$ mapped to a is *absent* from the layer, a state mapped to o is on the layer and *owes* a visit, and a state mapped to v is in the layer and *visited* the accepting set since the last reset. Alternatively, we can think of the states of $\mathcal{E}$ as pairs of subsets $\langle Q_L, Q_O\rangle$ where Q_L are the states in the layer (those that are mapped to v or o) and Q_O are the states in the layer that owe a visit (those that are mapped to o). The transition relation is then defined as $\delta_\mathcal{E}(\langle Q_L, Q_O\rangle, \sigma) = \langle Q'_L, Q'_O\rangle$ where $Q'_L = \delta(Q_L, \sigma)$ and if $Q_O \neq \emptyset$ then $Q'_O = \delta(Q_O, \sigma) \setminus F$ and otherwise (when $Q_O = \emptyset$) $Q'_O = \delta(Q_L, \sigma) \setminus F$. The accepting states of $\mathcal{E}$ are the reset states, i.e. those of the form $\langle Q_L, \emptyset\rangle$. The initial state is $\langle I, \emptyset\rangle$.

▶ **Claim 1** ([22]). *$\mathcal{E}$ accepts w iff all paths of $\mathcal{A}$ accept w.*

Miyano-Hayashi on the Reduced Tree. The reduced tree already computes the next successors and separates the states into accepting and non-accepting. The remaining task is to carry the bookkeeping bit of the MH-construction. This can be achieved by the DBW $D_1 = (\Sigma, Q_1, I_1, \delta_{\mathbf{MHoR}}, F_1)$ as follows. The states Q_1 are annotated slices of the form $\langle (S_1, b_1), (S_2, b_2), \ldots, (S_k, b_k)\rangle$. That is, $Q_1 \subseteq (2^Q \times \{\mathsf{o}, \mathsf{v}\})^{\leq n}$. The accepting states F_1 are those where all the b_i components are v. The initial state I_1 is $\langle (I, \mathsf{o})\rangle$. The transition relation builds the next slice of the reduced tree, and annotates the node as dictated by the MH-construction. (A node of the reduced tree is considered to be *accepting* if it consists of accepting states, otherwise it is considered *non-accepting*. Note that in the reduced tree there are no nodes with both accepting and rejecting states.) Formally, given an annotated slice $\mathbb{S} = \langle (S_1, b_1), (S_2, b_2), \ldots, (S_k, b_k)\rangle$, a next letter σ, for $1 \leq i \leq k$, let

$$\begin{array}{lll} \tilde{S}_i & = \cup \{S'_j \mid j < 2i\} & b'_{2i} = \mathsf{v} \\ S'_{2i} & = \delta(S_i, \sigma) \cap F \setminus \tilde{S}_i, & b'_{2i+1} = \begin{cases} \mathsf{o} & \text{if } \forall i.\ b_i = \mathsf{v} \\ b_i & \text{otherwise} \end{cases} \\ S'_{2i+1} & = \delta(S_i, \sigma) \cap \overline{F} \setminus \tilde{S}_i & \end{array}$$

Let $\mathbb{S}' = \langle (S'_2, b'_2), (S'_3, b'_3), \ldots, (S'_{2k+1}, b'_{2k+1})\rangle$. Let $\mathbb{S}''$ be the sequence obtained from $\mathbb{S}'$ by deleting all pairs (S'_i, b'_i) where S'_i is the empty set. Then $\delta_{\mathbf{MHoR}}(\mathbb{S}, \sigma) = \mathbb{S}''$.

▶ **Proposition 1.** *For any word w for which skel-width$(\mathcal{A}, w) = 1$ the* DBW *constructed above accepts w iff $\mathcal{A}$ accepts w.*

Similar to $\delta_{\mathbf{RS}}$, for later constructions only, we define $\delta_{\mathbf{MHoR}}(\mathbb{S}, \mathbb{I}, \sigma)$ for an interval $\mathbb{I} = \langle (S_i, b_i), \ldots, (S_j, b_j)\rangle$ of $\mathbb{S}$, for some $1 \leq i \leq j \leq k$. Let $\mathbb{I}'$ be the sequence $\langle (S'_{2i}, b'_{2i}), (S'_{2i+1}, b'_{2i+1}), \ldots, (S'_{2j+1}, b'_{2j+1})\rangle$. Let $\mathbb{I}''$ be the sequence obtained from $\mathbb{I}'$ by removing all pairs whose first component is the empty set then $\delta_{\mathbf{MHoR}}(\mathbb{S}, \mathbb{I}, \sigma) = \mathbb{I}''$.

4 Deciding the width

Perhaps the simplest question regarding widths is: *what is the width of the tree?* or put as a decision problem: *is the width of the tree smaller than a given k?* Before tackling this problem, let's consider a simplified case as a motivating discussion. We know the width of the skeleton tree is monotonically non-decreasing, and bounded by n. Suppose we could start at a time point z after the width has already stabilized. Suppose we could track each of the nodes separately from the others, within its own *interval*. If we have ℓ nodes in level z

we would start with ℓ intervals. The successor of a node in interval j are kept within the same interval j.

If the width is k, then exactly k of the nodes we started with at level z have infinitely many descendants, therefore their interval will always be non-empty. If we started with $\ell > k$ nodes, then $\ell - k$ of these have only finitely many descendants and therefore eventually their interval will become empty. If we throw away empty intervals, then we will at some point remain with exactly k non-empty intervals, forever long. Below we develop this intuition to tackle the exact problem of deciding the width.

The states of the constructed DBW $\mathcal{B}_k$ are sequences of sequences of nodes of the reduced tree, corresponding to a slice partitioned into several intervals. While a node of a reduced tree is a subset of $\mathcal{A}$'s states, in this construction a node can be thought of as the smallest element. If $\mathbb{S} = \langle \mathbb{I}_1, \mathbb{I}_2, \ldots, \mathbb{I}_\ell \rangle$ we say that $\mathbb{S}$ has ℓ intervals, and denote it $|\mathbb{S}| = \ell$. If $|\mathbb{S}| < k$ we put all nodes of the next step in different (singleton) intervals, to track each of them separately. We call such step a *shredding* step, and such state a *shredding state*. The accepting set is the set of all shredding states. The initial state consists of a single interval $\mathbb{S}_0 = \langle \mathbb{I}_1 \rangle$ where $\mathbb{I}_1 = \langle I \rangle$, i.e. the interval consists of one node – the root of the reduced tree.

We first define the transition relation for a non-shredding state. Let $\mathbb{S} = \langle \mathbb{I}_1, \mathbb{I}_2, \ldots, \mathbb{I}_\ell \rangle$ be a state of $\mathcal{B}_k$. We use $\cup\mathbb{S}$ to abbreviate $\cup_{1 \le i \le \ell} \mathbb{I}_i$, i.e. the set of nodes in all the intervals together. For $1 \le i \le \ell$ let $\mathbb{I}'_i = \delta_{\mathbf{RS}}(\cup\mathbb{S}, \mathbb{I}_i, \sigma)$ and let $\mathbb{S}' = \langle \mathbb{I}'_1, \mathbb{I}'_2, \ldots, \mathbb{I}'_\ell \rangle$. Let $\mathbb{S}''$ be the sequences obtained from $\mathbb{S}'$ by deleting empty intervals. Then if $\mathbb{S}$ is non shredding (i.e. $\ell \ge k$) then $\delta_{\mathcal{B}_k}(\mathbb{S}, \sigma) = \mathbb{S}''$.

To define the transition relation for a shredding state we introduce an additional notation. If $\cup\mathbb{S} = \{S_1, S_2, \ldots, S_m\}$ then we use $\mathit{Shred}(\mathbb{S})$ to denote the sequence $\langle \mathbb{I}'_1, \mathbb{I}'_2, \ldots, \mathbb{I}'_m \rangle$ where $\mathbb{I}'_i = \langle S_i \rangle$. That is, in $\mathit{Shred}(\mathbb{S})$ every node of $\mathbb{S}$ is put in its own interval. Now let $\mathbb{I}''_i = \delta_{\mathbf{RS}}(\cup\mathbb{S}, \mathbb{I}'_i, \sigma)$ and let $\mathbb{S}'' = \langle \mathbb{I}''_1, \mathbb{I}''_2, \ldots, \mathbb{I}''_\ell \rangle$. Let $\mathbb{S}'''$ be the sequences obtained from $\mathbb{S}''$ by deleting empty intervals. Then if $\mathbb{S}$ is a shredding state (i.e. $\ell < k$) then $\delta_{\mathcal{B}_k}(\mathbb{S}, \sigma) = \mathbb{S}'''$.

▶ **Proposition 2.** *The* DBW $\mathcal{B}_k$ *constructed above accepts a word* w *iff* $\mathit{skel\text{-}width}(\mathcal{A}, w) < k$.

▶ **Example 2.** Consider the NBW $\mathcal{W}$ of Figure 3. Both words a^ω and $(ab)^\omega$ have width 2 (as is evident from the respective skeleton trees). The automata $\mathcal{B}_2$ and $\mathcal{B}_3$ for widths < 2 and < 3 respectively, are given in Fig. 5. The accepting states are marked with a double edge (and also colored blue). It can be seen that $\mathcal{B}_2$ rejects both a^ω and $(ab)^\omega$ as required (since their width is not smaller than 2), while $\mathcal{B}_3$ accepts both words as required (since their width is smaller than 3).

Tracking the path of either words a^ω or $(ab)^\omega$ on $\mathcal{B}_2$ or $\mathcal{B}_3$ (and comparing to the respective trees in Figure 3) we can see how a state represents a slice of the reduced tree, separated into intervals. And that the transition relation maintains the successors within the same interval, unless shredding occurred in which case they are separated into singleton intervals.

Last, we can see that each skeleton path is eventually trapped within its own interval. This brings us to our next claim.

▶ **Claim 2** (skel-paths separation). *Let* w *be a word with skel-width* k*, and let* $\rho_1, \rho_2, \ldots, \rho_k$ *be the skel-paths from left to right. Let* $\mathbb{S}_0, \mathbb{S}_1, \mathbb{S}_2, \ldots$ *be the run of* $\mathcal{B}_k$ *on* w*. Then there exists* $z_1 \in \mathbb{N}$ *such that* $|\mathbb{S}_z| = k$ *for every* $z > z_1$*. Moreover, assume* $\mathbb{S}_z = \langle \mathbb{I}^z_1, \mathbb{I}^z_2, \ldots, \mathbb{I}^z_k \rangle$ *then for every skel-path* $\rho_i = S^i_0, S^i_1, S^i_2, \ldots$*, for every* $z > z_1$*, we have* $S^i_z \in \mathbb{I}^z_i$.

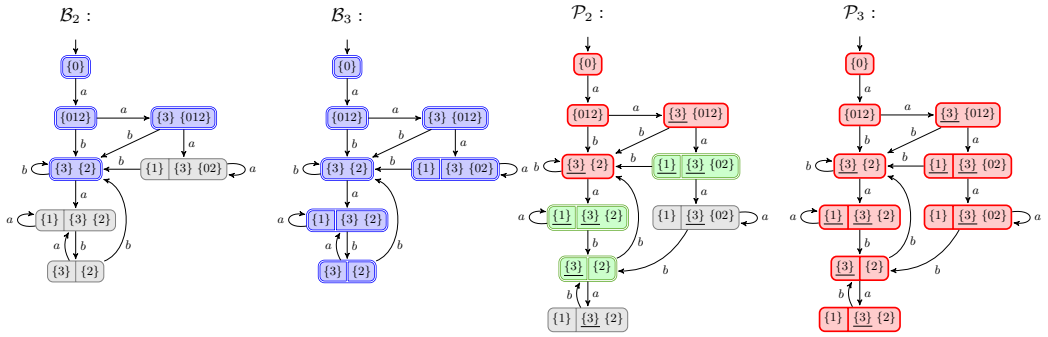


Figure 5 On the left: automata $\mathcal{B}_2$ and $\mathcal{B}_3$ answering whether the width of a word w.r.t. $\mathcal{W}$ given in Fig. 3 is smaller than 2 and 3, respectively. The accepting states are those with a double frame (also colored blue). On the right: automata $\mathcal{P}_2$ and $\mathcal{P}_3$ answering whether a word with width exactly 2 and 3, resp. (w.r.t. $\mathcal{W}$ given in Fig. 3) is accepted by $\mathcal{A}$. The states with a thick frame have color 0 (red). The states with a double frame have color 1 (green). The states with a thin frame have color 2 (gray). An underlined node is a node whose MH-bit is set to v.

5 Determinizing for a given width

We now show that we can build a deterministic parity automaton $\mathcal{P}_k$ that accepts a word of width k iff it is accepted by $\mathcal{A}$. The idea is to combine both constructions we have seen previously. That is, we use the idea of the construction of $\mathcal{B}_k$ to keep the skel-paths separated one from the other in different intervals, and we use the MHoR construction to check on each of these intervals whether the skel-path is accepting. Essentially, we add the Miyano-Hayashi bit to the construction of $\mathcal{B}_k$.

As in $\mathcal{B}_k$ we work with k intervals, and apply shredding when the number of intervals is smaller than k. Instead of using $\delta_{\mathbf{RS}}$ as in $\mathcal{B}_k$ we use $\delta_{\mathbf{MHoR}}$ as in $\mathcal{D}_1$, so the transition relation computes for each successor of a node its MH-bit, and an MH-reset is preformed when all bits of that interval are v. The states of $\mathcal{P}_k$ are sequences of at most n intervals, where each interval is a sequence of pairs whose first element is a node of the reduced tree, and whose second element is a bit in $\{\mathsf{o},\mathsf{v}\}$. (Again, nodes are subsets of $\mathcal{A}$'s states, but can be thought of as the smallest elements of this construction.) The initial state is the sequence $\mathbb{S}_0$ where $\mathbb{S}_0 = \langle \mathbb{I}_1 \rangle$ and $\mathbb{I}_1 = \langle (I, \mathsf{o}) \rangle$, i.e. the root of the reduced tree, labeled as owing.

Let $\mathbb{S} = \langle \mathbb{I}_1, \mathbb{I}_2, \ldots, \mathbb{I}_\ell \rangle$ be a state of $\mathcal{P}_k$. First we define a non-shredding transition (i.e. assume $\ell \geq k$). In this case, for $1 \leq i \leq \ell$ and $\sigma \in \Sigma$, let $\mathbb{I}'_i = \delta_{\mathbf{MHoR}}(\cup\mathbb{S}, \mathbb{I}_i, \sigma)$ (where $\delta_{\mathbf{MHoR}}$ is as defined in Section 3). Let $\mathbb{S}' = \langle \mathbb{I}'_1, \mathbb{I}'_2, \ldots, \mathbb{I}'_k \rangle$ and let $\mathbb{S}''$ be the sequence obtained from $\mathbb{S}'$ by deleting all pairs with an empty interval. Then $\delta_{\mathcal{P}_k}(\mathbb{S}, a) = \mathbb{S}''$.

For a shredding transition (i.e. when $\ell < k$), if $\cup\mathbb{S} = \{P_1, P_2, \ldots, P_m\}$ where P_i is a pair (S_i, b_i) where S_i is a reduced-tree node and $b_i \in \{\mathsf{o},\mathsf{v}\}$ then we use $\mathit{Shred}(\mathbb{S})$ to denote the sequence $\langle \mathbb{I}'_1, \mathbb{I}'_2, \ldots, \mathbb{I}'_m \rangle$ where $\mathbb{I}'_i = \langle P_i \rangle$. That is, in $\mathit{Shred}(\mathbb{S})$ every pair of $\mathbb{S}$ is put in its own interval. Now let $\mathbb{I}''_i = \delta_{\mathbf{MHoR}}(\cup\mathbb{S}, \mathbb{I}'_i, \sigma)$ and let $\mathbb{S}'' = \langle \mathbb{I}''_1, \mathbb{I}''_2, \ldots, \mathbb{I}''_\ell \rangle$. Let $\mathbb{S}'''$ be the sequences obtained from $\mathbb{S}''$ by deleting the empty intervals. Then $\delta_{\mathcal{P}_k}(\mathbb{S}, \sigma) = \mathbb{S}'''$.

Let $\mathbb{S}$ be a state. The coloring function assigns it 0 if it is a shredding states (i.e. $|\mathbb{S}| < k$); it assigns it 1 if $|\mathbb{S}| \geq k$ and an MH-reset occurs, (i.e. for some interval i all the MH-bits of pairs in that interval are v); and it assigns it 2 otherwise.

▶ **Proposition 3.** *For all words w if skel-width$(\mathcal{A}, w) = k$ then $\mathcal{P}_k$ accepts w iff $\mathcal{A}$ does.*

▶ **Example 3.** Consider again the NBW $\mathcal{W}$ of Figure 3. Figure 5 provides the automata $\mathcal{P}_2$ and $\mathcal{P}_3$ for widths 2 and 3, respectively. The red states (also having a thick frame) have color 0, the green states (also having a double frame) have color 1 and the gray states have color 2. An underlined node corresponds to a node whose MH-bit is set to v. (The un-underlined nodes have their MH-bit set to o.)

It is easy to construct $\mathcal{P}_2$ and $\mathcal{P}_3$ given $\mathcal{B}_2$ and $\mathcal{B}_3$. Indeed they just add the MH-bit for the nodes of $\mathcal{B}_2$ and $\mathcal{B}_3$, respectively, undergoing an MH-reset, if all nodes of an interval are underlined (i.e. have visited the accepting set recently). Note that $\mathcal{P}_2$ has more states than $\mathcal{B}_2$ since for instance, we need two version of the lower right state, for different settings of the node's MH-bits.

Recall that both words a^ω and $(ab)^\omega$ have width 2, thus $\mathcal{P}_2$ should provide the correct result for both of them. Indeed $\mathcal{P}_2$ accepts $(ab)^\omega$ and rejects a^ω exactly as $\mathcal{A}$ does.

We note that $\mathcal{D}_1$ is a private case of the construction of $\mathcal{P}_k$ for $k = 1$. Indeed, for $k = 1$ shredding will never occur, and so all states consist of a single interval which is the entire slice. Thus $\mathcal{P}_1$, exactly as $\mathcal{D}_1$, simply tracks the MH-bits on the reduced tree.

We next turn to understand what answer $\mathcal{P}_k$ provides for words with skel-width different from k. If the actual skel-width is k_* and $k > k_*$ then after the stabilization point, we won't be able to maintain for long more than k_* non-empty intervals, therefore shredding will occur infinitely often and $\mathcal{P}_k$ will reject. If $k_* > k$ then as we state in Proposition 4 below, if $\mathcal{P}_k$ accepts it does so rightfully.

▶ **Proposition 4.** *The* DPW $\mathcal{P}_k$ *described above uses three colors* $\{0, 1, 2\}$ *and for any word* w

- *if skel-width*$(\mathcal{A}, w) = k$ *then* $\mathcal{P}_k$ *accepts* w *iff* $\mathcal{A}$ *does,*
- *if skel-width*$(\mathcal{A}, w) < k$ *then* $\mathcal{P}_k$ *rejects* w*,*
- *if* $\mathcal{P}_k$ *accepts* w *then* w *is accepted by* $\mathcal{A}$*, and*
- $\mathcal{P}_k$ *visits* 0 *infinitely often iff skel-width*$(\mathcal{A}, w) < k$*.*

Self-Correction

We observe that in a sense, the intervals are self-correcting. That is, for every word w and every suffix $w[j..]$ of w, if we are given the slice of the reduced tree on the respective prefix $w[1..j-1]$, partition it arbitrarily to intervals, and apply $\mathcal{B}_k$ from there on that suffix, we will still get a correct result (i.e. a correct answer to the question whether the width of w is smaller than k). Intuitively, because the intervals' role is to detect a property that depends only on the future.

In a similar manner, the MH-bits are self correcting. That is, for every word w and every suffix $w[j..]$ of w, if we are given the slice of the reduced tree on the respective prefix $w[1..j-1]$, and annotate it arbitrarily with MH-bits, and apply MHoR from there on that suffix, we will still get a correct result (i.e. an answer whether w is accepted on all skel-paths). Again, intuitively because the role of the MH-bits is to detect a property that depends only on the future. Putting these together we get the following claim.

▶ **Claim 3** (Intervals and MH-bits are self-correcting). *Let* $\mathbb{S}$ *be a slice of the reduced tree on prefix* $w[1..j]$ *partitioned arbitrarily into intervals, and annotated arbitrarily with MH-bits. Applying the construction of* $\mathcal{P}_k$ *on the suffix* $w[j+1..]$ *from state* $\mathbb{S}$ *still satisfies all premises of Proposition 4.*

6 Determinizing for any width

We now build a deterministic parity automaton $\mathcal{P}$ that recognizes the same language as $\mathcal{A}$. Intuitively, we want to run the automata $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$ from the previous section, in parallel. We know that if the exact width is k_* then the answer we look for is given by $\mathcal{P}_{k_*}$ and apart from this, $\mathcal{P}_k$ for all $k > k_*$ rejects, and for all $k < k_*$, if $\mathcal{P}_k$ accepts it does so rightfully. Thus, prioritizing results of $\mathcal{P}_k$'s with lower k's should give us the correct result.

Before we continue, let's contemplate on the complexity we'll get by running all $\mathcal{P}_k$'s in parallel. A state of $\mathcal{P}_k$ needs to encode, among other things, the partition into intervals, which requires at least 2^n states. Since we have n such components in a state, we'd need 2^{n^2} which is more that we can allow, if we'd like to stay in the bounds of $2^{O(n \log n)}$.

We overcome this by working with a modification of $\mathcal{P}_k$, termed $\mathcal{R}_k$, which is obtained using the observation of Claim 3. Specifically, $\mathcal{R}_k$ will differ from $\mathcal{P}_k$ in two places. One is that $\mathcal{R}_k$ will undergo shredding whenever one of the $\mathcal{R}_j$ with $j \leq k$ decides to undergo shredding. This entails that the partitions to intervals of $\mathcal{R}_{k+1}$ will be a refinement of that of $\mathcal{R}_k$. Or put otherwise, every interval of $\mathcal{R}_{k+1}$ would be fully contained in an interval of $\mathcal{R}_k$. The second place is that an interval of $\mathcal{R}_k$ will undergo an MH-reset whenever a subsuming interval of one of the $\mathcal{R}_j$'s for $j \leq k$ decides to undergo an MH-reset. As we shall see in the next subsection, this will enable an encoding of the state space that does not exceed $n^{O(n)}$.

To run these automata in parallel a state of our parity automaton will have n components one corresponding to each of the automata $\mathcal{R}_k$. The initial state will be the n-tuple consisting of the initial states of $\mathcal{R}_1, \mathcal{R}_2 \ldots, \mathcal{R}_n$ and the transition relation will follow that of the $\mathcal{R}_k$'s. For the acceptance condition, we need to assign colors to these compound states.

A compound state has the following form $(\mathbb{S}_1, \mathbb{S}_2, \ldots, \mathbb{S}_n)$ where $\mathbb{S}_i$ is a state of $\mathcal{R}_i$ for $1 \leq i \leq n$. To assign a color to the compound state, it is convenient to let each $\mathcal{R}_i$ use its own set of colors. More precisely, we will assume automaton $\mathcal{R}_k$ assigns colors in $\{2k,\ 2k{+}1,\ 2n{+}2\}$ instead of $\{0, 1, 2\}$, respectively. That is, the colors $2k$ and $2k{+}1$ are uniquely used by $\mathcal{R}_k$ whereas the color $2n{+}2$ may be used by all $\mathcal{R}_k$'s. Now for a given state $\mathbb{P} = (\mathbb{S}_1, \mathbb{S}_2, \ldots, \mathbb{S}_n)$, we can look at the corresponding sequence of colors $(c_1, c_2, \ldots, c_n)$, where c_i has values in $\{2k,\ 2k{+}1,\ 2n{+}2\}$, and color the compound state $\mathbb{P}$ with the minimal color among the c_i's.

▶ **Theorem 4.** *Let $\mathcal{A}$ be an* NBW*, with n states. The* DPW *$\mathcal{P}$ described above has $2n + 1$ colors and it accepts an infinite word w iff $\mathcal{A}$ accepts w.*

Complexity. We turn to show that the complexity of the construction of the DPW $\mathcal{P}$ in Theorem 4 is $n^{O(n)}$.

A state $\mathbb{S}_k$ of $\mathcal{R}_k$ for some $1 \leq k \leq n$ is a slice of the reduced tree, partitioned into intervals, where each node is annotated with its MH-bit. A state of $\mathcal{P}$ is an n-tuple $(\mathbb{S}_1, \mathbb{S}_2, \ldots, \mathbb{S}_k)$ of such states. We shall see that we can encode a state of $\mathcal{P}$ more succinctly than directly encoding each state $\mathbb{S}_i$ of the tuple. In fact, we chose to work with $\mathcal{R}_k$ instead of $\mathcal{P}_k$ for this reason exactly. For $\mathcal{R}_k$ we can show that the intervals of $\mathcal{R}_{k+1}$ refine those of $\mathcal{R}_k$. This will entail also that the MH-bits for all $\mathcal{R}_k$'s can be encoded more compactly.

First, we claim that the intervals of $\mathcal{R}_{k+1}$ refine those of $\mathcal{R}_k$, for every $1 \leq k \leq n$.

▶ **Claim 4** (Intervals refinement)**.** *Every interval $\mathbb{J}_m$ of the state of $\mathcal{R}_{k+1}$ after reading $w[..z]$, is fully contained in an interval $\mathbb{I}_\ell$ of the state of $\mathcal{R}_k$ after reading $w[..z]$, for any $z \in \mathbb{N}$.*

Next, we show that if $b_1, b_2, \ldots, b_n$ are the MH-bit of a given node in $\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_n$ resp. then $b_1 b_2 \ldots b_n \in \mathsf{v}^*\mathsf{o}^*$.

▶ **Claim 5** (MH-bits entailment). *Let $b_k(S)$ and $b_{k+1}(S)$ be the MH-bit encoding of node S in the state of $\mathcal{R}_k$ and $\mathcal{R}_{k+1}$, resp. after reading $w[..z]$ for some $z \in \mathbb{N}$. Then $b_k(S) = \mathsf{o}$ implies $b_{k+1}(S) = \mathsf{o}$, and $b_{k+1}(S) = \mathsf{v}$ implies $b_k(S) = \mathsf{v}$.*

We are now ready to fully encode a state $\mathbb{P} = (\mathbb{S}_1, \mathbb{S}_2, \ldots, \mathbb{S}_n)$ of $\mathcal{P}$. A state $\mathbb{S}_i$ needs to record (1) the original automaton states that are on the slice, in the order they appear, (2) the partition to nodes, (3) the partition to intervals and (4) the Miyano-Hayshi bit of each node. We have that (1) and (2) are shared among all $\mathbb{S}_i$'s since they all track the obtained slice of the reduced tree. We'll use Claims 4 and 5 to represent (3) and (4) in a combined manner, rather than separately for each $\mathbb{S}_i$.

The states on the slice, can be represented by a permutation on $[1..n]$.[3] This requires $n! < n^{O(n)}$. The partitions of states to nodes, can be encoded using a function h from $[1..n]$ to $\{0, 1\}$, so that $h(i) = 1$ means that a new node starts after the i-th state of slice (i.e. the i-th and $i{+}1$-th states of the slice are in different nodes). This requires $2^n < n^{O(n)}$. The partition to intervals for all $\mathcal{R}_k$'s together, by Claim 4, can be encoded by a function F from $[1..n]$ to $[0..n]$ so that $F(i) = j$ means that a new interval begins between states i and $i{+}1$ of the slice, in all $\mathcal{R}_k$ for $k > j$. This requires $n^{n+1} < n^{O(n)}$. Finally, let $[1..n]_h$ be the set of indices for which $h(i) = 1$. That is, $[1..n]_h$ represents the nodes of the slice. By Claim 5, the MH-bits of nodes in all $\mathcal{R}_k$'s together can be represented by a function G from $[1..n]_h$ to $[1..n+1]$ so that $G(i) = j$ means that node i is v in all $\mathcal{R}_k$ for $k < j$ and it is o, in all $\mathcal{R}_k$ for $k \geq j$. This requires $n^{n+1} < n^{O(n)}$. So all in all, $n^{O(n)}$ suffices to represent all states of $\mathcal{P}$ from Theorem 4.[4]

▶ **Corollary 5.** *Let $\mathcal{A}$ be an* NBW*, with n states. The* DPW *$\mathcal{P}$ described above has $n^{O(n)}$ states and $2n + 1$ colors and it accepts an infinite word w iff $\mathcal{A}$ accepts w.*

7 Conclusions

Building on the reduced and skeleton trees of Kähler and Wilke [11] we provide a novel simple construction for Büchi determinization.

A key observation is that *conceptually* we can partition the reduced tree into *intervals* so that each infinite path of the reduced tree (along with the finite paths attached to it) resides in an interval of its own, as depicted in Figure 2. The word is accepted by $\mathcal{A}$ iff in one of these intervals the infinite path is accepting. The question whether in such an interval the infinite path is accepting, can be answered by applying the MH-construction on that interval.

We show how we can build an automaton $\mathcal{P}_k$ that for words with width k, finds this conceptual separation and by applying the MH-construction to each of the k intervals, returns a correct result for all words of width k. Our overall construction runs in parallel (a minor tweaking of) the automata $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$ to produce a correct result for words of any width.[5]

We have thus broken the determinization problem into two simpler problems (1) *partitioning* the reduced tree into its skeleton-paths, and (2) providing an answer for a *single* infinite path (in the presence of finite paths). For the latter we adapted the MH-construction.

[3] We assume some arbitrary given order $\prec$ on the states of $\mathcal{A}$ so that states in the same node will be ordered according to $\prec$, and we obtain a total order on the states of a slice.

[4] In fact, the partition to nodes can be seen as the finest refinement of the partition to intervals, so we can represent both (2) and (3) together by a function from $[1..n]$ to $[0..n{+}1]$. So the overall number of state is bounded by $n! \cdot n^{n+2} \cdot n^{n+1} < n^{3n+3}$.

[5] Appendix A provides an illustrative presentation of the constructions using tokens, bells, and buzzers.

We tackled the former by answering the elemental decision problem of whether the number of infinite paths in a tree with both finite and infinite paths is smaller than a given k.

Acknowledgements. We would like to thank Nir Piterman for his helpful comments on a draft of this paper.

References

1 R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *In Proc. 38th Symp. on Foundations of Computer Science*, pages 100–109, 1997.

2 J.R. Büchi. Weak second-order arithmetic and finite automata. *Zeit. Math. Logik und Grundl. Math.*, 6:66–92, 1960.

3 A. Church. Applications of recursive arithmetic to the problem of circuit synthesis. In *Summaries of the Summer Institute of Symbolic Logic*, volume I, pages 3–50, 1957.

4 C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

5 J. Esparza and J. Kretínský. From LTL to deterministic automata: A Safraless compositional approach. In *In Proc 26th Int. Conf. on Comp. Aided Verif.*, pages 192–208, 2014.

6 S. Fogarty, O. Kupferman, M.Y. Vardi, and T. Wilke. Profile trees for büchi word automata, with application to determinization. In *GandALF*, pages 107–121, 2013.

7 S. Fogarty, O. Kupferman, T. Wilke, and M.Y. Vardi. Unifying büchi complementation constructions. *ogical Methods in Computer Science*, 9, 2013.

8 E. Friedgut, O. Kupferman, and M.Y. Vardi. Büchi complementation made tighter. *Int. J. Found. Comput. Sci.*, 17:851–868, 2004.

9 S. Gurumurthy, O. Kupferman, F. Somenzi, and M.Y. Vardi. On complementing nondeterministic Büchi automata. In *In Proc. 12th Conf. on CHARME*, volume 2860 of *LNCS*, pages 96–110. Springer, 2003.

10 T.A. Henzinger and N. Piterman. Solving games without determinization. In *Annual Conf. of the Eur. Asso. for Comp. Sci. Log.*, volume 4207 of *LNCS*, pages 394–410, 2006.

11 D. Kähler and T. Wilke. Complementation, disambiguation, and determinization of Büchi automata unified. In *ICALP08*, volume 5125 of *LNCS*, pages 724–735. Springer, 2008.

12 N. Klarlund. Progress measures for complementation of ω-automata with applications to temporal logic. In *In Proc. 32nd Symp. on Found. of Comp. Sci.*, pages 358–367, 1991.

13 D. Kozen. *Theory of Computation*. Texts in Computer Science. Springer, 2006.

14 O. Kupferman, N. Piterman, and M.Y. Vardi. Safraless compositional synthesis. In *Proc 18th of CAV*, volume 4144 of *LNCS*, pages 31–44. Springer, 2006.

15 O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. In *Proc. 5th Israeli Symp. on Theory of Computing and Systems*, pages 147–158, 1997.

16 O. Kupferman and M.Y. Vardi. Synthesizing distributed systems. In *In Proc. 16th IEEE Symp. on Logic in Computer Science*, pages 389–398, 2001.

17 O. Kupferman and M.Y. Vardi. From linear time to branching time. *ACM Transactions on Computational Logic*, 6(2):273–294, 2005.

18 L.H. Landweber. Decision problems for ω–automata. *Mathematical Systems Theory*, 3:376–384, 1969.

19 W. Liu and J. Wang. A tighter analysis of piterman's büchi determinization. *Inf. Process. Lett.*, 109(16):941–945, 2009.

20 R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

21 M. Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, 1988.

22 S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

23 D.E. Muller and P.E. Schupp. Simulating alternating tree automata by nondeterministic automata. *TCS*, 141:69–107, 1995.

24 Jean-Pierre Pécuchet. On the complementation of büchi automata. *Theoretical Computer Science*, 47:95–98, 1986.

25 N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, 2006.

26 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *In Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.

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

28 S. Safra. On the complexity of ω-automata. In *In Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, 1988.

29 S. Safra. Exponential determinization for ω-automata with strong-fairness acceptance condition. In *Proc. 24th ACM Symp. on Theory of Computing*, Victoria, 1992.

30 S. Schewe. Tighter bounds for the determinisation of büchi automata. In *Found. of Soft. Sci. and Comp. Struc. 12th Inter. Conf. FOSSACS*, pages 167–181, 2009.

31 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.

32 W. Thomas. Complementation of büchi automata revised. *Jewels are Forever*, pages 109–120, 1999.

33 M.-H. Tsai, S. Fogarty, M.Y. Vardi, and Y.-K. Tsay. State of büchi complementation. In *15th Int. Conf. on Impl. and Appl. of Aut.*, volume 6482, pages 261–271, 2010.

34 M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *n Proc. 26th IEEE Symp. on Foundations of Computer Science*, pages 327–338, 1985.

35 M.Y. Vardi. The büchi complementation saga. In *In Proc. 24th Symp. on Theoretical Aspects of Computer Science*, pages 12–22, 2007.

36 Q. Yan. Lower bounds for complementation of ω-automata via the full automata technique. In *Proc. 33rd ICALP*, volume 4052 of *LNCS*, pages 589–600. Springer, 2006.

A Recap – The Construction using Tokens, Buzzers and Bells

Below we give a description of our construction, using tokens, bells and buzzers in the style of the representation of Safra's construction in [13]. The tokens are a visual means to describe the states of the automaton. For a parity automaton with 3 colors $\{0, 1, 2\}$ the bell corresponds to a state colored 1 and the buzzer to a state colored 0, so that a word is rejected if the buzzer buzzed infinitely many times, and otherwise, accepted if the bell rang infinitely many times. For a parity automaton with $2m$ colors, we use m buzzers and m bells. Suppose during a run on the word the minimal buzzer to buzz infinitely often is k_{buzz} and the minimal bell to ring infinitely often is k_{bell}. Then $k_{buzz}, k_{bell} \in [1..m] \cup \{\infty\}$ and if $k_{bell} < k_{buzz}$ the word is accepted.

For width one. The construction builds the slices of the reduce tree slice by slice and marks the slices' nodes with *violet* and *orange* tokens (for the MH bits v and o, resp). On the root of the tree, put violet if it is accepting, and orange otherwise. Given the current slice, put tokens on the next slice as follows. For a son of a violet token, put a violet token. For a son of an orange token, put a violet token if it is accepting and an orange token otherwise. If all

tokens in the slice are violet, ring the bell. Then replace the tokens on all the non-accepting nodes with orange tokens.

For width k. The construction builds the slices of the reduce tree slice by slice and marks the slices' nodes with *violet* and *orange* tokens that are numbered 1 to n (where n is the number of states of the given NBW). That is, $\mathit{tokens} = \{\mathit{violet}(i) \mid i \in [1..n]\} \cup \{\mathit{orange}(i) \mid i \in [1..n]\}$. We use $\mathit{token}(i)$ for $\mathit{violet}(i) \vee \mathit{orange}(i)$.

On the root of the tree, put $\mathit{violet}(1)$ if it is accepting, and $\mathit{orange}(1)$ otherwise. Given the current slice, put tokens on the next slice as follows. For a son of a $\mathit{violet}(i)$ token, put a $\mathit{violet}(i)$ token. For a son of an $\mathit{orange}(i)$ token, put a $\mathit{violet}(i)$ token if it is accepting and an $\mathit{orange}(i)$ token otherwise. If the number of i's for which $\mathit{token}(i)$ is in the current slice is less than k, buzz the buzzer. Then put on each node of the slice a new token, with i's increasing from left to right, and with $\mathit{orange}(i)$ placed on a non-accepting node and $\mathit{violet}(i)$ on an accepting node. If for some i, all $\mathit{token}(i)$ are violet, ring the bell. Then replace all the $\mathit{violet}(i)$ tokens that are on a non-accepting node with $\mathit{orange}(i)$ tokens.

Overall construction.[6] Again, the construction builds the slices of the reduce tree slice by slice and marks the slices' nodes with tokens. Here we use n sets of violet and orange tokens, numbered 1 to n. That is, $\mathit{tokens} = \{J\text{-}\mathit{violet}(i) \mid J, i \in [1..n]\} \cup \{J\text{-}\mathit{orange}(i) \mid J, i \in [1..n]\}$. We use $J\text{-}\mathit{token}(i)$ for $\{J\text{-}\mathit{violet}(i), J\text{-}\mathit{orange}(i)\}$ and $J\text{-}\mathit{token}()$ for $\{J\text{-}\mathit{violet}(i) \mid i \in [1..n]\} \cup \{J\text{-}\mathit{orange}(i) \mid i \in [1..n]\}$.

On the root of the tree, for every $J \in [1..n]$, put $J\text{-}\mathit{violet}(1)$ if it is accepting, and $J\text{-}\mathit{orange}(1)$ otherwise. Given the current slice, put tokens on the next slice as follows. For a son of a $J\text{-}\mathit{violet}(i)$ token, put a $J\text{-}\mathit{violet}(i)$ token. For a son of a $J\text{-}\mathit{orange}(i)$ token, put $J\text{-}\mathit{violet}(i)$ if it is accepting and $J\text{-}\mathit{orange}(i)$ otherwise. If for some J the number of used $J\text{-}\mathit{token}()$'s on the current slice is less than J, buzz the J-buzzer. Then, for all $J' \geq J$, put on all nodes of the slice a new $J'\text{-}\mathit{token}()$, with i's increasing from left to right, and with $J'\text{-}\mathit{orange}(i)$ placed on a non-accepting node and $J'\text{-}\mathit{violet}(i)$ on an accepting node. If for some i and J, all $J\text{-}\mathit{token}(i)$ are violet, ring the J-bell. The for every non-accepting node with $J\text{-}\mathit{violet}(i)$ on it, for every $J' \geq J$, if the current $J'\text{-}\mathit{token}()$ on it is $J'\text{-}\mathit{violet}(i')$, replace it with a $J'\text{-}\mathit{orange}(i')$ token.

[6] The construction here incorporates the modification required to get the complexity of $n^{O(n)}$.

On Reachability Analysis of Pushdown Systems with Transductions: Application to Boolean Programs with Call-by-Reference*

Fu Song, Weikai Miao, Geguang Pu, and Min Zhang†

Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China
{fsong,wkmiao,ggpu,mzhang}@sei.ecnu.edu.cn

Abstract

Pushdown systems with transductions (TrPDSs) are an extension of pushdown systems (PDSs) by associating each transition rule with a transduction, which allows to inspect and modify the stack content at each step of a transition rule. It was shown by Uezato and Minamide that TrPDSs can model PDSs with checkpoint and discrete-timed PDSs. Moreover, TrPDSs can be simulated by PDSs and the predecessor configurations $pre^*(C)$ of a regular set C of configurations can be computed by a saturation procedure when the closure of the transductions in TrPDSs is finite. In this work, we comprehensively investigate the reachability problem of finite TrPDSs. We propose a novel saturation procedure to compute $pre^*(C)$ for finite TrPDSs. Also, we introduce a saturation procedure to compute the successor configurations $post^*(C)$ of a regular set C of configurations for finite TrPDSs. From these two saturation procedures, we present two efficient implementation algorithms to compute $pre^*(C)$ and $post^*(C)$. Finally, we show how the presence of transductions enables the modeling of Boolean programs with call-by-reference parameter passing. The TrPDS model has finite closure of transductions which results in model-checking approach for Boolean programs with call-by-reference parameter passing against safety properties.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Verification, Reachability problem, Pushdown system with transductions, Boolean programs with call-by-reference

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.383

1 Introduction

A pushdown system (PDS) consists of a finite set of states and a finite stack alphabet, where stack can store context. PDS is one of the most widely used models for sequential programs with recursion [13, 25, 28]. Thanks to the efficient algorithms for the reachability problem of PDSs [6, 13], several software model-checking tools such as Moped [16], PDSolver [17], PuMoC [26] are implemented based the theory of PDSs for C/C++, Java and Boolean program verification. Several extensions of PDSs are proposed to model more complex behaviors.

* This work was partially supported by Shanghai Pujiang Program (No. 14PJ1403200), NSFC Projects (Nos. 61402179, 91418203,61202105,61361136002,91118007), Shanghai ChenGuang Program (No. 13CG21) and China HGJ Project (No. 2014ZX01038-101-001).

† Corresponding author

Various classes of pushdown automata such as multi-stack PDSs [9], pushdown networks [7, 18, 27, 29] and well-structure PDSs [10] have been proposed for modeling concurrent (thread-creation) programs with recursion. In order to model timed (resp. probabilistic) behavior, models that combine timed (resp. probabilistic) automata and PDSs are investigated in the literature, e.g., discrete/dense-timed pushdown systems [1, 2], nested timed automata [20] (resp. probabilistic PDSs [14, 8]). For dataflow analysis purpose, weighted PDSs [23] and extended weighted PDSs [19] are proposed, where transitions are associated with values from semirings. For stack manipulation of PDSs, Esparza et al. introduced PDSs with checkpoint [15] that can check the full stack content against a regular language (recognized by a finite state automaton) over the stack alphabet. This model is called conditional PDSs in [21] and transformable PDSs in [30]. Uezato and Minamide extended PDSs with transductions (TrPDSs) which associate each transition with a transduction. The associated transductions can check the stack content and modify the whole stack content. TrPDSs are a generalization of PDSs with checkpoint and discrete-timed PDSs. In general, TrPDSs are Turing complete. To achieve decidability result, Uezato and Minamide considered *finite* TrPDSs which restrict the closure of transductions appearing in the transitions of a TrPDS to be finite. They showed that a finite TrPDS can be simulated by a PDS. Therefore, the reachability problem of finite TrPDSs is decidable. Moreover, the saturation procedure that calculates the set $pre^*(C)$ of predecessor configurations for a given regular set of configurations C can be directly extended from PDSs to finite TrPDSs.

In this work, we follow the direction of [30] and make a comprehensive study of the reachability problem of TrPDSs. The main contributions of this paper can be summarized as follows:

- A novel saturation procedure is proposed that computes the set $pre^*(C)$ of predecessor configurations for a given regular set of configurations C of TrPDSs (cf. Section 3.1). This saturation procedure avoids *pseudo formal power series semiring* that was introduced in [30] to compute $pre^*(C)$.
- A saturation procedure is introduced to compute the set $post^*(C)$ of successor configurations for a given regular set of configurations C of TrPDSs (cf. Section 4.1). TrPDSs can be simulated by PDSs as shown in [30]. Therefore, $post^*(C)$ could be computed by applying the saturation procedure of PDSs [13]. Our saturation procedure directly computes a kind of finite state automaton that exactly recognizes $post^*(C)$. We believe our direct approach is more convenient for studying optimal algorithm or BDD-based symbolic techniques.
- Efficient implementation algorithms of the saturation procedures for computing $pre^*(C)$ and $post^*(C)$ are presented (cf. Section 3.2 and Section 4.2). We show that the computations of both $pre^*(C)$ and $post^*(C)$ are fixed-parameter tractable with the fixed-parameter of transductions.
- We show that TrPDSs are powerful enough to model Boolean programs with call-by-reference parameter passing. Boolean programs in the literature [3] only consider call-by-value parameter passing which can be modeled by PDSs. Using our approach, safety properties of Boolean programs with mixed call-by-reference and call-by-value parameter passing can be directly verified (cf. Section 5).

Section 2 presents basic definitions. Section 3 (resp. Section 4) introduce the saturation procedure and its efficient implementation algorithm for computing $pre^*(C)$ (resp. $post^*(C)$). In Section 5, we present a potential application of TrPDSs for modeling and verifying Boolean programs with call-by-reference parameter passing. Section 6 discusses related work. Section 7 concludes and discusses future work. Due to space limitation, proofs and details of Examples (9 and 13) are omitted and will appear in the journal version of this paper.

2 Preliminaries

2.1 Finite-State Transducers and Transduction

▶ **Definition 1.** A *finite-state transducer* (FST) $\mathbf{T}$ is a tuple $(Q, \Gamma, \delta, I, F)$, where Q is a finite set of states, Γ is a finite alphabet, $\delta \subseteq Q \times \Gamma^* \times \Gamma^* \times Q$ is a finite set of transition rules, $I \subseteq Q$ (resp. $F \subseteq Q$) is a finite set of initial (resp. final) states. The transducer is *letter-to-letter* if $\delta \subseteq Q \times \Gamma \times \Gamma \times Q$.

We will write $q \xrightarrow{\omega_1/\omega_2} q'$ if $(q, \omega_1, \omega_2, q') \in \delta$. Let $\longrightarrow^*$ be the smallest relation such that $q \xrightarrow{\epsilon/\epsilon}{}^* q$ for every $q \in Q$; if $q \xrightarrow{\omega_1/\omega_2}{}^* q'$ and $q' \xrightarrow{\omega_3/\omega_4} q''$, then $q \xrightarrow{\omega_1\omega_3/\omega_2\omega_4}{}^* q''$. A FST $\mathbf{T}$ transduces a string $\omega_1 \in \Gamma^*$ into a string $\omega_2 \in \Gamma^*$ if there exist states $q_0 \in I$ and $q_f \in F$ such that $q_0 \xrightarrow{\omega_1/\omega_2}{}^* q_f$. The language $L(\mathbf{T})$ of a FST $\mathbf{T}$ is the set of pairs (ω_1, ω_2) such that $\mathbf{T}$ can transduce ω_1 into ω_2.

A *transduction* $\tau \subseteq \Gamma^* \times \Gamma^*$ is a relation over Γ^*. A transduction τ is *rational* (regular) and *length-preserving* if there is a letter-to-letter transducer $\mathbf{T}$ such that $\tau = L(\mathbf{T})$. Let τ_{id} denote the *identity transduction*, i.e., $\tau_{id} = \{(\omega, \omega) \mid \forall \omega \in \Gamma^*\}$. In the rest of this paper, we assume that transductions (resp. transducers) are length-preserving rational (resp. letter-to-letter) unless stated explicitly, and we do not differentiate the terms transduction and transducer. Given a transduction τ, let $\tau(\omega) = \{\omega' \mid (\omega, \omega') \in \tau\}$ for $\omega \in \Gamma^*$.

The *composition* $\circ$ of two transductions τ_1, τ_2 is defined as

$$\tau_1 \circ \tau_2 = \{(\omega_1, \omega_3) \mid \exists \omega_2 \in \Gamma^*, (\omega_1, \omega_2) \in \tau_1 \text{ and } (\omega_2, \omega_3) \in \tau_2\}.$$

▶ **Proposition 2.** *For every transduction τ, $\tau \circ \tau_{id} = \tau = \tau_{id} \circ \tau$.*

The *left quotient* $\lceil \cdot, \cdot \rfloor^{-1}$ over transductions is defined as follows: $\forall \omega_1, \omega_2 \in \Gamma^*$ with $|\omega_1| = |\omega_2|$, $\lceil \omega_1, \omega_2 \rfloor^{-1}\tau = \{(\omega, \omega') \mid (\omega_1\omega, \omega_2\omega') \in \tau\}$.

▶ **Proposition 3.** *[30] For every $\omega_1, \omega_2 \in \Gamma^n$, for every transduction τ_1, τ_2,*

$$\lceil \omega_1, \omega_2 \rfloor^{-1}(\tau_1 \circ \tau_2) = \bigcup_{\omega_3 \in \Gamma^{|\omega_1|}} \left((\lceil \omega_1, \omega_3 \rfloor^{-1}\tau_1) \circ (\lceil \omega_3, \omega_2 \rfloor^{-1}\tau_2)\right).$$

Let $\mathcal{T}$ be a set of transductions, the closure $\langle \mathcal{T} \rangle^\cup$ of $\mathcal{T}$ over the composition $\circ$, left quotient $\lceil \cdot, \cdot \rfloor^{-1}$ and union $\cup$ is defined as follows:

- $\mathcal{T} \subseteq \langle \mathcal{T} \rangle^\cup$, $\emptyset \in \langle \mathcal{T} \rangle^\cup$ and $\tau_{id} \in \langle \mathcal{T} \rangle^\cup$;
- if $\tau_1, \tau_2 \in \langle \mathcal{T} \rangle^\cup$, then $\tau_1 \circ \tau_2 \in \langle \mathcal{T} \rangle^\cup$ and $\tau_1 \cup \tau_2 \in \langle \mathcal{T} \rangle^\cup$;
- if $\tau \in \langle \mathcal{T} \rangle^\cup$, then $\lceil \gamma, \gamma' \rfloor^{-1}\tau \in \langle \mathcal{T} \rangle^\cup$ for all $\gamma, \gamma' \in \Gamma$.

Similarly, let $\langle \mathcal{T} \rangle$ denote the closure of $\mathcal{T}$ over the composition $\circ$ and left quotient $\lceil \cdot, \cdot \rfloor^{-1}$.

▶ **Proposition 4.** *(a) The set $\langle \mathcal{T} \rangle$ is finite iff the set $\langle \mathcal{T} \rangle^\cup$ is finite.*

(b) The set $\langle \mathcal{T} \rangle^\cup$ is the semigroup generated by $(\langle \mathcal{T} \rangle, \cup)$, that is, $\forall \tau \in \langle \mathcal{T} \rangle^\cup$, $\exists \tau_1, ..., \tau_m \in \langle \mathcal{T} \rangle$ for $m \geq 1$ such that $\tau = \bigcup_{i=1}^m \tau_i$.

2.2 Pushdown Systems with Transductions

Pushdown systems with transductions (TrPDSs) [30] are an extension of pushdown systems by associating each transition with a transduction which modifies the stack content by applying the transduction. This extension allows TrPDSs to model sequential programs that manipulate the stack content rather than only the top of the stack.

▶ **Definition 5.** A *pushdown system with transductions* (TrPDS) $\mathcal{P}$ is a tuple $(P, \Gamma, \mathcal{T}, \Delta)$, where P is a finite set of control states, Γ is a finite alphabet, $\mathcal{T}$ is a finite set of transductions over Γ^*, $\Delta \subseteq P \times \Gamma \times \mathcal{T} \times P \times \Gamma^*$ is a finite set of transition rules. A TrPDS is a *pushdown system* (PDS) if $\mathcal{T} = \{\tau_{id}\}$.

We will write $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \omega\rangle$ instead, if $(p, \gamma, \tau, p', \omega) \in \Delta$. A *configuration* of a TrPDS $\mathcal{P}$ is a pair $\langle p, \omega\rangle \in P \times \Gamma^*$ where p is the control state and ω is the stack content. Let $\mathcal{C}_\mathcal{P}$ denote the set of all the configurations $P \times \Gamma^*$ of the TrPDS $\mathcal{P}$. The TrPDS $\mathcal{P}$ is called *finite* if the set $\langle\mathcal{T}\rangle$ (i.e.,$\langle\mathcal{T}\rangle^\cup$) is finite. If $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \omega\rangle$, then for every $\omega' \in \Gamma^*$, the configuration $\langle p, \gamma\omega'\rangle$ is an *immediate predecessor* of the configuration $\langle p', \omega u\rangle$ for every $u \in \tau(\omega')$, and the configuration $\langle p', \omega u\rangle$ for every $u \in \tau(\omega')$ is an *immediate successor* of the configuration $\langle p, \gamma\omega'\rangle$. Let $\Longrightarrow \subseteq \mathcal{C}_\mathcal{P} \times \mathcal{C}_\mathcal{P}$ be the *immediate successor relation*, i.e., for every $\omega', u \in \Gamma^*$, $\langle p, \gamma\omega'\rangle \Longrightarrow \langle p', \omega u\rangle$ if $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \omega\rangle$ and $u \in \tau(\omega')$. A *run* of $\mathcal{P}$ is a sequence of configurations $c_1 c_2 \cdots$ such that for every $i \geq 1$, c_{i+1} is an immediate successor of c_i.

Let $\Longrightarrow^n \subseteq \mathcal{C}_\mathcal{P} \times \mathcal{C}_\mathcal{P}$ be the successor relation over configurations of $\mathcal{P}$ defined as follows:

- $c \Longrightarrow^0 c$ for every $c \in \mathcal{C}_\mathcal{P}$;
- $c \Longrightarrow^n c''$ if there exists $c' \in \mathcal{C}_\mathcal{P}$ such that $c \Longrightarrow c'$ and $c' \Longrightarrow^{n-1} c''$.

Let $\Longrightarrow^* \subseteq \mathcal{C}_\mathcal{P} \times \mathcal{C}_\mathcal{P}$ denote the reflexive transitive closure of the immediate successor relation $\Longrightarrow$, i.e., $\Longrightarrow^* = \bigcup_{i\geq 0} \Longrightarrow^i$. Let $\Longrightarrow^+ \subseteq \mathcal{C}_\mathcal{P} \times \mathcal{C}_\mathcal{P}$ denote the transitive closure of the immediate successor relation $\Longrightarrow$, i.e., $\Longrightarrow^+ = \bigcup_{i\geq 1} \Longrightarrow^i$.

The *predecessor function* $pre : 2^{\mathcal{C}_\mathcal{P}} \longrightarrow 2^{\mathcal{C}_\mathcal{P}}$ of $\mathcal{P}$ is defined as follows: $pre(C) = \{c \in \mathcal{C}_\mathcal{P} \mid \exists c' \in C : c \Longrightarrow c'\}$. The reflexive transitive closure of pre is denoted by pre^*. Formally, $pre^*(C) = \{c \in \mathcal{C}_\mathcal{P} \mid \exists c' \in C : c \Longrightarrow^* c'\}$. Similarly, the *successor function* $post : 2^{\mathcal{C}_\mathcal{P}} \longrightarrow 2^{\mathcal{C}_\mathcal{P}}$ of $\mathcal{P}$ is defined as follows: $post(C) = \{c \in \mathcal{C}_\mathcal{P} \mid \exists c' \in C : c' \Longrightarrow c\}$. The reflexive transitive closure $post^*$ of $post$ is defined as $post^*(C) = \{c \in \mathcal{C}_\mathcal{P} \mid \exists c' \in C : c' \Longrightarrow^* c\}$.

2.3 Finite Automata with Transductions

To finitely represent regular sets of configurations of TrPDSs, we use finite automata with transductions.

▶ **Definition 6** ([30])**.** Given a TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$, a *finite automaton with transduction and ϵ-moves* (ϵ-TrNFA) $\mathbf{A}$ is a tuple $(S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$, where S is a finite set of states, $\Lambda \subseteq S \times (\Gamma \cup \{\epsilon\}) \times \langle\mathcal{T}\rangle^\cup \times S$ is a finite set of transition rules, $S_0, S_f \subseteq S$ are initial and final states. An ϵ-TrNFA $\mathbf{A}$ is TrNFA if $\Lambda \subseteq S \times \Gamma \times \langle\mathcal{T}\rangle^\cup \times S$.

We write $s \overset{\gamma|\tau}{\longmapsto} s'$ if $(s, \gamma, \tau, s') \in \Lambda$ (note that $\gamma \in \Gamma \cup \{\epsilon\}$). Let $\longmapsto^n : S \times \Gamma^* \times \langle\mathcal{T}\rangle^\cup \times S$ be a relation over states of $\mathbf{A}$ defined as follows:

- $s \overset{\epsilon|\tau_{id}}{\longmapsto} {}^0 s$, for every $s \in S$;
- $s \overset{\gamma\gamma_1\cdots\gamma_n|\left(\lceil\gamma_1\cdots\gamma_n,\gamma'_1\cdots\gamma'_n\rfloor^{-1}\tau\right)\circ\tau_1}{\longmapsto} {}^{n+1}\, s_2$ for all $\gamma_1, ..., \gamma_n \in \Gamma$, if $\exists s_1 \in S$ such that $s \overset{\gamma|\tau}{\longmapsto} s_1$ and $s_1 \overset{\gamma'_1\cdots\gamma'_n|\tau_1}{\longmapsto} {}^n s_2$.

TrNFA is the standard finite state automata if $\mathcal{T} = \{\tau_{id}\}$, a.k.a. $\mathcal{P}$-automata [6] if S corresponds to the control states of $\mathcal{P}$.

Let $\longmapsto^* = \bigcup_{i\geq 0} \longmapsto^i$. A configuration $\langle p, \omega\rangle \in P \times \Gamma^*$ of a TrPDS $\mathcal{P}$ is *recognized* (accepted) by an ϵ-TrNFA $\mathbf{A}$ iff $s \overset{\omega|\tau}{\longmapsto} {}^* s'$ such that $s = p \in S_0$, $s' \in S_f$ and $(\epsilon, \epsilon) \in \tau$. A set C of configurations is *rational* (regular) if there exists an ϵ-TrPDS $\mathbf{A}$ such that $L(\mathbf{A}) = C$. From now on, we omit the paths of the form $s_1 \overset{\omega|\tau}{\longmapsto} {}^n s_2$ such that $\tau = \emptyset$, as these paths do not allow the ϵ-TrNFA to accept a configuration.

▶ **Theorem 7** ([30]). *Given a finite TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ and a rational set of configurations C of $\mathcal{P}$, both $post^*(C)$ and $pre^*(C)$ are rational and effectively computable.*

3 Computing pre^*

In this section, we present a saturation procedure to compute pre^* which is different from the way presented in [30] and an efficient implementation for pre^*.

3.1 Saturation Procedure for Computing pre^*

Given a finite TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ and a TrNFA $\mathbf{A} = (S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$ that recognizes a rational set of configurations of $\mathcal{P}$, w.l.o.g., we assume $P = S_0$ and there is no transition rule in $\mathbf{A}$ leading to an initial state and $\mathbf{A}$ uses only the identity transduction τ_{id} (cf. Section 6.4 of [30]), we construct a new TrNFA $\mathbf{A}^{pre^*} = (S, \Gamma, \Lambda^{pre^*}, \mathcal{T}, S_0, S_f)$ such that $\mathbf{A}^{pre^*}$ recognizes $pre^*(L(\mathbf{A}))$, i.e., $L(\mathbf{A}^{pre^*}) = pre^*(L(\mathbf{A}))$. The construction of $\mathbf{A}^{pre^*}$ is based on a kind of saturation procedure which extends the saturation procedure to compute pre^* of PDSs [6]. Initially, $\mathbf{A}^{pre^*} = \mathbf{A}$, then we iteratively apply the following saturation procedure until no new transition rule can be added into $\mathbf{A}^{pre^*}$.

> If $\langle p, \gamma\rangle \overset{\tau_1}{\hookrightarrow} \langle q, \omega\rangle \in \Delta$ and $q \overset{\omega|\tau_2}{\longmapsto} {}^* q'$ in the current automaton $\mathbf{A}^{pre^*}$, add a transition rule $p \overset{\gamma|\tau_1\circ\tau_2}{\longmapsto} q'$ into Λ^{pre^*}.

Since the set of states of $\mathbf{A}^{pre^*}$ and the set $\langle\mathcal{T}\rangle$ of transductions are finite, the set of transition rules in $\mathbf{A}^{pre^*}$ is finite. Thus, the above saturation will eventually reach a fixpoint. Intuitively, if there is a transition rule $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle q, \gamma_1^1 \cdots \gamma_n^1\rangle \in \Delta$, then $\langle p, \gamma\gamma_{n+1}\cdots\gamma_m\rangle \Longrightarrow \langle q, \gamma_1^1\cdots\gamma_n^1\gamma_{n+1}^1\cdots\gamma_m^1\rangle$ for all $\gamma_{n+1}^1\cdots\gamma_m^1 \in \tau(\gamma_{n+1}\cdots\gamma_m)$. If the automaton $\mathbf{A}^{pre^*}$ recognizes the configuration $\langle q, \gamma_1^1\cdots\gamma_m^1\rangle$ by a path $q \overset{\gamma_1^1\cdots\gamma_m^1|\tau'}{\longmapsto}{}^m g$ for some final state g of $\mathbf{A}^{pre^*}$ and $(\epsilon, \epsilon) \in \tau'$, then, we can decompose this path to $q \overset{\gamma_1^1\cdots\gamma_n^1|\tau''}{\longmapsto}{}^n q'$ and $q' \overset{\gamma_{n+1}^{n+1}\cdots\gamma_m^{n+1}|\tau'''}{\longmapsto}{}^{m-n} g$ such that if $\tau' = (\lceil\gamma_2^1\cdots\gamma_m^1, \gamma_2^2\cdots\gamma_m^2\rfloor^{-1}\tau_1)\circ\cdots\circ(\lceil\gamma_m^{m-1}, \gamma_m^m\rfloor^{-1}\tau_{m-1})\circ\tau_m$, then

- $\tau'' = (\lceil\gamma_2^1\cdots\gamma_n^1, \gamma_2^2\cdots\gamma_n^2\rfloor^{-1}\tau_1)\circ\cdots\circ(\lceil\gamma_n^{n-1}, \gamma_n^n\rfloor^{-1}\tau_{n-1})\circ\tau_n$,
- $\tau''' = (\lceil\gamma_{n+2}^{n+1}\cdots\gamma_m^{n+1}, \gamma_{n+2}^{n+2}\cdots\gamma_m^{n+2}\rfloor^{-1}\tau_{n+1})\circ\cdots\circ(\lceil\gamma_m^{m-1}, \gamma_m^m\rfloor^{-1}\tau_{m-1})\circ\tau_m$.

Moreover, since $(\epsilon, \epsilon) \in \tau'$, we get that $(\gamma_{n+1}^1\cdots\gamma_m^1, \gamma_{n+1}^{n+1}\cdots\gamma_m^{n+1}) \in \tau''$ and $(\epsilon, \epsilon) \in \tau'''$. Applying the saturation procedure, the transition rule $p \overset{\gamma|\tau\circ\tau''}{\longmapsto} q'$ is added into $\mathbf{A}^{pre^*}$. Therefore, $\mathbf{A}^{pre^*}$ recognizes the configuration $\langle p, \gamma\gamma_{n+1}\cdots\gamma_m\rangle$ by composing $p \overset{\gamma|\tau\circ\tau''}{\longmapsto} q'$ and $q' \overset{\gamma_{n+1}^{n+1}\cdots\gamma_m^{n+1}|\tau'''}{\longmapsto}{}^{m-n} g$ into $p \overset{\gamma\gamma_{n+1}\cdots\gamma_m|\left(\lceil\gamma_{n+1}\cdots\gamma_m, \gamma_{n+1}^{n+1}\cdots\gamma_m^{n+1}\rfloor^{-1}(\tau\circ\tau'')\right)\circ\tau'''}{\longmapsto}{}^{m-n+1} g$ (note that $(\gamma_{n+1}\cdots\gamma_m, \gamma_{n+1}^{n+1}\cdots\gamma_m^{n+1}) \in \tau\circ\tau''$ implies that $(\epsilon, \epsilon) \in \left(\lceil\gamma_{n+1}\cdots\gamma_m, \gamma_{n+1}^{n+1}\cdots\gamma_m^{n+1}\rfloor^{-1}(\tau\circ\tau'')\right)\circ\tau'''$. Thus, we get the following theorem.

▶ **Theorem 8.** *Given a finite TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ and a rational set of configurations C of $\mathcal{P}$ recognized a TrNFA $\boldsymbol{A} = (S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$, we can construct a TrNFA $\boldsymbol{A}'$ such that $L(\boldsymbol{A}') = pre^*(C)$ in time $\mathbf{O}(|\Delta|^3\cdot|S|^3\cdot|\Lambda|\cdot f(|\mathcal{T}|))$ and in space $\mathbf{O}(|\Delta|\cdot|S|\cdot|\langle\mathcal{T}\rangle|)$, where f is some computable function.*

We notice that the number $|\Lambda^{pre^*}|$ of transition rules of $\mathbf{A}^{pre^*}$ is at most $\mathbf{O}(|\Lambda| + |\Delta|\cdot|S|\cdot|\langle\mathcal{T}\rangle|)$. For each transition rule $\langle p, \gamma\rangle \overset{\tau_1}{\hookrightarrow} \langle q, \gamma_1\gamma_2\rangle \in \Delta$, paths $q \overset{\gamma_1\gamma_2|\tau_2}{\longmapsto}{}^* g$ can be computed in time $\mathbf{O}(f(|\mathcal{T}|)\cdot(|S| + |P|)\cdot|\Lambda^{pre^*}|)$ for some computable function f. Thus, we get that the saturation procedure executes at most in time $\mathbf{O}(|\Delta|^3\cdot|S|^3\cdot|\Lambda|\cdot f(|\mathcal{T}|))$. Memory is needed for storing the new transition rules which is bounded by $\mathbf{O}(|\Delta|\cdot|S|\cdot|\langle\mathcal{T}\rangle|)$.

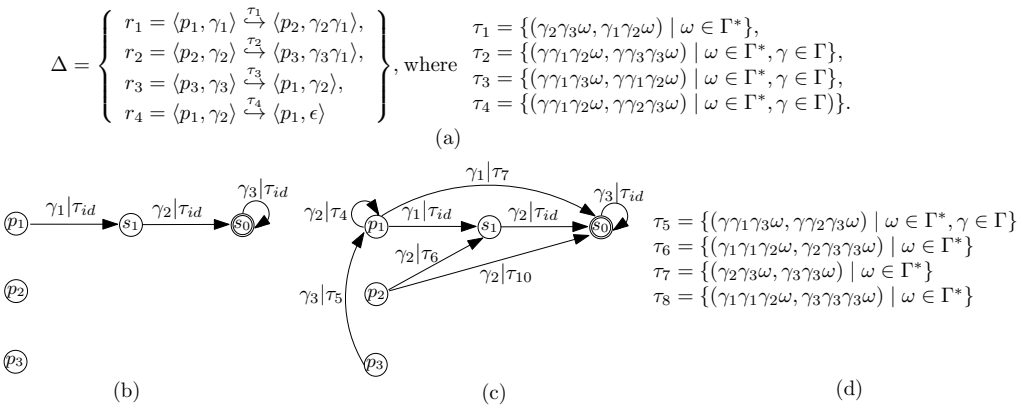


Figure 1 (a) The set of transition rules Δ, (b) the TrNFA $\mathbf{A}$, (c) the TrNFA $\mathbf{A}^{pre^*}$ and (d) consists of related transductions.

▶ Remark. In [30], the authors introduce TrNFA and present a saturation procedure to compute pre^* without its complexity. They define the relation $\longmapsto^*$ by introducing a *pseudo formal power series semiring* to solve the associativity problem of the composition of transitions of TrNFAs. Their saturation procedure is proceeded based on this semiring. Our approach is proceeded based on TrNFAs and we show that this problem is fixed-parameter tractable (FPT). We believe our direct approach is more convenient for studying optimal algorithm or BDD-based symbolic techniques.

▶ **Example 9.** Consider the TrPDS with control states $\{p_1, p_2, p_3\}$ and Δ as shown in Figure 1(a). Let $\mathbf{A}$ be the TrNFA as shown in Figure 1(b). The result of applying the saturation procedure is shown in Figure 1(c). .

3.2 An Efficient Algorithm for Computing pre^*

In this section, we present an efficient implementation of the saturation procedure given in Section 3.1. W.l.o.g., we suppose in this section that for every TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$, $|\omega| \leq 2$ for every transition rule $\langle p, \gamma \rangle \overset{\tau}{\hookrightarrow} \langle q, \omega \rangle \in \Delta$. Let Δ_i denote $\{\langle p, \gamma \rangle \overset{\tau}{\hookrightarrow} \langle g, \omega \rangle \in \Delta \mid |\omega| = i\}$, for every $i \in \{0, 1, 2\}$.

Algorithm 1 computes the transition rules of $\mathbf{A}^{pre^*}$ by implementing the saturation procedure from Section 3.1. The basic idea follows from the efficient algorithm for computing pre^* of PDSs [13] which avoids unnecessary operations. Intuitively, for the transition rules of the form $\langle p, \gamma \rangle \overset{\tau}{\hookrightarrow} \langle p', \epsilon \rangle$ or $\langle p, \gamma_1 \rangle \overset{\tau'}{\hookrightarrow} \langle q, \gamma \rangle$ in Δ, the algorithm proceeds exactly the same as the saturation procedure given in Section 3.1. Whenever $\mathcal{P}$ has a transition rule in the form of $\langle p, \gamma_1 \rangle \overset{\tau'}{\hookrightarrow} \langle q, \gamma\gamma_2 \rangle$, we look out for every $q', q'' \in S$ and $\gamma_2' \in \Gamma$, the pairs of transition rules $q \overset{\gamma|\tau}{\longmapsto} q'$ and $q' \overset{\gamma_2'|\tau_2}{\longmapsto} q''$ such that $\lceil \gamma_2, \gamma_2' \rfloor^{-1}\tau \neq \emptyset$, so that we can add the transition rule $p \overset{\gamma_1|\tau' \circ (\lceil \gamma_2, \gamma_2' \rfloor^{-1}\tau) \circ \tau_2}{\longmapsto} q''$. However, the order of such transitions added into the automaton $\mathbf{A}^{pre^*}$ can be arbitrary. Whenever a transition rule like $q' \overset{\gamma_2'|\tau_2}{\longmapsto} q''$ is found, we have to check whether $q \overset{\gamma|\tau}{\longmapsto} q'$ exists or not. Then, this checking may be negative, and wastes time to no avail. However, once a transition rule $q \overset{\gamma|\tau}{\longmapsto} q'$ is seen, we know that all subsequent transitions like $q' \overset{\gamma_2'|\tau_2}{\longmapsto} q''$ must lead to the addition of the transition rule $p \overset{\gamma_1|\tau' \circ (\lceil \gamma_2, \gamma_2' \rfloor^{-1}\tau) \circ \tau_2}{\longmapsto} q'$. That's why we introduce a new transition rule $\langle p, \gamma_1 \rangle \overset{\tau' \circ (\lceil \gamma_2, \gamma_2' \rfloor^{-1}\tau)}{\hookrightarrow} \langle q', \gamma_2' \rangle$ into Δ' which allows

Input : A finite TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ and a TrNFA $\mathbf{A} = (S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$ such that $\mathbf{A}$ uses only τ_{id} and Λ has no transition rule leading to a state in P

Output : The set of transition rules of $\mathbf{A}^{pre^*}$

1 $\Lambda' := \Lambda$; $trans := \Lambda$; $\Delta' := \emptyset$;
2 **foreach** $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \epsilon\rangle \in \Delta$ **do** **Update**$(p \overset{\gamma|\tau}{\longmapsto} p')$;
3 ;
4 **while** $trans \neq \emptyset$ **do**
5 remove $t = q \overset{\gamma|\tau}{\longmapsto} q'$ from $trans$;
6 **foreach** $\langle p, \gamma_1\rangle \overset{\tau'}{\hookrightarrow} \langle q, \gamma\rangle \in \Delta \cup \Delta'$ **do** **Update**$(p \overset{\gamma_1|\tau' \circ \tau}{\longmapsto} q')$;
7 ;
8 **foreach** $\langle p, \gamma_1\rangle \overset{\tau'}{\hookrightarrow} \langle q, \gamma\gamma_2\rangle \in \Delta$ *and* $\gamma_2' \in \Gamma$ **do**
9 **if** $\lceil \gamma_2, \gamma_2' \rfloor^{-1} \tau \neq \emptyset$ **then**
10 $\Delta' := \Delta' \cup \{\langle p, \gamma_1\rangle \xrightarrow{\tau' \circ (\lceil \gamma_2, \gamma_2' \rfloor^{-1} \tau)} \langle q', \gamma_2'\rangle\}$;
11 **foreach** $q' \overset{\gamma_2'|\tau_2}{\longmapsto} q'' \in \Lambda'$ **do**
12 **Update**$(p \xrightarrow{\gamma_1|\tau' \circ (\lceil \gamma_2, \gamma_2' \rfloor^{-1} \tau) \circ \tau_2} q'')$;
13 **return** Λ';
14 **Procedure Update**$(q \overset{\gamma|\tau}{\longmapsto} q')$
15 **if** $t = q \overset{\gamma|\tau'}{\longmapsto} q' \in \Lambda'$ **then**
16 $t' := q \overset{\gamma|\tau' \cup \tau}{\longmapsto} q'$;
17 $\Lambda' := \Lambda' \cup \{t'\} \setminus \{t\}$;
18 **if** $\tau' \neq \tau' \cup \tau$ **then** $trans := trans \cup \{t'\} \setminus \{t\}$;
19 ;
20 **else if** $\tau \neq \emptyset$ **then**
21 $\Lambda' := \Lambda' \cup \{q \overset{\gamma|\tau}{\longmapsto} q'\}$;
22 $tran := tran \cup \{q \overset{\gamma|\tau}{\longmapsto} q''\}$;

Algorithm 1. An efficient algorithm for computing pre^*.

us to add the transition rule $p \xrightarrow{\gamma_1|\tau' \circ (\lceil \gamma_2, \gamma_2' \rfloor^{-1} \tau) \circ \tau_2} q''$ once $q' \overset{\gamma_2'|\tau_2}{\longmapsto} q''$ occurs. Let us explain Algorithm 1 line by line as follows.

Line 1 initializes the algorithm by assigning Λ to Λ' and $trans$, $\emptyset$ to Δ'. Line 2 handles normal transition rules of the form $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \epsilon\rangle$, where new transitions $p \overset{\gamma|\tau}{\longmapsto} p'$ can be immediately added. Once a new transition rule is created, we call the procedure **Update** which will be explained later. Lines 3-10 iteratively removes a transition $t = q \overset{\gamma|\tau}{\longmapsto} q'$ from $trans$ until it is empty. The loop at Line 5 handles the case when q and γ match the right-hand side of transition rules in $\Delta \cup \Delta'$.

The procedure **Update** listed at Lines 12-19 is called whenever a new transition rule $q \overset{\gamma|\tau}{\longmapsto} q'$ is created. If Λ' contains a transition rule of the form $t = q \overset{\gamma|\tau'}{\longmapsto} q'$ for any τ', then, we remove t from Λ' and add a new transition rule $q \overset{\gamma|\tau' \cup \tau}{\longmapsto} q'$ into Λ' at Line 15. In other words, we update the transduction τ' by $\tau' \cup \tau$. Moreover, if $\tau' \cup \tau$ does not equal to τ', we remove t from $trans$ and add $q \overset{\gamma|\tau' \cup \tau}{\longmapsto} q'$ into $trans$ at Line 16 for later processing. Otherwise if Λ' has no transition rule like t, we add $q \overset{\gamma|\tau}{\longmapsto} q'$ into Λ' and $trans$.

▶ **Theorem 10.** *Given a finite TrPDS* $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ *and a TrNFA* $\boldsymbol{A} = (S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$, *we can compute a TrNFA* $\boldsymbol{A}^{pre^*}$ *in time* $\mathbf{O}(|S|^2 \cdot f(|\mathcal{T}|) \cdot |\Delta| \cdot |\Gamma|)$ *for some computable function* f *and in space* $\mathbf{O}(|S| \cdot |\Delta| \cdot |\langle\mathcal{T}\rangle| \cdot |\Gamma|)$ *such that* $L(\boldsymbol{A}^{pre^*}) = pre^*(L(\boldsymbol{A}))$.

4 Computing $post^*$

In this section, we present an approach to compute $post^*$ which is different from the way presented in [30]. In [30], $post^*$ is computed by transforming a *finite* TrPDS into an equivalent PDS and then computing $post^*$ of the resulting PDS. We will present a saturation procedure which directly computes $post^*$ similar as computing pre^* given in Section 3. Finally, we give an efficient algorithm implementing this saturation procedure.

4.1 Saturation Procedure for Computing $post^*$

Given a finite TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ and a TrNFA $\mathbf{A} = (S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$ that recognizes a rational set of configurations of $\mathcal{P}$, we can construct an ϵ-TrNFA $\mathbf{A}^{post^*} = (S^{post^*}, \Lambda^{post^*}, S_0, S_f)$ such that $\mathbf{A}^{post^*}$ recognizes $post^*(L(\mathbf{A}))$, i.e., $L(\mathbf{A}^{post^*}) = post^*(L(\mathbf{A}))$. W.l.o.g., we assume that $S_0 = P$ and there is no transition rule in $\mathbf{A}$ leading to an initial state and $\mathbf{A}$ uses only the identity transduction τ_{id}. The construction of $\mathbf{A}^{post^*}$ is similar than the construction of $\mathbf{A}^{pre^*}$ which is an extension of the saturation procedure for computing $post^*$ of PDSs [13].

Given a transduction τ, let $\overline{\tau}$ denote the inversion $\{(\omega_1, \omega_2) \mid (\omega_2, \omega_1) \in \tau\}$ of τ, let $\overline{T}$ denote $\bigcup_{\tau \in T} \overline{\tau}$ for a given set T of transductions.

▶ **Proposition 11.** $\overline{\langle\mathcal{T}\rangle} = \langle\overline{\mathcal{T}}\rangle$ *and* $\overline{\langle\mathcal{T}\rangle^\cup} = \langle\overline{\mathcal{T}}\rangle^\cup$.

Initially, $\mathbf{A}^{post^*} = \mathbf{A}$, then we iteratively apply the following saturation procedure until the automaton is *saturated* (i.e., no new transition rule can be added):

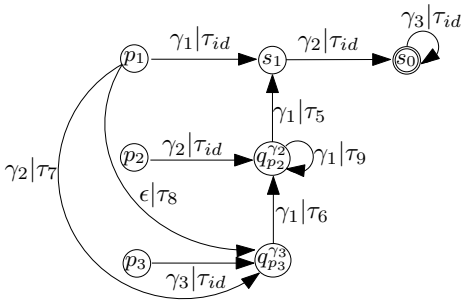
(i) If $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \epsilon\rangle \in \Delta$ and $p \overset{\gamma|\tau'}{\longmapsto}{}^* s$, add $p' \overset{\epsilon|\ \overline{\tau}\circ\tau'}{\longmapsto} s$ into Λ^{post^*};

(ii) If $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \gamma_1\rangle \in \Delta$ and $p \overset{\gamma|\tau'}{\longmapsto}{}^* s$, add $p' \overset{\gamma_1|\ \overline{\tau}\circ\tau'}{\longmapsto} s$ into Λ^{post^*};

(iii) If $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \gamma_1\gamma_2\rangle \in \Delta$ and $p \overset{\gamma|\tau'}{\longmapsto}{}^* s$, add $p' \overset{\gamma_1|\ \tau_{id}}{\longmapsto} q_{p'}^{\gamma_1}$ and $q_{p'}^{\gamma_1} \overset{\gamma_2|\ \overline{\tau}\circ\tau'}{\longmapsto} s$ into Λ^{post^*} and add a new state $q_{p'}^{\gamma_1}$ into S^{post^*}.

Intuitively, if there is a transition rule $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \epsilon\rangle \in \Delta$, then $\langle p, \gamma\omega\rangle$ is an immediate predecessor of the configuration $\langle p', \omega_1\rangle$ for every $\omega_1 \in \tau(\omega)$. Thus, if the automaton already accepts the configuration $\langle p, \gamma\omega\rangle$ by $p \overset{\gamma|\tau'}{\longmapsto}{}^* s$ and $s \overset{\omega_2|\tau_2}{\longmapsto}{}^* q_f$ for some final state q_f, where $\omega_2 \in \tau'(\omega)$ and $(\epsilon, \epsilon) \in \tau_2$. Then it also ought to accept $\langle p', \omega_1\rangle$, for every $\omega_1 \in \tau(\omega)$. Adding the transition rule $p' \overset{\epsilon|\ \overline{\tau}\circ\tau'}{\longmapsto} s$ allows the automaton to accept $\langle p', \omega_1\rangle$, for every $\omega_1 \in \tau(\omega)$, as $\omega_2 \in (\overline{\tau} \circ \tau')(\omega_1)$ for all $\omega_1 \in \tau(\omega)$.

If there is a transition rule $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \gamma_1\rangle \in \Delta$, then $\langle p, \gamma\omega\rangle$ is an immediate predecessor of the configuration $\langle p', \gamma_1\omega_1\rangle$ for every $\omega_1 \in \tau(\omega)$. Thus, if the automaton already accepts the configuration $\langle p, \gamma\omega\rangle$ by $p \overset{\gamma|\tau'}{\longmapsto}{}^* s$ and $s \overset{\omega_2|\tau_2}{\longmapsto}{}^* q_f$ for some final state q_f, where $\omega_2 \in \tau'(\omega)$ and $(\epsilon, \epsilon) \in \tau_2$. Then it also ought to accept $\langle p', \gamma_1\omega_1\rangle$, for every $\omega_1 \in \tau(\omega)$. Adding the transition rule $p' \overset{\gamma_1|\ \overline{\tau}\circ\tau'}{\longmapsto} s$ allows the automaton to accept $\langle p', \gamma_1\omega_1\rangle$, for every $\omega_1 \in \tau(\omega)$, as $\omega_2 \in (\overline{\tau} \circ \tau')(\omega_1)$ for all $\omega_1 \in \tau(\omega)$.



$\tau_5 = \{(\gamma_1\gamma_2\omega, \gamma_2\gamma_3\omega) \mid \omega \in \Gamma^*\}$
$\tau_6 = \{(\gamma\gamma_3\gamma_3\omega, \gamma\gamma_1\gamma_2\omega) \mid \gamma \in \Gamma, \omega \in \Gamma^*\}$
$\tau_7 = \{(\gamma\gamma_1\gamma_2\omega, \gamma\gamma_1\gamma_3\omega) \mid \gamma \in \Gamma, \omega \in \Gamma^*\}$
$\tau_8 = \{((\gamma\gamma_2\gamma_3\omega, \gamma\gamma_1\gamma_3\omega) \mid \gamma \in \Gamma, \omega \in \Gamma^*\}$
$\tau_9 = \{(\gamma_1\gamma_2\gamma_3\omega, \gamma_1\gamma_1\gamma_2\omega) \mid \omega \in \Gamma^*\}$

Figure 2 The resulting TrNFA $\mathbf{A}^{post^*}$.

If there is a transition rule $\langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p', \gamma_1\gamma_2\rangle \in \Delta$, then $\langle p, \gamma\omega\rangle$ is an immediate predecessor of the configuration $\langle p', \gamma_1\gamma_2\omega_1\rangle$ for every $\omega_1 \in \tau(\omega)$. Thus, if the automaton already accepts the configuration $\langle p, \gamma\omega\rangle$ by $p \overset{\gamma|\tau'}{\longmapsto}{}^* s$ and $s \overset{\omega_2|\tau_2}{\longmapsto}{}^* q_f$ for some final state q_f, where $\omega_2 \in \tau'(\omega)$ and $(\epsilon, \epsilon) \in \tau_2$. Then it also ought to accept $\langle p', \gamma_1\gamma_2\omega_1\rangle$, for every $\omega_1 \in \tau(\omega)$. Adding the transition rules $p' \overset{\gamma_1|\ \tau_{id}}{\longmapsto} q_{p'}^{\gamma_1}$ and $q_{p'}^{\gamma_1} \overset{\gamma_2|\ \overline{\tau}\circ\tau'}{\longmapsto} s$ allows the automaton to accept $\langle p', \gamma_1\gamma_2\omega_1\rangle$, for every $\omega_1 \in \tau(\omega)$, as $\omega_2 \in (\overline{\tau} \circ \tau')(\omega_1)$ for all $\omega_1 \in \tau(\omega)$.

▶ **Theorem 12.** *Given a finite TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ and a rational set of configurations C of $\mathcal{P}$ recognized a TrNFA $\boldsymbol{A} = (S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$, we can construct a TrNFA $\boldsymbol{A}'$ such that $L(\boldsymbol{A}') = post^*(C)$.*

▶ **Example 13.** Consider the TrPDS shown in Figure 1(a) and the TrNFA $\mathbf{A}$ shown in Figure 1(b). The result of applying the saturation procedure is shown in Figure 2.

4.2 An Efficient Algorithm for Computing $post^*$

In this section, we present an efficient implementation of the saturation procedure given in Section 4.1 which avoids unnecessary operations. Given a rational set of configurations of C represented by a TrNFA $\mathbf{A}$.

Algorithm 2 computes the transition rules of $\mathbf{A}^{post^*}$ by implementing the saturation procedure given in Section 4.1. The approach is similar to the solution for efficiently computing pre^*. We use *trans* to store the transition rules that we still need to examine. Lines 1-2 initialize the algorithm. Initially, Λ' is equal to Λ, while *trans* is equal to $\Lambda \cap P \times \Gamma \times \mathcal{T} \times S$, as transition rules starting from states outside of P do not need to be examined. The set S^{post^*} of states is equal to $S \cup \{q_{p_1}^{\gamma_1} \mid \langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p_1, \gamma_1\gamma_2\rangle \in \Delta\}$ as described in Section 4.1. The algorithm iteratively removes a transition $t = p \overset{\gamma|\tau}{\longmapsto} q$ from *trans* until it is empty. The loops at Line 6, Line 7 and Lines 8-10 handle the case when p and γ match the left-hand sides of transition rules in Δ. This is done similar as the saturation rules (i), (ii), and (iii), respectively. The loops at Lines 11-12 and Lines 13-14 handle ϵ-transition rules. In the saturation procedure given in Section 4.1, we have to compute paths $p \overset{\gamma|\tau'}{\longmapsto}{}^* s$ which may involve several ϵ-transitions. In Algorithm 2, we solve this problem by combining transition pairs of the form $p \overset{\epsilon|\tau_1}{\longmapsto} q_1$ and $q_1 \overset{\gamma|\tau_2}{\longmapsto} q$ into transition rules $p \overset{\gamma'|(\lceil\gamma',\gamma\rfloor^{-1}\tau_1)\circ\tau_2}{\longmapsto} q$ for $\gamma' \in \Gamma$ whenever such a pair is found.

Input : A finite TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ and a TrNFA $\mathbf{A} = (S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$ such that $\mathbf{A}$ uses only τ_{id}, Λ has no transition leading to a state in P and no ϵ-transition

Output : The TrNFA $\mathbf{A}^{post^*} = (S^{post^*}, \Lambda', S_0, S_f)$

$\Lambda' := \Lambda$; $trans := \Lambda \cap P \times \Gamma \times \mathcal{T} \times S$;
$S^{post^*} := S \cup \{q_{p_1}^{\gamma_1} \mid \langle p, \gamma\rangle \overset{\tau}{\hookrightarrow} \langle p_1, \gamma_1\gamma_2\rangle \in \Delta\}$;
while $trans \neq \emptyset$ **do**
  remove $t = p \overset{\gamma|\tau}{\longmapsto} q$ from $trans$;
  if $\gamma \neq \epsilon$ **then**
    foreach $\langle p, \gamma\rangle \overset{\tau_1}{\hookrightarrow} \langle p_1, \epsilon\rangle \in \Delta$ **do** **Update**$(p_1 \overset{\epsilon \mid \overline{\tau_1}\circ\tau}{\longmapsto} q)$;
    ;
    foreach $\langle p, \gamma\rangle \overset{\tau_1}{\hookrightarrow} \langle p_1, \gamma_1\rangle \in \Delta$ **do** **Update**$(p_1 \overset{\gamma_1 \mid \overline{\tau_1}\circ\tau}{\longmapsto} q)$;
    ;
    foreach $\langle p, \gamma\rangle \overset{\tau_1}{\hookrightarrow} \langle p_1, \gamma_1\gamma_2\rangle \in \Delta$ **do**
      Update$(p_1 \overset{\gamma_1 \mid \tau_{id}}{\longmapsto} q_{p_1}^{\gamma_1})$;
      Update$(q_{p_1}^{\gamma_1} \overset{\gamma_2 \mid \overline{\tau_1}\circ\tau}{\longmapsto} q)$;
      foreach $p_2 \overset{\epsilon \mid \tau_2}{\longmapsto} q_{p_1}^{\gamma_1} \in \Lambda',\ \gamma_2' \in \Gamma$ **do**
        Update$(p_2 \overset{\gamma_2' \mid (\lceil\gamma_2', \gamma_2\rfloor^{-1}\tau_2)\circ\overline{\tau_1}\circ\tau}{\longmapsto} q)$
  else **foreach** $q \overset{\gamma_1 \mid \tau_1}{\longmapsto} q' \in \Lambda',\ \gamma_1' \in \Gamma$ **do**
    Update$(p \overset{\gamma_1' \mid (\lceil\gamma_1', \gamma_1\rfloor^{-1}\tau)\circ\tau_1}{\longmapsto} q')$
  ;
return $\mathbf{A}^{post^*}$;

Algorithm 2. An efficient algorithm for computing $post^*$.

▶ **Theorem 14.** *Given a finite TrPDS $\mathcal{P} = (P, \Gamma, \mathcal{T}, \Delta)$ and a TrNFA $\boldsymbol{A} = (S, \Gamma, \Lambda, \mathcal{T}, S_0, S_f)$, we can compute a TrNFA $\boldsymbol{A}^{post^*}$ in $\mathbf{O}(|S|\cdot f(|\mathcal{T}|)\cdot|\Delta|^3\cdot|\Gamma|)$ time and space for some computable function f such that $L(\boldsymbol{A}^{post^*}) = post^*(L(\boldsymbol{A}))$.*

5 Application

In [30], Uezato and Minamide presented two potential applications of TrPDSs: checking reachability of conditional PDSs [22, 15] and discrete-timed PDSs [1] via pre^* or $post^*$ computing of TrPDSs. In this section, we will present another potential application of TrPDSs. We show how the presence of transductions enables the modeling of Boolean programs with call-by-reference parameter passing. Boolean programs in which all variables and parameters (call-by-value) have Boolean type are thought of as an abstract representation of C/C++ programs with recursion [3]. In their definition, Boolean programs contain procedures with call-by-value parameter passing rather than call-by-reference parameter passing. While call-by-reference parameter passing is a widely used programming paradigm in C/C++, Java, etc. Using TrPDSs, we can verify safety properties of Boolean programs with call-by-reference parameter passing.

5.1 Boolean Programs with Call-by-Reference Parameter Passing

A *Boolean program* BP is a tuple $(Proc, main, G)$, where $Proc$ is a finite set of procedures, $main \in Proc$ is the *initial* procedure, G is a finite set of global Boolean variables. Every procedure $r \in Proc$ is a tuple (N_r, E_r, L_r), where N_r is a finite set of control points with r_{entry} as the unique entry node, E_r is a finite set of edges, L_r is the finite set of local Boolean variables in r. W.l.o.g., we assume that $L_r \cap G = \emptyset$ for all $r \in Proc$. $L'_r = L_r \cup G$ is a set of *visible variables* in r. Let L_r^{ref} be the set of all the call-by-reference formal parameters of the procedure r, L_n^{ref} (resp. L_n) be the set L_r^{ref} (resp. L_r) such that $n \in N_r$. Given a procedure call $stmt = r(v_1, ..., v_m)$ at the control point n whose return address is n', for every formal parameter v' of r, let $fRef(n')(v') \in \{v_1, ..., v_m\}$ be the actual parameter of v' at the caller site n'.

A *valuation* $\xi \subseteq L'_r$ is a subset of L'_r meaning that the Boolean value of $x \in L'_r$ is 1 if $x \in \xi$, otherwise 0. Let $\xi(x) = 1$ if $x \in \xi$, otherwise 0. Let $\xi[d/x]$ be the valuation such that $\xi[d/x](y) = d$ if $x = y$, otherwise $\xi(y)$. The edges in E_r are of the form $(n, \xi, stmt, n')$ meaning that n' is the next control point of n when the valuation at n is ξ, where $stmt$ is the statement at n. Let $[\![stmt]\!]_\xi$ be the valuation after executing the statement $stmt$ that is neither a procedure call nor return. The details of the execution model and semantics of other statements refer to [3].

5.2 Modeling Approach

W.l.o.g., we assume that call-by-reference actual parameters are local variables. Indeed, global variables are always visible for all procedures and do not need to be passed by parameters. Different from call-by-value parameter passing which keeps its own copy at callee site, a parameter passed by call-by-reference will not keep its own copy at callee site. Precisely speaking, the values of call-by-reference actual parameters at a caller procedure are always same as the values of the corresponding formal parameters at the callee procedure. We will use transductions to encode the changing of call-by-reference formal parameters, as transductions in TrPDSs allow us to manipulate the stack content rather than the top of stack in PDSs.

The construction of TrPDSs from Boolean programs BP with call-by-reference parameter passing follows the standard modeling approach of PDSs from Boolean programs [13] except the assignments with call-by-reference formal parameter as left value for which the side-effect of assignments should also effect on the value of the corresponding actual parameters at the corresponding caller site. The valuations of global variables G are put in the control locations of the TrPDSs, the pairs of the valuations of local variables and control points (i.e., nodes) of the program are stored in the stack of the TrPDSs. The TrPDS model will be $\mathcal{P} = (2^G, \bigcup_{r \in Proc}(N_r \times 2^L_r), \mathcal{T}, \Delta)$. A configuration of the TrPDS model is in the form of $c = \langle \xi, (n_0, \xi_0) \cdots (n_k, \xi_k) \rangle$ meaning that the execution of BP is at the control point n_0 with ξ as the valuation of global variables, ξ_0 as the valuation of local variables of the procedure containing n_0. Moreover, $(n_1, \xi_1) \cdots (n_k, \xi_k)$ is the calling history of the execution such that for every $i : 1 \leq i \leq k$, n_i is the return address of the procedure call that jump into the procedure containing n_{i-1} and ξ_i is the stored valuation of local variables when the procedure call is made. Different from Boolean programs only with call-by-value parameter passing, a local variable of a procedure may be a call-by-reference parameter of the procedure. In this case, the value of a local variable and its referenced variable are identical. Therefore, the potential possible configurations of the TrPDS model should only have admitted valuations with respect to the local variables and their referenced variables.

Formally, a word $(n_0,\xi_0)\cdots(n_k,\xi_k)$ or a configuration $\langle \xi,(n_0,\xi_0)\cdots(n_k,\xi_k)\rangle$ is *admissible* if for every $i: 0 \le i \le k-1$ and every $v \in L^{ref}_{n_i}$, $v \in \xi_i$ iff $fRef(n_{i+1})(v) \in \xi_{i+1}$.

Given two variable sets $\xi'_0,\xi_0 \subseteq \bigcup_{r\in Proc} L^{ref}_r$, let $\overrightarrow{(\xi'_0,\xi_0)} \subseteq \Gamma^*\times\Gamma^*$ be the transduction such that for every $((n_1,\xi_1)...(n_k,\xi_k),(n_1,\xi'_1)...(n_k,\xi'_k)) \in \Gamma^*\times\Gamma^*$, $((n_1,\xi_1)...(n_k,\xi_k),(n_1,\xi'_1)...(n_k,\xi'_k)) \in \overrightarrow{(\xi'_0,\xi_0)}$ iff $(n_1,\xi_1)...(n_k,\xi_k)$ is admissible, and for every $i: 1 \le i \le k$, $\xi'_i = \xi_i \cup \{fRef(n_i)(v) \mid v \in \xi'_{i-1} \setminus \xi_{i-1}\} \setminus \{fRef(n_i)(v) \mid v \in \xi_{i-1} \setminus \xi'_{i-1}\}$. Intuitively, given an admissible word $(n_1,\xi_1)...(n_k,\xi_k)$ and two sets ξ_0 and ξ'_0 denoting respectively the valuations of local variables before and after an assignment, $(n_1,\xi'_1)...(n_k,\xi'_k)$ is the admissible word obtained from $(n_1,\xi_1)...(n_k,\xi_k)$ with the updating of all the actual parameters of the corresponding formal call-by-reference parameters with respect to ξ'_0 and ξ_0.

The set Δ of transition rules that mimic the control flow of BP is defined as follows: for every edge $e=(n,\xi,stmt,n')$ in a procedure r,

- $\langle \xi\cap G,(n,\xi\cap L_r)\rangle \overset{\tau_{id}}{\hookrightarrow} \langle \xi\cap G,(r'_{enrty},\xi')(n',\xi\cap L_r)\rangle \in \Delta$ if $stmt$ is a procedure call $r'(v_1,...,v_m)$, where $\xi' = \{v \in L_{r'} \mid fRef(n')(v) \in \xi\}$;
- $\langle \xi\cap G,(n,\xi\cap L_r)\rangle \overset{\tau_{id}}{\hookrightarrow} \langle \xi\cap G,\epsilon\rangle \in \Delta$ if $stmt$ is a return,
- $\langle \xi\cap G,(n,\xi\cap L_r)\rangle \overset{\tau_e}{\hookrightarrow} \langle [\![stmt]\!]_\xi\cap G,(n',[\![stmt]\!]_\xi\cap L_r)\rangle \in \Delta$ otherwise, where $\tau_e = \overrightarrow{([\![stmt]\!]_\xi,\xi)}$.

The set $\mathcal{T}$ of transductions is $\{\tau_{id},\tau_e \mid e=(n,\xi,stmt,n') \in E_r, r\in Proc, stmt$ is neither a procedure call nor return$\}$. Intuitively, the TrPDS model mimics the execution of BP. The intuition behind function calls and returns is similar as translating from Boolean programs into PDSs. For details refer to [13]. We explain the assignments.

Suppose the execution of BP is at the control point n with the valuation ξ, and the calling history is $(n_1,\xi_1)...(n_k,\xi_k)$. If the edge $e=(n,\xi,stmt,n')$ is in a procedure r such that $stmt$ is neither a procedure call nor return, then, the execution of BP will move from n to the next point n' with the valuation $[\![stmt]\!]_\xi$. Moreover, the local variables at n_1 that corresponds to the formal call-by-reference parameters in r should take the values of the call-by-reference parameters at n'. Similarly, for every $i: 2 \le i \le k$, the local variables at n_i that corresponds to the formal call-by-reference parameters in the procedure containing n_{i-1} should take the values of the call-by-reference parameters at n_i. Therefore, we add the transition rule $\langle \xi\cap G,(n,\xi\cap L_r)\rangle \overset{\tau_e}{\hookrightarrow} \langle [\![stmt]\!]_\xi\cap G,(n',[\![stmt]\!]_\xi\cap L_r)\rangle$ into Δ which allows the TrPDS model to move from the configuration $\langle \xi\cap G,(n,\xi\cap L_r)(n_1,\xi_1)...(n_k,\xi_k)\rangle$ to $\langle [\![stmt]\!]_\xi\cap G,(n',[\![stmt]\!]_\xi\cap L_r)(n_1,\xi'_1)...(n_k,\xi'_k)\rangle$ for every $((n_1,\xi_1)...(n_k,\xi_k),(n_1,\xi'_1)...(n_k,\xi'_k)) \in \tau_e$. The transduction $\tau_e = \overrightarrow{([\![stmt]\!]_\xi,\xi)}$ correctly specifies the updating of all the actual parameters of the corresponding call-by-reference parameters in the calling history.

▶ Remark. From the definitions of transductions $\mathcal{T}$ and admissible, we can see that all the reachable configurations in the TrPDS model from an admissible configuration are also admissible.

Given two transductions $\tau_1 = \overrightarrow{(\xi'_1,\xi_1)}$, $\tau_2 = \overrightarrow{(\xi'_2,\xi_2)} \in \mathcal{T}$, then

$$\tau_1\circ\tau_2 = \begin{cases} \emptyset & \text{if } \xi'_1 \neq \xi_2,\\ \overrightarrow{(\xi'_2,\xi_1)} & \text{otherwise.}\end{cases}$$

Given two symbols $(n_1,\xi_1),(n'_1,\xi'_1) \in \Gamma^*$ and a transduction $\tau = \overrightarrow{(\xi',\xi)} \in \mathcal{T}$, $\lceil (n_1,\xi_1),(n'_1,\xi'_1)\rfloor^{-1}\tau$ is $\overrightarrow{(\xi'_1,\xi_1)}$ if $n_1 = n'_1 \wedge \xi'_1 = (\xi_1 \cup \{fRef(n_1)(v) \mid v \in \xi' \setminus \xi\}) \setminus \{fRef(n_1)(v) \mid v \in \xi \setminus \xi'\}$, $\emptyset$ otherwise.

Then, we can get that $\langle\mathcal{T}\rangle \subseteq \{\overrightarrow{(\xi',\xi)} \mid \exists r \in Proc : \xi,\xi' \subseteq L_r\}$ which is finite.

▶ **Theorem 15.** *The Boolean program BP can reach a control point n of the procedure r with the valuation ξ and the calling history ω from a control point n' of r' with the valuation ξ' and the calling history ω' iff* $\langle \xi' \cap G, (n', \xi' \cap L_{r'})\omega' \rangle \Longrightarrow^* \langle \xi \cap G, (n, \xi \cap L_r)\omega \rangle$.

Using Theorem 15, we can verify safety properties of Boolean programs with mixed call-by-reference and call-by-value parameter passing via solving the reachability problem of TrPDSs. The efficiency heavily relies upon the number of transductions from the modeling of the Boolean program and the saturation. From the modeling, the number of transductions is linear in the size of the edges labeled by assignments which assign values to reference variables. During the saturation procedure, transductions are computing via left quotient and composition operators. Therefore, the number of transductions added by the saturation procedure is exponential in the size of return nodes in the Boolean program and doubly exponential in the size of reference variables.

▶ Remark. One may argue that Boolean program with mixed call-by-reference and call-by-value parameter passing can be translated into a Boolean program with only call-by-value parameter passing by using global variables which can be verified by existing techniques such as [3]. However, this will leads to larger state space and may degrade performance.

6 Related Work

Model-checking techniques for PDSs were widely studied and applied to program analysis in the literature [6, 13, 16, 17, 26]. PDSs with checkpoint were introduced in [15] as an extension of PDSs. PDSs with checkpoint can inspect the stack content and are applied to analyse programs with runtime inspection. The reachability problem and LTL model-checking for PDSs with checkpoint were studied in [15] and were applied to the analysis of the HTML5 parser specification in [22]. CTL model-checking for PDSs with checkpoint was studied in [25, 28]. A similar extension of PDSs was used to formulate abstract garbage collection in the control flow analysis of higher-order programs [12].

Weighted PDSs and extended weighted PDSs were introduced in [23, 19] for data-flow analysis purpose. These two extensions associate transitions with elements from semiring domains. The reachability problem is decidable for bounded idempotent semiring. (Extended) weighted PDSs and TrPDSs are quite different two computation models. At least, the elements from semiring can neither inspect nor modify the stack content except the top most symbol on the stack.

Recently, well-structured PDSs (WSPDSs) that combine well-structured transition systems and PDSs was introduced by [10] in which the infinite set of control states and the infinite stack alphabet are well-quasi-order. WSPDS is a powerful model in which recursive vector addition system with states [4, 5], multi-set PDSs [24] and dense-timed PDSs are subsumed [11]. However, the reachability problem is undecidable for WSPDSs. But coverability becomes decidable when the set of control states is finite. In TrPDSs, the set of control states and the stack alphabet are both finite, but the transductions can inspect and modify the stack content.

We should clarify the relation between our work and the work [30]. TrPDSs were first introduced in [30] and are generalization of PDSs with checkpoint and discrete-timed PDSs. The authors showed that TrPDSs can be simulated by PDSs and proposed a saturation procedure to compute pre^* which different from ours. Indeed, our approach is essential to get an efficient implementation algorithm. We also proposed a saturation procedure to compute $post^*$ and its efficient implementation algorithm. These two efficient implementation algorithms necessarily improve the complexity due to the fact that the algorithms have

better complexity than the saturation procedures for pushdown systems. Moveover, we presented a potential application of TrPDSs to modeling and verifying Boolean programs with call-by-reference parameter passing.

7 Conclusion and Future Work

We introduced two saturation procedures to compute pre^* and $post^*$. We also presented two efficient implementation algorithms for the saturation procedures and measured their complexity. We showed that TrPDSs are powerful enough to model Boolean programs with call-by-reference parameter passing. This allows us to verify safety properties of Boolean programs with mixed call-by-reference and call-by-value parameter passing.

In future, we plan to implement our techniques in a tool and investigate BDD-based symbolic algorithms by representing transductions and valuations of global and local variables in BDDs.

Acknowledgements. We want to thank Lijun Zhang and Zhilin Wu for discussions and suggestions.

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jari Stenman. Dense-timed pushdown automata. In *Proceedings of LICS*, pages 35–44, 2012.
2 Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Jari Stenman. The minimal cost reachability problem in priced timed pushdown systems. In *Proceedings of LATA*, pages 58–69, 2012.
3 Thomas Ball and Sriram K. Rajamani. Bebop: A symbolic model checker for boolean programs. In *Proceedings of SPIN*, pages 113–130, 2000.
4 Ahmed Bouajjani and Michael Emmi. Analysis of recursively parallel programs. In *Proceedings of POPL*, pages 203–214, 2012.
5 Ahmed Bouajjani and Michael Emmi. Analysis of recursively parallel programs. *ACM Trans. Program. Lang. Syst.*, 35(3):10, 2013.
6 Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proceedings of CONCUR*, pages 135–150, 1997.
7 Ahmed Bouajjani, Markus Müller-Olm, and Tayssir Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *Proceedings of CONCUR*, pages 473–487, 2005.
8 Tomás Brázdil, Antonín Kucera, and Oldrich Strazovský. On the decidability of temporal properties of probabilistic pushdown automata. In *Proceedings of STACS*, pages 145–157, 2005.
9 Luca Breveglieri, Alessandra Cherubini, Claudio Citrini, and Stefano Crespi-Reghizzi. Multi-push-down languages and grammars. *Int. J. Found. Comput. Sci.*, 7(3):253–292, 1996.
10 Xiaojuan Cai and Mizuhito Ogawa. Well-structured pushdown systems. In *Proceedings of CONCUR*, pages 121–136, 2013.
11 Xiaojuan Cai and Mizuhito Ogawa. Well-structured pushdown system: Case of dense timed pushdown automata. In *Proceedings of FLOPS*, pages 336–352, 2014.
12 Christopher Earl, Ilya Sergey, Matthew Might, and David Van Horn. Introspective pushdown analysis of higher-order programs. In *Proceedings of ICFP*, pages 177–188, 2012.
13 Javier Esparza, David Hansel, Peter Rossmanith, and Stefan Schwoon. Efficient algorithms for model checking pushdown systems. In *Proceedings of CAV*, pages 232–247, 2000.

14 Javier Esparza, Antonín Kucera, and Richard Mayr. Model checking probabilistic pushdown automata. In *Proceedings of LICS*, pages 12–21, 2004.

15 Javier Esparza, Antonín Kucera, and Stefan Schwoon. Model-checking LTL with regular valuations for pushdown systems. In *Proceedings of TACS*, pages 316–339, 2001.

16 Javier Esparza and Stefan Schwoon. A bdd-based model checker for recursive programs. In *Proceedings of CAV*, pages 324–336, 2001.

17 Matthew Hague and C.-H. Luke Ong. Analysing mu-calculus properties of pushdown systems. In *Proceedings of SPIN*, pages 187–192, 2010.

18 Vineet Kahlon, Franjo Ivancic, and Aarti Gupta. Reasoning about threads communicating via locks. In *Proceedings of CAV*, pages 505–518, 2005.

19 Akash Lal, Thomas W. Reps, and Gogul Balakrishnan. Extended weighted pushdown systems. In *Proceedings of CAV*, pages 434–448, 2005.

20 Guoqiang Li, Xiaojuan Cai, Mizuhito Ogawa, and Shoji Yuen. Nested timed automata. In *Proceedings of FORMATS*, pages 168–182, 2013.

21 Xin Li and Mizuhito Ogawa. Conditional weighted pushdown systems and applications. In *Proceedings of PEPM*, pages 141–150, 2010.

22 Yasuhiko Minamide and Shunsuke Mori. Reachability analysis of the HTML5 parser specification and its application to compatibility testing. In *Proceedings of FM*, pages 293–307, 2012.

23 Thomas W. Reps, Stefan Schwoon, and Somesh Jha. Weighted pushdown systems and their application to interprocedural dataflow analysis. In *Proceedings of SAS*, pages 189–213, 2003.

24 Koushik Sen and Mahesh Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *Proceedings of CAV*, pages 300–314, 2006.

25 Fu Song and Tayssir Touili. Efficient CTL model-checking for pushdown systems. In *Proceedings of CONCUR*, pages 434–449, 2011.

26 Fu Song and Tayssir Touili. PuMoC: a CTL model-checker for sequential programs. In *Proceedings of ASE*, pages 346–349, 2012.

27 Fu Song and Tayssir Touili. Model checking dynamic pushdown networks. In *Proceedings of APLAS*, pages 33–49, 2013.

28 Fu Song and Tayssir Touili. Efficient CTL model-checking for pushdown systems. *Theor. Comput. Sci.*, 549:127–145, 2014.

29 Fu Song and Tayssir Touili. Model checking dynamic pushdown networks. *Formal Asp. Comput.*, 27(2):397–421, 2015.

30 Yuya Uezato and Yasuhiko Minamide. Pushdown systems with stack manipulation. In *Proceedings of ATVA*, pages 412–426, 2013.

Characteristic Bisimulations for Higher-Order Session Processes

Dimitrios Kouzapas[1,2], Jorge A. Pérez[3], and Nobuko Yoshida[1]

1 **Imperial College London, UK**
2 **University of Glasgow, UK**
3 **University of Groningen, The Netherlands**

Abstract

Characterising contextual equivalence is a long-standing issue for higher-order (process) languages. In the setting of a higher-order π-calculus with sessions, we develop *characteristic bisimilarity*, a typed bisimilarity which fully characterises contextual equivalence. To our knowledge, ours is the first characterisation of its kind. Using simple values inhabiting (session) types, our approach distinguishes from untyped methods for characterising contextual equivalence in higher-order processes: we show that observing as inputs only a precise finite set of higher-order values suffices to reason about higher-order session processes. We demonstrate how characteristic bisimilarity can be used to justify optimisations in session protocols with mobile code communication.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs, F.3.2 Semantics of Programming Languages

Keywords and phrases Behavioural equivalences, session types, higher-order process calculi

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.398

1 Introduction

Context. In *higher-order process calculi* communicated values may contain processes. Higher-order concurrency has received significant attention from untyped and typed perspectives (see, e.g., [15, 5, 10, 6, 13]). In this work, we consider HOπ, a higher-order process calculus with *session primitives*: in addition to functional abstractions/applications (as in the call-by-value λ-calculus), HOπ contains constructs for synchronisation on shared names, session communication on linear names, and recursion. Thus, HOπ processes may specify protocols for higher-order processes that can be type-checked using *session types* [3]. Although models of session communication with process passing exist [12, 2], their *behavioural equivalences* remain little understood. Since types can limit the contexts (environments) in which processes can interact, typed equivalences usually offer *coarser* semantics than untyped semantics. Hence, clarifying the status of these equivalences is key to, e.g., justify non-trivial optimisations in protocols involving both name- and process-passing.

A well-known behavioural equivalence for higher-order processes is *context bisimilarity* [16]. This characterisation of barbed congruence offers an adequate distinguishing power at the price of heavy universal quantifications in output clauses. Obtaining alternative characterisations of context bisimilarity is thus a recurring, important problem for higher-order calculi—see, e.g., [15, 16, 5, 6]. In particular, Sangiorgi [15, 16] has given characterisations of context bisimilarity for higher-order processes; such characterisations, however, do not scale to calculi with *recursive types*, which are essential to session-based concurrency. A characterisation that solves this limitation was developed by Jeffrey and Rathke in [5].

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 398–411

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

This Work. Building upon [15, 16, 5], our discovery is that *linearity* of session types plays a vital role in solving the open problem of characterising context bisimilarity for higher-order mobile processes with session communications. Our approach is to exploit the coarser semantics induced by session types to limit the behaviour of higher-order session processes. Formally, we enforce this limitation by defining a *refined* labelled transition system (LTS) which effectively narrows down the spectrum of allowed process behaviours, exploiting elementary processes inhabiting session types. We then introduce *characteristic bisimilarity*: this new notion of typed bisimilarity is *tractable*, in that it relies on the refined LTS for input actions and, more importantly, does not appeal to universal quantifications on output actions. Our main result is that characteristic bisimilarity coincides with context bisimilarity. Besides confirming the value of characteristic bisimilarity as an useful reasoning technique for higher-order processes with sessions, this result is remarkable also from a technical perspective, for associated completeness proofs do not require operators for name-matching, in contrast to untyped methods for higher-order processes with recursive types [5].

We explain how we exploit session types to define characteristic bisimilarity. Key notions are *triggered* and *characteristic processes/values*. Below, we write $s?(x).P$ for an input on endpoint s, and $\overline{s}!\langle V\rangle.Q$ for an output of value V on endpoint $\overline{s}$ (the *dual* of s). Also, $R \xrightarrow{s?\langle V\rangle} R'$ denotes an input transition along n and $R \xrightarrow{(\nu\,\widetilde{m})s!\langle V\rangle} R'$ denotes an output transition along s, sending value V, and extruding names $\widetilde{m}$. Weak transitions are as usual. Throughout the paper, we write $\Re, \Re', \ldots$ to denote binary relations on (typed) processes.

Issues of Context Bisimilarity. Context bisimilarity ($\approx$, Def. 10) is an overly demanding relation on higher-order processes. There are two issues, associated to demanding clauses for output and input actions. A *first issue* is the universal quantification on the output clause of context bisimilarity. Suppose $P\,\Re\,Q$, for some context bisimulation $\Re$. We have:

($\star$) Whenever $P \xrightarrow{(\nu\,\widetilde{m_1})s!\langle V\rangle} P'$ there exist Q' and W such that $Q \overset{(\nu\,\widetilde{m_2})s!\langle W\rangle}{\Longrightarrow} Q'$ and, ***for all*** R with $\mathtt{fv}(R) = x$, $(\nu\,\widetilde{m_1})(P' \mid R\{^V/_x\})\,\Re\,(\nu\,\widetilde{m_2})(Q' \mid R\{^W/_x\})$.

The *second issue* is due to inputs: it follows from the fact that we work with an *early* labelled transition system (LTS). Thus, an input prefix may observe infinitely many different values. To alleviate this burden, in *characteristic bisimilarity* ($\approx^{\mathsf{C}}$) we take two (related) steps:

(a) We replace ($\star$) with a clause involving a *more tractable* process closure; and
(b) We refine inputs to avoid observing infinitely many actions on the same input prefix.

Trigger Processes. To address (a), we exploit session types. We first observe that closure $R\{^V/_x\}$ in ($\star$) is context bisimilar to the process $P = (\nu\,s)((\lambda z.\, z?(x).R)\,s \mid \overline{s}!\langle V\rangle.\mathbf{0})$. In fact, we do have $P \approx R\{^V/_x\}$, since application and reduction of dual endpoints are deterministic.

Now let us consider process T_V below, where t is a fresh name. If T_V inputs value $\lambda z.\, z?(x).R$ then we can simulate the closure of P:

$$T_V = t?(x).(\nu\,s)(x\,s \mid \overline{s}!\langle V\rangle.\mathbf{0}) \quad \text{and} \quad T_V \xrightarrow{t?\langle \lambda z.\, z?(x).R\rangle} P \approx R\{^V/_x\} \tag{1}$$

Processes such as T_V offer a value at a fresh name; this class of ***trigger processes*** already suggests a tractable formulation of bisimilarity without the demanding clause ($\star$). Process T_V in (1) requires a higher-order communication along t. As we explain below, we can give an alternative trigger process; the key is using *elementary inhabitants* of session types.

Characteristic Processes and Values. To address (b), we limit the possible input values (such as $\lambda z.\, z?(x).R$ above) by exploiting session types. The key concept is that of ***charac-***

teristic process/value of a type, the simplest term inhabiting that type (Def. 11). This way, for instance, let $S = ?(S_1 \to \diamond); !\langle S_2 \rangle; \mathtt{end}$ be a session type: first input an abstraction, then output a value of type S_2. Then, process $u?(x).(u!\langle s_2 \rangle.\mathbf{0} \mid x\, s_1)$ is a characteristic process for S along u. Given a session type S, we write $(\!|S|\!)^u$ for its characteristic process along u (cf. Def. 11). Also, given value type U, then $(\!|U|\!)_{\mathsf{c}}$ denotes its characteristic value. As we explain now, we use $(\!|U|\!)_{\mathsf{c}}$ to limit input transitions.

Refined Input Transitions. To refine input transitions, we need to observe an additional value, $\lambda x.\, t?(y).(y\, x)$, called the ***trigger value***. This is necessary: it turns out that a characteristic value alone as the observable input is not enough to define a sound bisimulation (cf. Ex. 13). Intuitively, the trigger value is used to observe/simulate application processes. Based on the above discussion, we refine the transition rule for input actions (cf. Def. 14). Roughly, the refined rule is:

$$P \xrightarrow{s?\langle V \rangle} P' \wedge (V = m \vee V \equiv \lambda x.\, t?(y).(y\, x) \vee V \equiv (\!|U|\!)_{\mathsf{c}} \text{ with } t \text{ fresh}) \;\Rightarrow\; P' \overset{s?\langle V \rangle}{\longmapsto} P'$$

Note the distinction between standard and refined transitions: $\xrightarrow{s?\langle V \rangle}$ vs. $\overset{s?\langle V \rangle}{\longmapsto}$. Our refined rule for (higher-order) input admits only names, trigger values, and characteristic values. Using this rule, we define an alternative, refined LTS on typed processes: we use it to define characteristic bisimulation ($\approx^{\mathsf{c}}$, Def. 15), in which the demanding clause ($\star$) is replaced with a more tractable output clause based on characteristic trigger processes (cf. (2)).

Characteristic Triggers. Following the same reasoning as (1), we can use an alternative trigger process, called ***characteristic trigger process*** with type U to replace clause ($\star$):

$$t \Leftarrow V : U \stackrel{\mathtt{def}}{=} t?(x).(\nu\, s)((\!|?(U); \mathtt{end}|\!)^s \mid \overline{s}!\langle V \rangle.\mathbf{0}) \tag{2}$$

This is justified because in (1) $T_V \overset{t?\langle (\!|?(U);\mathtt{end}|\!)_{\mathsf{c}} \rangle}{\longmapsto} \approx (\nu\, s)((\!|?(U); \mathtt{end}|\!)^s \mid \overline{s}!\langle V \rangle.\mathbf{0})$. Thus, unlike process (1), the characteristic trigger process in (2) does not involve a higher-order communication on t. In contrast to previous approaches [15, 5] our characteristic trigger processes do *not* use recursion or replication. This is key to preserve linearity of session endpoints.

It is also noteworthy that HOπ lacks name matching, which is usually crucial to prove completeness of bisimilarity—see, e.g., [5]. Instead of matching, we use types: a process trigger embeds a name into a characteristic process so to observe its session behaviour.

Outline. Next we present the session calculus HOπ. § 3 gives the session type system for HOπ and states type soundness. § 4 develops *characteristic* bisimilarity and states our main result: characteristic and context bisimilarities coincide for well-typed HOπ processes (Thm. 18). § 5 concludes with related works.

2 A Higher-Order Session π-Calculus

We introduce the *Higher-Order Session π-Calculus* (HOπ). HOπ includes both name- and abstraction-passing, shared and session communication, as well as recursion; it is essentially the language proposed in [12] (where tractable bisimilarities are not addressed).

$$
\begin{array}{llllllllll}
u, w & ::= & n & | & x, y, z & n & ::= & a, b & | & s, \overline{s} \qquad V, W \quad ::= \quad u \quad | \quad \lambda x.\, P \\
P, Q & ::= & u!\langle V\rangle.P & | & u?(x).P & | & u \triangleleft l.P & | & u \triangleright \{l_i : P_i\}_{i\in I} \\
 & | & X & | & \mu X.P & | & V\, W & | & P \mid Q \quad | \quad (\nu\, n)P \quad | \quad \mathbf{0}
\end{array}
$$

$$
P \mid \mathbf{0} \equiv P \quad P_1 \mid P_2 \equiv P_2 \mid P_1 \quad P_1 \mid (P_2 \mid P_3) \equiv (P_1 \mid P_2) \mid P_3 \quad \mu X.P \equiv P\{^{\mu X.P}/_X\}
$$
$$
(\nu\, n)\mathbf{0} \equiv \mathbf{0} \qquad P \mid (\nu\, n)Q \equiv (\nu\, n)(P \mid Q)\ (n \notin \mathtt{fn}(P)) \qquad P \equiv Q \text{ if } P \equiv_\alpha Q
$$

$$
\begin{array}{llll}
\text{[App]} & (\lambda x.\, P)\, V \longrightarrow P\{^V/_x\} & \text{[Pass]} & n!\langle V\rangle.P \mid \overline{n}?(x).Q \longrightarrow P \mid Q\{^V/_x\} \\
\text{[Res]} & P \longrightarrow P' \Rightarrow (\nu\, n)P \longrightarrow (\nu\, n)P' & \text{[Sel]} & n \triangleleft l_j.Q \mid \overline{n} \triangleright \{l_i : P_i\}_{i\in I} \longrightarrow Q \mid P_j \quad (j \in I) \\
\text{[Par]} & P \longrightarrow P' \Rightarrow P \mid Q \longrightarrow P' \mid Q & \text{[Cong]} & P \equiv Q \longrightarrow Q' \equiv P' \Rightarrow P \longrightarrow P'
\end{array}
$$

Figure 1 HOπ: Syntax and Operational Semantics (Structural Congruence and Reduction).

Syntax. The syntax of HOπ is given in Fig. 1 (upper part). We use $a, b, c, \ldots$ (resp. $s, \overline{s}, \ldots$) to range over shared (resp. session) names. We use $m, n, t, \ldots$ for session or shared names. We define the dual operation over names n as $\overline{n}$ with $\overline{\overline{s}} = s$ and $\overline{a} = a$. Intuitively, names s and $\overline{s}$ are dual *endpoints* while shared names represent non-deterministic points. Variables are denoted with $x, y, z, \ldots$, and recursive variables are denoted with $X, Y \ldots$. An abstraction $\lambda x.\, P$ is a process P with name parameter x. Values V, W include identifiers $u, v, \ldots$ and abstractions $\lambda x.\, P$ (first- and higher-order values, resp.).

Terms include π-calculus constructs for sending/receiving values V. Process $u!\langle V\rangle.P$ denotes the output of V over name u, with continuation P; process $u?(x).P$ denotes the input prefix on name u of a value that will substitute variable x in continuation P. Recursion $\mu X.P$ binds the recursive variable X in process P. Process $V\, W$ is the application which substitutes values W on the abstraction V. Typing ensures that V is not a name. Processes $u \triangleright \{l_i : P_i\}_{i\in I}$ and $u \triangleleft l.P$ define labelled choice: given a finite index set I, process $u \triangleright \{l_i : P_i\}_{i\in I}$ offers a choice among processes with pairwise distinct labels; process $u \triangleleft l.P$ selects label l on name u and then behaves as P. Constructs for inaction $\mathbf{0}$, parallel composition $P_1 \mid P_2$, and name restriction $(\nu\, n)P$ are standard. Session name restriction $(\nu\, s)P$ binds endpoints s and $\overline{s}$ in P. We use $\mathtt{fv}(P)$ and $\mathtt{fn}(P)$ to denote sets of free variables and names; we assume V in $u!\langle V\rangle.P$ does not include free recursive variables. If $\mathtt{fv}(P) = \emptyset$, we call P *closed*.

Semantics. Fig. 1 (lower part) defines the operational semantics of HOπ, given as a reduction relation that relies on a *structural congruence* $\equiv$. We assume the expected extension of $\equiv$ to values V. Reduction is denoted $\longrightarrow$; some intuitions on the rules in Fig. 1 follow. Rule [App] is a value application; rule [Pass] defines a shared interaction at n (with $\overline{n} = n$) or a session interaction; rule [Sel] is the standard rule for labelled choice/selection: given an index set I, a process selects label l_j on name n over a set of labels $\{l_i\}_{i\in I}$ offered by a branching on the dual endpoint $\overline{n}$; and other rules are standard. We write $\longrightarrow^*$ for a multi-step reduction.

▶ **Example 1** (Hotel Booking Scenario). To illustrate HOπ and its expressive power, we consider a usecase scenario that adapts the example given by Mostrous and Yoshida [12, 13]. The scenario involves a Client process that wants to book a hotel room. Client narrows the choice down to two hotels, and requires a quote from the two in order to decide. The round-trip time (RTT) required for taking quotes from the two hotels is not optimal, so the client sends mobile processes to both hotels to automatically negotiate and book a room.

We now present two HOπ implementations of this scenario. For convenience, we write $\mathtt{if}\ e\ \mathtt{then}\ (P_1\ \mathtt{;}\ P_2)$ to denote a conditional process that executes P_1 or P_2 depending on

boolean expression e (encodable using labelled choice). The *first implementation* is as follows:

$$P_{xy} \stackrel{\text{def}}{=} x!\langle \mathsf{room}\rangle.x?(\mathsf{quote}).y!\langle \mathsf{quote}\rangle.y \triangleright \left\{ \begin{array}{l} \mathsf{accept} : x \triangleleft \mathsf{accept}.x!\langle \mathsf{credit}\rangle.\mathbf{0}, \\ \mathsf{reject} : x \triangleleft \mathsf{reject}.\mathbf{0} \end{array} \right\}$$

$$\begin{array}{rl} \mathsf{Client}_1 \stackrel{\text{def}}{=} & (\nu\, h_1, h_2)(s_1!\langle \lambda x.\, P_{xy}\{h_1/y\}\rangle.s_2!\langle \lambda x.\, P_{xy}\{h_2/y\}\rangle.\mathbf{0} \mid \\ & \overline{h_1}?(x).\overline{h_2}?(y).\mathtt{if}\ x \leq y\ \mathtt{then}\ (\overline{h_1} \triangleleft \mathsf{accept}.\overline{h_2} \triangleleft \mathsf{reject}.\mathbf{0}\ \mathbf{;}\ \overline{h_1} \triangleleft \mathsf{reject}.\overline{h_2} \triangleleft \mathsf{accept}.\mathbf{0})) \end{array}$$

Process Client_1 sends two abstractions with body P_{xy}, one to each hotel, using sessions s_1 and s_2. That is, P_{xy} is the mobile code: while name x is meant to be instantiated by the hotel as the negotiating endpoint, name y is used to interact with Client_1. Intuitively, process P_{xy} (i) sends the room requirements to the hotel; (ii) receives a quote from the hotel; (iii) sends the quote to Client_1; (iv) expects a choice from Client_1 whether to accept or reject the offer; (v) if the choice is accept then it informs the hotel and performs the booking; otherwise, if the choice is reject then it informs the hotel and ends the session. Client_1 instantiates two copies of P_{xy} as abstractions on session x. It uses two fresh endpoints h_1, h_2 to substitute channel y in P_{xy}. This enables communication with the mobile code(s). In fact, Client_1 uses the dual endpoints $\overline{h_1}$ and $\overline{h_2}$ to receive the negotiation result from the two remote instances of P and then inform the two processes for the final booking decision.

Notice that the above implementation does not affect the time needed for the whole protocol to execute, since the two remote processes are used to send/receive data to Client_1.

We present now a *second implementation* in which the two mobile processes are meant to interact with each other (rather than with the client) to reach to an agreement:

$$\begin{array}{rcl} R_x & \stackrel{\text{def}}{=} & \mathtt{if}\ \ \mathsf{quote}_1 \leq \mathsf{quote}_2\ \mathtt{then}\ (x \triangleleft \mathsf{accept}.x!\langle \mathsf{credit}\rangle.\mathbf{0}\ \mathbf{;}\ x \triangleleft \mathsf{reject}.\mathbf{0}) \\ Q_1 & \stackrel{\text{def}}{=} & x!\langle \mathsf{room}\rangle.x?(\mathsf{quote}_1).y!\langle \mathsf{quote}_1\rangle.y?(\mathsf{quote}_2).R_x \\ Q_2 & \stackrel{\text{def}}{=} & x!\langle \mathsf{room}\rangle.x?(\mathsf{quote}_1).y?(\mathsf{quote}_2).y!\langle \mathsf{quote}_1\rangle.R_x \\ \mathsf{Client}_2 & \stackrel{\text{def}}{=} & (\nu\, h)(s_1!\langle \lambda x.\, Q_1\{h/y\}\rangle.s_2!\langle \lambda x.\, Q_2\{\overline{h}/y\}\rangle.\mathbf{0}) \end{array}$$

Processes Q_1 and Q_2 negotiate a quote from the hotel in the same fashion as process P_{xy} in Client_1. The key difference with respect to P_{xy} is that y is used for interaction between process Q_1 and Q_2. Both processes send their quotes to each other and then internally follow the same logic to reach to a decision. Process Client_2 then uses sessions s_1 and s_2 to send the two instances of Q_1 and Q_2 to the two hotels, using them as abstractions on name x. It further substitutes the two endpoints of a fresh channel h to channels y respectively, in order for the two instances to communicate with each other.

The differences between Client_1 and Client_2 can be seen in the sequence diagrams of Fig. 2. We will assign session types to these client processes in Example 4. Later on, we will show that they are behaviourally equivalent using characteristic bisimilarity; see Prop. 19.

3 Types and Typing

We define a session typing system for HOπ and state its main properties. Our system distills the key features of [12, 13]. We give selected definitions; see [7] for a full description.

Types. The syntax of types of HOπ is given below:

$$\begin{array}{llcl} \text{(value)} & U & ::= & C \mid L \\ \text{(name)} & C & ::= & S \mid \langle S\rangle \mid \langle L\rangle \\ \text{(abstr)} & L & ::= & U\!\to\!\diamond \mid U\!\multimap\!\diamond \end{array} \qquad \begin{array}{llcl} \text{(session)} & S & ::= & !\langle U\rangle;S \mid ?(U);S \mid \mathtt{end} \\ & & \mid & \oplus\{l_i : S_i\}_{i\in I} \mid \mu \mathtt{t}.S \mid \mathtt{t} \\ & & \mid & \&\{l_i : S_i\}_{i\in I} \end{array}$$

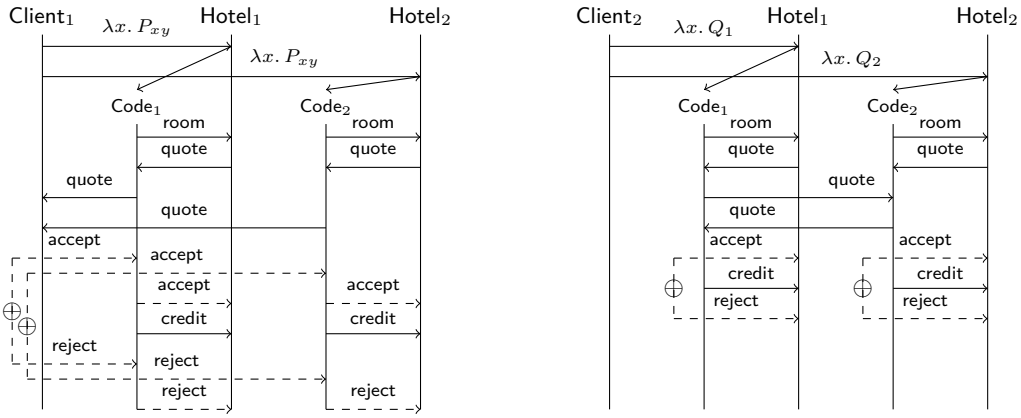


Figure 2 Sequence diagrams for Client_1 and Client_2 as in Example 1.

Value type U includes the first-order types C and the higher-order types L. Session types are denoted with S and shared types with $\langle S \rangle$ and $\langle L \rangle$. Types $U{\to}\diamond$ and $U{\multimap}\diamond$ denote *shared* and *linear* higher-order types, respectively. As for session types, the *output type* $!\langle U \rangle; S$ first sends a value of type U and then follows the type described by S. Dually, $?(U); S$ denotes an *input type*. The *branching type* $\&\{l_i : S_i\}_{i \in I}$ and the *selection type* $\oplus\{l_i : S_i\}_{i \in I}$ define the labelled choice. We assume the *recursive type* $\mu \mathtt{t}.S$ is guarded, i.e., $\mu \mathtt{t}.\mathtt{t}$ is not allowed. Type $\mathtt{end}$ is the termination type.

Following [1], we write S_1 dual S_2 if S_1 is the *dual* of S_2. Intuitively, duality converts ! into ? and $\oplus$ into & (and viceversa).

Typing Environments and Judgements. Typing *environments* are defined below:

$$\begin{array}{lllllllllll} \Gamma & ::= & \emptyset & | & \Gamma \cdot x : U{\to}\diamond & | & \Gamma \cdot u : \langle S \rangle & | & \Gamma \cdot u : \langle L \rangle & | & \Gamma \cdot X : \Delta \\ \Lambda & ::= & \emptyset & | & \Lambda \cdot x : U{\multimap}\diamond & & \Delta & ::= & \emptyset & | & \Delta \cdot u : S \end{array}$$

Γ maps variables and shared names to value types, and recursive variables to session environments; it admits weakening, contraction, and exchange principles. Λ maps variables to linear higher-order types, and Δ maps session names to session types. Both Λ and Δ are only subject to exchange. The domains of Γ, Λ and Δ are assumed pairwise distinct. $\Delta_1 \cdot \Delta_2$ is the disjoint union of Δ_1 and Δ_2. We define *typing judgements* for values and processes:

$$\Gamma; \Lambda; \Delta \vdash V \triangleright U \qquad\qquad \Gamma; \Lambda; \Delta \vdash P \triangleright \diamond$$

First judgement says that under environments $\Gamma; \Lambda; \Delta$ value V has type U; the second judgement says that under environments $\Gamma; \Lambda; \Delta$ process P has the process type $\diamond$. The type soundness result for HOπ (Thm. 3) relies on two auxiliary notions on session environments:

▶ **Definition 2** (Session Environments: Balanced/Reduction)**.** Let Δ be a session environment.

- A session environment Δ is *balanced* if whenever $s : S_1, \overline{s} : S_2 \in \Delta$ then S_1 dual S_2.
- We define the reduction relation $\longrightarrow$ on session environments as:

$$\Delta \cdot s : !\langle U \rangle; S_1 \cdot \overline{s} : ?(U); S_2 \longrightarrow \Delta \cdot s : S_1 \cdot \overline{s} : S_2$$
$$\Delta \cdot s : \oplus\{l_i : S_i\}_{i \in I} \cdot \overline{s} : \&\{l_i : S'_i\}_{i \in I} \longrightarrow \Delta \cdot s : S_k \cdot \overline{s} : S'_k \ (k \in I)$$

We rely on a typing system that is similar to the one developed in [12, 13]. We state the type soundness result for HOπ processes; see [7] for details of the associated proofs.

▶ **Theorem 3** (Type Soundness). *Suppose* $\Gamma;\emptyset;\Delta \vdash P \triangleright \diamond$ *with* Δ *balanced. Then* $P \longrightarrow P'$ *implies* $\Gamma;\emptyset;\Delta' \vdash P' \triangleright \diamond$ *and* $\Delta = \Delta'$ *or* $\Delta \longrightarrow \Delta'$ *with* Δ' *balanced.*

▶ **Example 4** (Hotel Booking Revisited). Assume $S = !\langle \mathsf{quote}\rangle;\&\{\mathsf{accept} : \mathtt{end}, \mathsf{reject} : \mathtt{end}\}$ and $U = !\langle \mathsf{room}\rangle;?(\mathsf{quote});\oplus\{\mathsf{accept} : !\langle \mathsf{credit}\rangle;\mathtt{end}, \mathsf{reject} : \mathtt{end}\}$. We give types to the client processes of Ex. 1:

$$\begin{array}{rcl} \emptyset;\emptyset;y : S & \vdash & \lambda x.\, P_{xy} \triangleright U \multimap \diamond \\ \emptyset;\emptyset;s_1 : !\langle U \multimap \diamond\rangle;\mathtt{end} \cdot s_2 : !\langle U \multimap \diamond\rangle;\mathtt{end} & \vdash & \mathsf{Client}_1 \triangleright \diamond \\ \emptyset;\emptyset;y : !\langle \mathsf{quote}\rangle;?(\mathsf{quote});\mathtt{end} & \vdash & \lambda x.\, Q_i \triangleright U \multimap \diamond \quad (i = 1, 2) \\ \emptyset;\emptyset;s_1 : !\langle U \multimap \diamond\rangle;\mathtt{end} \cdot s_2 : !\langle U \multimap \diamond\rangle;\mathtt{end} & \vdash & \mathsf{Client}_2 \triangleright \diamond \end{array}$$

4 Characteristic Session Bisimulation

We develop a theory for observational equivalence over session typed HOπ processes that follows the principles laid in our previous works [9, 8]. We introduce *characteristic bisimulation* (Def. 15) and prove that it coincides with reduction-closed, barbed congruence (Thm. 18).

We begin by defining an (early) labelled transition system (LTS) on untyped processes (§ 4.1). Then, using the *environmental* transition semantics (§ 4.2), we define a typed LTS to formalise how a typed process interacts with a typed observer.

4.1 Labelled Transition System for Processes

Interaction is defined on action labels ℓ:

$$\ell \;::=\; \tau \;\mid\; n?\langle V\rangle \;\mid\; (\nu\,\widetilde{m})n!\langle V\rangle \;\mid\; n \oplus l \;\mid\; n\&l$$

Label τ defines internal actions. Action $(\nu\,\widetilde{m})n!\langle V\rangle$ denotes the sending of value V over channel n with a possible empty set of restricted names $\widetilde{m}$ (we may write $n!\langle V\rangle$ when $\widetilde{m}$ is empty). Dually, the action for value reception is $n?\langle V\rangle$. Actions for select and branch on a label l are denoted $n \oplus l$ and $n\&l$, resp. We write $\mathtt{fn}(\ell)$ and $\mathtt{bn}(\ell)$ to denote the sets of free/bound names in ℓ, resp. Given $\ell \neq \tau$, we write $\mathtt{subj}(\ell)$ to denote the *subject* of ℓ.

Dual actions occur on subjects that are dual between them and carry the same object; thus, output is dual to input and selection is dual to branching. Formally, duality on actions is the symmetric relation $\asymp$ that satisfies: (i) $n \oplus l \asymp \overline{n}\&l$ and (ii) $(\nu\,\widetilde{m})n!\langle V\rangle \asymp \overline{n}?\langle V\rangle$.

The LTS over *untyped processes* is given in Fig. 3. We write $P_1 \xrightarrow{\ell} P_2$ with the usual meaning. The rules are standard [9, 8]. A process with an output prefix can interact with the environment with an output action that carries a value V (rule $\langle$Snd$\rangle$). Dually, in rule $\langle$Rv$\rangle$ a receiver process can observe an input of an arbitrary value V. Select and branch processes observe the select and branch actions in rules $\langle$Sel$\rangle$ and $\langle$Bra$\rangle$, resp. Rule $\langle$Res$\rangle$ closes the LTS under restriction if the restricted name does not occur free in the observable action. If a restricted name occurs free in the carried value of an output action, the process performs scope opening (rule $\langle$New$\rangle$). Rule $\langle$Rec$\rangle$ handles recursion unfolding. Rule $\langle$Tau$\rangle$ states that two parallel processes which perform dual actions can synchronise by an internal transition. Rules $\langle \text{Par}_L\rangle/\langle \text{Par}_R\rangle$ and $\langle$Alpha$\rangle$ close the LTS under parallel composition and α-renaming.

4.2 Environmental Labelled Transition System

Figure 4 defines a labelled transition relation between a triple of environments, denoted $(\Gamma_1, \Lambda_1, \Delta_1) \xrightarrow{\ell} (\Gamma_2, \Lambda_2, \Delta_2)$. It extends the LTSs in [9, 8] to higher-order sessions. Notice that due to weakening we have $(\Gamma', \Lambda_1, \Delta_1) \xrightarrow{\ell} (\Gamma', \Lambda_2, \Delta_2)$ if $(\Gamma, \Lambda_1, \Delta_1) \xrightarrow{\ell} (\Gamma', \Lambda_2, \Delta_2)$.

$$\langle \textsc{App} \rangle \ \frac{}{(\lambda x.\, P)\, V \xrightarrow{\tau} P\{V/x\}} \qquad \langle \textsc{Snd} \rangle \ \frac{}{n!\langle V\rangle.P \xrightarrow{n!\langle V\rangle} P} \qquad \langle \textsc{Rv} \rangle \ \frac{}{n?(x).P \xrightarrow{n?\langle V\rangle} P\{V/x\}}$$

$$\langle \textsc{Sel} \rangle \ \frac{}{s \triangleleft l.P \xrightarrow{s \oplus l} P} \qquad \langle \textsc{Bra} \rangle \ \frac{}{s \triangleright \{l_i : P_i\}_{i\in I} \xrightarrow{s\& l_j} P_j \ (j \in I)}$$

$$\langle \textsc{Alpha} \rangle \ \frac{P \equiv_\alpha Q \quad Q \xrightarrow{\ell} P'}{P \xrightarrow{\ell} P'} \qquad \langle \textsc{Res} \rangle \ \frac{P \xrightarrow{\ell} P' \quad n \notin \mathtt{fn}(\ell)}{(\nu\, n)P \xrightarrow{\ell} (\nu\, n)P'} \qquad \langle \textsc{New} \rangle \ \frac{P \xrightarrow{(\nu\, \widetilde{m})n!\langle V\rangle} P' \quad m \in \mathtt{fn}(V)}{(\nu\, m)P \xrightarrow{(\nu\, m\cdot\widetilde{m}')n!\langle V\rangle} P'}$$

$$\langle \textsc{Par}_L \rangle \ \frac{P \xrightarrow{\ell} P' \quad \mathtt{bn}(\ell) \cap \mathtt{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\ell} P' \mid Q} \qquad \langle \textsc{Tau} \rangle \ \frac{P \xrightarrow{\ell_1} P' \qquad Q \xrightarrow{\ell_2} Q' \qquad \ell_1 \asymp \ell_2}{P \mid Q \xrightarrow{\tau} (\nu\, \mathtt{bn}(\ell_1) \cup \mathtt{bn}(\ell_2))(P' \mid Q')} \qquad \langle \textsc{Rec} \rangle \ \frac{P\{\mu X.P/X\} \xrightarrow{\ell} P'}{\mu X.P \xrightarrow{\ell} P'}$$

Figure 3 The Untyped LTS for HOπ processes. We omit rule $\langle \textsc{Par}_R \rangle$.

Input Actions. Input Actions are defined by rules [SRv] and [ShRv]. In rule [SRv] the type of value V and the type of the object associated to the session type on s should coincide.

The resulting type tuple must contain the environments associated to V. The dual endpoint $\overline{s}$ cannot be present in the session environment: if it were present the only possible communication would be the interaction between the two endpoints (cf. rule [Tau]). Rule [ShRv] is for shared names and follows similar principles.

Output Actions. Output Actions are defined by rules [SSnd] and [ShSnd]. Rule [SSnd] states the conditions for observing action $(\nu\, \widetilde{m})s!\langle V\rangle$ on a type tuple $(\Gamma, \Lambda, \Delta \cdot s : S)$. The session environment Δ with $s : S$ should include the session environment of the sent value V, *excluding* the session environments of names m_j in $\widetilde{m}$ which restrict the scope of value V. Analogously, the linear variable environment Λ' of V should be included in Λ. Scope extrusion of session names in $\widetilde{m}$ requires that the dual endpoints of $\widetilde{m}$ should appear in the resulting session environment. Similarly for shared names in $\widetilde{m}$ that are extruded. All free values used for typing V are subtracted from the resulting type tuple. The prefix of session s is consumed by the action. Rule [ShSnd] is for output actions on shared names: the name must be typed with $\langle U\rangle$; conditions on V are identical to those on rule [SSnd].

Other Actions. Rules [Sel] and [Bra] describe actions for select and branch. Rule [Tau] defines internal transitions: it keeps the session environment unchanged or reduces it (Def. 2).

▶ **Example 5.** Consider environment $(\Gamma; \emptyset; s : !\langle !\langle S\rangle; \mathtt{end} \multimap \diamond\rangle; \mathtt{end} \cdot s' : S)$ and typed value

$$\Gamma; \emptyset; s' : S \cdot m : ?(\mathtt{end}); \mathtt{end} \vdash V \triangleright\, !\langle S\rangle; \mathtt{end} \multimap \diamond \quad \text{with} \quad V = \lambda x.\, x!\langle s'\rangle.m?(z).\mathbf{0}$$

We illustrate rule [SSnd] in Fig. 4. Let $\Delta_1' = \{\overline{m} : !\langle \mathtt{end}\rangle; \mathtt{end}\}$ and $U = !\langle S\rangle; \mathtt{end} \multimap \diamond$. Then we can derive:

$$(\Gamma; \emptyset; s : !\langle !\langle S\rangle; \mathtt{end} \multimap \diamond\rangle; \mathtt{end} \cdot s' : S) \xrightarrow{(\nu\, m)s!\langle V\rangle} (\Gamma; \emptyset; s : \mathtt{end})$$

Our typed LTS combines the LTSs in Fig. 3 and Fig. 4.

▶ **Definition 6** (Typed Transition System)**.** A *typed transition relation* is a typed relation $\Gamma; \Delta_1 \vdash P_1 \xrightarrow{\ell} \Delta_2 \vdash P_2$ where (1) $P_1 \xrightarrow{\ell} P_2$; (2) $(\Gamma, \emptyset, \Delta_1) \xrightarrow{\ell} (\Gamma, \emptyset, \Delta_2)$ with $\Gamma; \emptyset; \Delta_i \vdash P_i \triangleright \diamond$ $(i = 1, 2)$. We extend to $\Longrightarrow$ and $\xRightarrow{\hat{\ell}}$ where we write $\Longrightarrow$ for the reflexive and transitive closure of $\rightarrow$, $\xRightarrow{\ell}$ for the transitions $\Longrightarrow\xrightarrow{\ell}\Longrightarrow$, and $\xRightarrow{\hat{\ell}}$ for $\xRightarrow{\ell}$ if $\ell \neq \tau$ otherwise $\Longrightarrow$.

$$[\textsc{SRv}]\ \frac{\overline{s} \notin \mathtt{dom}(\Delta) \quad \Gamma;\Lambda';\Delta' \vdash V \triangleright U}{(\Gamma;\Lambda;\Delta \cdot s :?(U);S) \xrightarrow{s?\langle V\rangle} (\Gamma;\Lambda\cdot\Lambda';\Delta\cdot\Delta'\cdot s:S)} \qquad [\textsc{ShRv}]\ \frac{\Gamma;\emptyset;\emptyset \vdash a \triangleright \langle U\rangle \quad \Gamma;\Lambda';\Delta' \vdash V \triangleright U}{(\Gamma;\Lambda;\Delta) \xrightarrow{a?\langle V\rangle} (\Gamma;\Lambda\cdot\Lambda';\Delta\cdot\Delta')}$$

$$[\textsc{SSnd}]\ \frac{\begin{array}{ll}\Gamma\cdot\Gamma';\Lambda';\Delta' \vdash V \triangleright U & \Gamma';\emptyset;\Delta_j \vdash m_j \triangleright U_j \quad \overline{s} \notin \mathtt{dom}(\Delta)\\ \Delta'\backslash \cup_j \Delta_j \subseteq (\Delta\cdot s:S) & \Gamma';\emptyset;\Delta'_j \vdash \overline{m}_j \triangleright U'_j \quad \Lambda' \subseteq \Lambda\end{array}}{(\Gamma;\Lambda;\Delta\cdot s:!\langle U\rangle;S) \xrightarrow{(\nu\,\widetilde{m})s!\langle V\rangle} (\Gamma\cdot\Gamma';\Lambda\backslash\Lambda';(\Delta\cdot s:S\cdot\cup_j\Delta'_j)\backslash\Delta')}$$

$$[\textsc{ShSnd}]\ \frac{\begin{array}{ll}\Gamma\cdot\Gamma';\Lambda';\Delta' \vdash V \triangleright U & \Gamma';\emptyset;\Delta_j \vdash m_j \triangleright U_j \quad \Gamma;\emptyset;\emptyset \vdash a \triangleright \langle U\rangle\\ \Delta'\backslash \cup_j \Delta_j \subseteq \Delta & \Gamma';\emptyset;\Delta'_j \vdash \overline{m}_j \triangleright U'_j \quad \Lambda' \subseteq \Lambda\end{array}}{(\Gamma;\Lambda;\Delta) \xrightarrow{(\nu\,\widetilde{m})a!\langle V\rangle} (\Gamma\cdot\Gamma';\Lambda\backslash\Lambda';(\Delta\cdot\cup_j\Delta'_j)\backslash\Delta')}$$

$$[\textsc{Sel}]\ \frac{\overline{s} \notin \mathtt{dom}(\Delta) \quad j \in I}{(\Gamma;\Lambda;\Delta\cdot s:\oplus\{l_i:S_i\}_{i\in I}) \xrightarrow{s\oplus l_j} (\Gamma;\Lambda;\Delta\cdot s:S_j)}$$

$$[\textsc{Bra}]\ \frac{\overline{s} \notin \mathtt{dom}(\Delta) \quad j \in I}{(\Gamma;\Lambda;\Delta\cdot s:\&\{l_i:T_i\}_{i\in I}) \xrightarrow{s\&l_j} (\Gamma;\Lambda;\Delta\cdot s:S_j)} \qquad [\textsc{Tau}]\ \frac{\Delta_1 \longrightarrow \Delta_2 \vee \Delta_1 = \Delta_2}{(\Gamma;\Lambda;\Delta_1) \xrightarrow{\tau} (\Gamma;\Lambda;\Delta_2)}$$

Figure 4 Labelled Transition System for Typed Environments.

4.3 Reduction-Closed, Barbed Congruence ($\cong$)

We now define *typed relations* and *contextual equivalence* (i.e., barbed congruence). We first define *confluence* over session environments Δ: we denote $\Delta_1 \rightleftharpoons \Delta_2$ if there exists Δ such that $\Delta_1 \longrightarrow^* \Delta$ and $\Delta_2 \longrightarrow^* \Delta$ (here we write $\longrightarrow^*$ for the multi-step reduction in Def. 2).

▶ **Definition 7.** We say that $\Gamma;\emptyset;\Delta_1 \vdash P_1 \triangleright \diamond \;\Re\; \Gamma;\emptyset;\Delta_2 \vdash P_2 \triangleright \diamond$ is a *typed relation* whenever P_1 and P_2 are closed; Δ_1 and Δ_2 are balanced; and $\Delta_1 \rightleftharpoons \Delta_2$. We write $\Gamma;\Delta_1 \vdash P_1 \;\Re\; \Delta_2 \vdash P_2$ for the typed relation $\Gamma;\emptyset;\Delta_1 \vdash P_1 \triangleright \diamond \;\Re\; \Gamma;\emptyset;\Delta_2 \vdash P_2 \triangleright \diamond$.

Typed relations relate only closed terms whose session environments are balanced and confluent. Next we define *barbs* [11] with respect to types.

▶ **Definition 8** (Barbs)**.** Let P be a closed process. We write $P \downarrow_n$ if $P \equiv (\nu\,\tilde{m})(n!\langle V\rangle.P_2 \mid P_3)$, with $n \notin \tilde{m}$. Also: $P \Downarrow_n$ if $P \longrightarrow^* \downarrow_n$. Similarly, we write $\Gamma;\emptyset;\Delta \vdash P \downarrow_n$ if $\Gamma;\emptyset;\Delta \vdash P \triangleright \diamond$ with $P \downarrow_n$ and $\overline{n} \notin \Delta$. Also: $\Gamma;\emptyset;\Delta \vdash P \Downarrow_n$ if $P \longrightarrow^* P'$ and $\Gamma;\emptyset;\Delta' \vdash P' \downarrow_n$.

A barb $\downarrow_n$ is an observable on an output prefix with subject n; a weak barb $\Downarrow_n$ is a barb after a number of reduction steps. Typed barbs $\downarrow_n$ (resp. $\Downarrow_n$) occur on typed processes $\Gamma;\emptyset;\Delta \vdash P \triangleright \diamond$. When n is a session name we require that its dual endpoint $\overline{n}$ is not in Δ.

To define a congruence relation, we introduce the family $\mathbb{C}$ of contexts:

$$\begin{array}{rcl}\mathbb{C} & ::= & - \mid u!\langle V\rangle.\mathbb{C} \mid u?(x).\mathbb{C} \mid u!\langle\lambda x.\mathbb{C}\rangle.P \mid (\nu\,n)\mathbb{C}(\lambda x.\mathbb{C})u \mid \mu X.\mathbb{C}\\ & \mid & \mathbb{C} \mid P \mid P \mid \mathbb{C} \mid u \triangleleft l.\mathbb{C} \mid u \triangleright \{l_1:P_1,\cdots,l_i:\mathbb{C},\cdots,l_n:P_n\}\end{array}$$

Notation $\mathbb{C}[P]$ denotes the result of substituting the hole $-$ in $\mathbb{C}$ with process P.

The first behavioural relation we define is reduction-closed, barbed congruence [4].

▶ **Definition 9** (Reduction-Closed, Barbed Congruence)**.** Typed relation $\Gamma;\Delta_1 \vdash P_1 \;\Re\; \Delta_2 \vdash P_2$ is a *reduction-closed, barbed congruence* whenever:

$$
\begin{array}{rcl@{\qquad}rcl}
(\!|?(U);S|\!)^u & \stackrel{\text{def}}{=} & u?(x).((\!|S|\!)^u \mid (\!|U|\!)^x) & (\!|!\langle U\rangle;S|\!)^u & \stackrel{\text{def}}{=} & u!\langle(\!|U|\!)_{\mathsf{c}}\rangle.(\!|S|\!)^u \\
(\!|\oplus\{l:S\}|\!)^u & \stackrel{\text{def}}{=} & u \triangleleft l.(\!|S|\!)^u & (\!|\&\{l_i:S_i\}_{i\in I}|\!)^u & \stackrel{\text{def}}{=} & u \triangleright \{l_i:(\!|S_i|\!)^u\}_{i\in I} \\
(\!|\mathtt{t}|\!)^u & \stackrel{\text{def}}{=} & X_{\mathtt{t}} & (\!|\mu \mathtt{t}.S|\!)^u & \stackrel{\text{def}}{=} & \mu X_{\mathtt{t}}.(\!|S|\!)^u \\
(\!|\mathtt{end}|\!)^u & \stackrel{\text{def}}{=} & \mathbf{0} & (\!|\langle S\rangle|\!)^u & \stackrel{\text{def}}{=} & u!\langle(\!|S|\!)_{\mathsf{c}}\rangle.\mathbf{0} \\
(\!|\langle L\rangle|\!)^u & \stackrel{\text{def}}{=} & u!\langle(\!|L|\!)_{\mathsf{c}}\rangle.\mathbf{0} & (\!|U{\to}\diamond|\!)^u & \stackrel{\text{def}}{=} & (\!|U{\multimap}\diamond|\!)^u \stackrel{\text{def}}{=} u\,(\!|U|\!)_{\mathsf{c}}
\end{array}
$$

$$
(\!|S|\!)_{\mathsf{c}} \stackrel{\text{def}}{=} s \ (s \text{ fresh}) \qquad (\!|\langle S\rangle|\!)_{\mathsf{c}} \stackrel{\text{def}}{=} (\!|\langle L\rangle|\!)_{\mathsf{c}} \stackrel{\text{def}}{=} a \ (a \text{ fresh}) \qquad (\!|U{\to}\diamond|\!)_{\mathsf{c}} \stackrel{\text{def}}{=} (\!|U{\multimap}\diamond|\!)_{\mathsf{c}} \stackrel{\text{def}}{=} \lambda x.(\!|U|\!)^x
$$

Figure 5 Characteristic Processes (top) and Values (bottom) as in Def. 11. For $(\!|S|\!)_{\mathsf{c}}$, $(\!|\langle S\rangle|\!)_{\mathsf{c}}$, and $(\!|\langle L\rangle|\!)_{\mathsf{c}}$ freshness is assumed with respect to any names in their contexts.

1. If $P_1 \longrightarrow P_1'$ then there exist P_2', Δ_2' such that $P_2 \longrightarrow^* P_2'$ and $\Gamma;\Delta_1' \vdash P_1' \mathrel{\Re} \Delta_2' \vdash P_2'$;
2. If $\Gamma;\Delta_1 \vdash P_1 \downarrow_n$ then $\Gamma;\Delta_2 \vdash P_2 \Downarrow_n$;
3. For all $\mathbb{C}$, Δ_1'', Δ_2'' we have: $\Gamma;\Delta_1'' \vdash \mathbb{C}[P_1] \mathrel{\Re} \Delta_2'' \vdash \mathbb{C}[P_2]$;
4. The symmetric cases of 1 and 2.

The largest such relation is denoted with $\cong$.

4.4 Context Bisimilarity ($\approx$)

Following Sangiorgi [16], we now define the standard (weak) context bisimilarity.

▶ **Definition 10** (Context Bisimilarity)**.** A typed relation $\Re$ is *a context bisimulation* if for all $\Gamma;\Delta_1 \vdash P_1 \mathrel{\Re} \Delta_2 \vdash Q_1$,

1. Whenever $\Gamma;\Delta_1 \vdash P_1 \xrightarrow{(\nu\,\widetilde{m_1})n!\langle V_1\rangle} \Delta_1' \vdash P_2$, there exist Q_2, V_2, Δ_2' such that $\Gamma;\Delta_2 \vdash Q_1 \stackrel{(\nu\,\widetilde{m_2})n!\langle V_2\rangle}{\Longrightarrow} \Delta_2' \vdash Q_2$ and for all R with $\mathtt{fv}(R) = x$:
$$\Gamma;\Delta_1'' \vdash (\nu\,\widetilde{m_1})(P_2 \mid R\{^{V_1}/_x\}) \mathrel{\Re} \Delta_2'' \vdash (\nu\,\widetilde{m_2})(Q_2 \mid R\{^{V_2}/_x\});$$
2. For all $\Gamma;\Delta_1 \vdash P_1 \xrightarrow{\ell} \Delta_1' \vdash P_2$ such that ℓ is not an output, there exist Q_2, Δ_2' such that $\Gamma;\Delta_2 \vdash Q_1 \stackrel{\hat{\ell}}{\Longrightarrow} \Delta_2' \vdash Q_2$ and $\Gamma;\Delta_1' \vdash P_2 \mathrel{\Re} \Delta_2' \vdash Q_2$; and
3. The symmetric cases of 1 and 2.

The largest such bisimulation is called *context bisimilarity* and denoted by $\approx$.

As hinted at in the Introduction, in the general case, context bisimilarity is hard to compute. Below we introduce *characteristic bisimulations*, which are meant to be a *tractable* proof technique over session typed processes with higher-order communication.

4.5 Characteristic Bisimilarity ($\approx^{\mathsf{C}}$)

We formalise the ideas given in the introduction. We define characteristic processes/values:

▶ **Definition 11** (Characteristic Process and Values)**.** Let u and U be a name and a type, respectively. Fig. 5 defines the *characteristic process* $(\!|U|\!)^u$ and the *characteristic value* $(\!|U|\!)_{\mathsf{c}}$.

▶ **Proposition 12.** *Let S be a session type. Then $\Gamma;\emptyset;\Delta \cdot s : S \vdash (\!|S|\!)^s \triangleright \diamond$. Also, let $\langle U\rangle$ be a first-order (channel) type. Then $\Gamma \cdot a : \langle U\rangle;\emptyset;\Delta \vdash (\!|\langle U\rangle|\!)^a \triangleright \diamond$.*

The following example motivates the refined LTS explained in the introduction.

▶ **Example 13** (The Need for Refined Typed LTS)**.** We show that observing a characteristic value input alone is not enough to define a sound bisimulation closure. Consider processes

$$P_1 = s?(x).(x\,s_1 \mid x\,s_2) \qquad P_2 = s?(x).(x\,s_1 \mid s_2?(y).\mathbf{0}) \tag{3}$$

where $\Gamma;\emptyset;\Delta \cdot s :?((?(C);\mathtt{end})\to\diamond);\mathtt{end} \vdash P_i \triangleright \diamond$ $(i \in \{1,2\})$. If P_1 and P_2 input and substitute over x the characteristic value $(\!|(?(C);\mathtt{end})\to\diamond|\!)_{\mathsf{c}} = \lambda x.\, x?(y).\mathbf{0}$, then they evolve into:

$$\Gamma;\emptyset;\Delta \vdash s_1?(y).\mathbf{0} \mid s_2?(y).\mathbf{0} \triangleright \diamond$$

therefore becoming context bisimilar. However, the processes in (3) are clearly *not* context bisimilar: many input actions may be used to distinguish them. For example, if P_1 and P_2 input $\lambda x.\,(\nu\, s)(a!\langle s\rangle.x?(y).\mathbf{0})$ with $\Gamma;\emptyset;\Delta \vdash s \triangleright \mathtt{end}$, then their derivatives are not bisimilar.

Observing only the characteristic value results in an under-discriminating bisimulation. However, if a trigger value $\lambda x.\, t?(y).(y\,x)$ is received on s, we can distinguish P_1, P_2 in (3):

$$P_1 \stackrel{\ell}{\Longrightarrow} t?(x).(x\,s_1) \mid t?(x).(x\,s_2) \text{ and } P_2 \stackrel{\ell}{\Longrightarrow} t?(x).(x\,s_1) \mid s_2?(y).\mathbf{0}$$

with $\ell = s?\langle\lambda x.\, t?(y).(y\,x)\rangle$. One question is whether the trigger value is enough to distinguish two processes (hence no need of characteristic values). This is not the case: the trigger value alone also results in an under-discriminating bisimulation relation. In fact, the trigger value can be observed on any input prefix of *any type*. For example, consider processes

$$(\nu\, s)(n?(x).(x\,s) \mid \overline{s}!\langle\lambda x.\, R_1\rangle.\mathbf{0}) \text{ and } (\nu\, s)(n?(x).(x\,s) \mid \overline{s}!\langle\lambda x.\, R_2\rangle.\mathbf{0}) \tag{4}$$

If these processes input the trigger value, we obtain:

$$(\nu\, s)(t?(x).(x\,s) \mid \overline{s}!\langle\lambda x.\, R_1\rangle.\mathbf{0}) \text{ and } (\nu\, s)(t?(x).(x\,s) \mid \overline{s}!\langle\lambda x.\, R_2\rangle.\mathbf{0})$$

thus we can easily derive a bisimulation closure if we assume a bisimulation definition that allows only trigger value input. But if processes in (4) input the characteristic value $\lambda z.\, z?(x).(x\,m)$, then they would become, under appropriate Γ and Δ:

$$\Gamma;\emptyset;\Delta \vdash (\nu\, s)(s?(x).(x\,m) \mid \overline{s}!\langle\lambda x.\, R_i\rangle.\mathbf{0}) \approx \Delta \vdash R_i\{^m/_x\} \quad (i = 1,2)$$

which are not bisimilar if $R_1\{^m/_x\} \not\approx R_2\{^m/_x\}$.

As explained in the introduction, we define the *refined* typed LTS by considering a transition rule for input in which admitted values are trigger or characteristic values or names:

▶ **Definition 14** (Refined Typed Labelled Transition Relation)**.** We define the environment transition rule for input actions using the input rules in Fig. 4:

$$[\text{RRcv}]\frac{(\Gamma_1;\Lambda_1;\Delta_1) \xrightarrow{n?\langle V\rangle} (\Gamma_2;\Lambda_2;\Delta_2) \quad V = m \vee V \equiv (\!|U|\!)_{\mathsf{c}} \vee V \equiv \lambda x.\, t?(y).(y\,x)\ t \text{ fresh}}{(\Gamma_1;\Lambda_1;\Delta_1) \stackrel{n?\langle V\rangle}{\longmapsto} (\Gamma_2;\Lambda_2;\Delta_2)}$$

Rule [RRcv] is defined on top of rules [Srv] and [ShRv] in Fig. 4. We use the non-receiving rules in Fig. 4 together with rule [RRcv] to define $\Gamma;\Delta_1 \vdash P_1 \stackrel{\ell}{\longmapsto} \Delta_2 \vdash P_2$ as in Def. 6.

Notice that $\Gamma;\Delta_1 \vdash P_1 \stackrel{\ell}{\longmapsto} \Delta_2 \vdash P_2$ (refined transition) implies $\Gamma;\Delta_1 \vdash P_1 \xrightarrow{\ell} \Delta_2 \vdash P_2$ (ordinary transition). Below we sometimes write $\stackrel{(\nu\,\widetilde{m})n!\langle V:U\rangle}{\longmapsto}$ when the type of V is U.

Characteristic Bisimulations. We define *characteristic bisimulations*, a tractable bisimulation for HOπ. As hinted at above, their definition uses trigger processes (cf. (2)):

$$t \Leftarrow V : U \quad \stackrel{\text{def}}{=} \quad t?(x).(\nu\, s)([?(U);\mathsf{end}]^s \mid \overline{s}!\langle V\rangle.\mathbf{0})$$

▶ **Definition 15** (Characteristic Bisimilarity). A typed relation $\Re$ is a *characteristic bisimulation* if for all $\Gamma;\Delta_1 \vdash P_1\ \Re\ \Delta_2 \vdash Q_1$,

1. Whenever $\Gamma;\Delta_1 \vdash P_1 \stackrel{(\nu\,\widetilde{m_1})n!\langle V_1:U\rangle}{\longmapsto} \Delta_1' \vdash P_2$ then there exist Q_2, V_2, Δ_2' such that $\Gamma;\Delta_2 \vdash Q_1 \stackrel{(\nu\,\widetilde{m_2})n!\langle V_2:U\rangle}{\Longmapsto} \Delta_2' \vdash Q_2$ and, for fresh t,
   $\Gamma;\Delta_1'' \vdash (\nu\,\widetilde{m_1})(P_2 \mid t \Leftarrow V_1 : U_1)\ \Re\ \Delta_2'' \vdash (\nu\,\widetilde{m_2})(Q_2 \mid t \Leftarrow V_2 : U_2)$
2. For all $\Gamma;\Delta_1 \vdash P_1 \stackrel{\ell}{\longmapsto} \Delta_1' \vdash P_2$ such that ℓ is not an output, there exist Q_2, Δ_2' such that $\Gamma;\Delta_2 \vdash Q_1 \stackrel{\hat{\ell}}{\Longmapsto} \Delta_2' \vdash Q_2$ and $\Gamma;\Delta_1' \vdash P_2\ \Re\ \Delta_2' \vdash Q_2$; and
3. The symmetric cases of 1 and 2.

The largest such bisimulation is called *characteristic bisimilarity* and denoted by $\approx^{\mathsf{C}}$.

Internal transitions associated to session interactions or β-reductions are deterministic.

▶ **Definition 16** (Deterministic Transition). Let $\Gamma;\emptyset;\Delta \vdash P \triangleright \diamond$ be a balanced HOπ process. Transition $\Gamma;\Delta \vdash P \stackrel{\tau}{\longmapsto} \Delta' \vdash P'$ is called *session transition* whenever the transition $P \stackrel{\tau}{\to} P'$ is derived using rule $\langle$Tau$\rangle$ (where $\mathtt{subj}(\ell_1)$ and $\mathtt{subj}(\ell_2)$ in the premise are dual endpoints), possibly followed by uses of $\langle$Alpha$\rangle$, $\langle$Res$\rangle$, $\langle$Rec$\rangle$, or $\langle \text{Par}_L\rangle/\langle \text{Par}_R\rangle$.

Transition $\Gamma;\Delta \vdash P \stackrel{\tau}{\longmapsto} \Delta' \vdash P'$ is called *β-transition* whenever the transition $P \stackrel{\tau}{\to} P'$ is derived using rule $\langle$App$\rangle$, possibly followed by uses of $\langle$Alpha$\rangle$, $\langle$Res$\rangle$, $\langle$Rec$\rangle$, or $\langle \text{Par}_L\rangle/\langle \text{Par}_R\rangle$. $\Gamma;\Delta \vdash P \stackrel{\tau_{\mathsf{d}}}{\longmapsto} \Delta' \vdash P'$ denotes either a session transition or a β-transition.

▶ **Proposition 17** (τ-inertness). *Let $\Gamma;\emptyset;\Delta \vdash P \triangleright \diamond$ be a balanced HOπ process. Then $\Gamma;\Delta \vdash P \stackrel{\tau_{\mathsf{d}}}{\longmapsto} \Delta' \vdash P'$ implies $\Gamma;\Delta \vdash P \approx^{\mathsf{C}} \Delta' \vdash P'$.*

See [7] for associated proofs. Our main theorem follows: it allows us to use $\approx^{\mathsf{C}}$ as a tractable reasoning technique for higher-order processes with sessions.

▶ **Theorem 18** (Coincidence). *$\cong$, $\approx$, and $\approx^{\mathsf{C}}$ coincide in HOπ.*

Proof (Sketch). We use *higher-order bisimilarity* ($\approx^{\mathsf{H}}$), an auxiliary equivalence that is defined as $\approx^{\mathsf{C}}$ but by using trigger processes with higher-order communication (cf. (1)). We first show that $\approx^{\mathsf{C}}$ and $\approx^{\mathsf{H}}$ coincide by using Prop. 17; then, we show that $\approx^{\mathsf{H}}$ coincides with $\approx$ and $\cong$. A key result is a substitution lemma which simplifies reasoning for $\approx^{\mathsf{H}}$ by exploiting characteristic processes/values. See [7] for full details. ◀

Now we prove that processes Client_1 and Client_2 in Example 1 are behaviourally equivalent.

▶ **Proposition 19.** *Let $S = !\langle \mathsf{room}\rangle;?(\mathsf{quote});\oplus\{\mathsf{accept} : !\langle \mathsf{credit}\rangle;\mathsf{end}, \mathsf{reject} : \mathsf{end}\}$ and $\Delta = s_1 : !\langle S \multimap \diamond\rangle;\mathsf{end} \cdot s_2 : !\langle S \multimap \diamond\rangle;\mathsf{end}$. Then $\emptyset;\Delta \vdash \mathsf{Client}_1 \approx^{\mathsf{C}} \Delta \vdash \mathsf{Client}_2$.*

Proof (Sketch). We show a bisimulation closure by following transitions on each Client. See [7] for details. First, the characteristic process is given as: $[\![?(S \multimap \diamond);\mathsf{end}]\!]^s = s?(x).(x\,k)$. We show that the clients can simulate each other on the first two output transitions, that

also generate the trigger processes:

$$\begin{array}{ll}
\emptyset;\emptyset;\Delta \vdash \mathsf{Client}_1 & \xrightarrow{s_1!\langle \lambda x.\, P_{xy}\{^{h_1}/_{y}\}\rangle}\xrightarrow{s_2!\langle \lambda x.\, P_{xy}\{^{h_2}/_{y}\}\rangle} \\
\emptyset;\emptyset;k_1 : S \cdot k_2 : S \vdash & (\nu\, h_1, h_2)(\overline{h_1}?(x).\overline{h_2}?(y). \\
& \quad \mathtt{if}\ \ x \leq y\ \mathtt{then}\ (\overline{h_1} \triangleleft \mathsf{accept}.\overline{h_2} \triangleleft \mathsf{reject}.\mathbf{0}; \overline{h_1} \triangleleft \mathsf{reject}.\overline{h_2} \triangleleft \mathsf{accept}.\mathbf{0}) \\
& \quad \mid t_1 \Leftarrow \lambda x.\, P_{xy}\{^{h_1}/_{y}\} : S\multimap\diamond \mid t_2 \Leftarrow \lambda x.\, P_{xy}\{^{h_2}/_{y}\} : S\multimap\diamond) \\
\emptyset;\emptyset;\Delta \vdash \mathsf{Client}_2 & \xrightarrow{s_1!\langle \lambda x.\, Q_1\{^{h}/_{y}\}\rangle}\xrightarrow{s_2!\langle \lambda x.\, Q_2\{^{\overline{h}}/_{y}\}\rangle} \\
\emptyset;\emptyset;k_1 : S \cdot k_2 : S \vdash & (\nu\, h)(t_1 \Leftarrow \lambda x.\, Q_1\{^{h}/_{y}\} : S\multimap\diamond \mid t_2 \Leftarrow \lambda x.\, Q_2\{^{\overline{h}}/_{y}\} : S\multimap\diamond)
\end{array}$$

After these transitions, we can analyse that the resulting processes are behaviourally equivalent since they have the same visible transitions; the rest is internal deterministic transitions. ◀

5 Related Work

As in this work, the bisimulations in [9, 8] (binary and multiparty sessions, respectively) are defined and characterised on an LTS which combines an untyped LTS for processes and an LTS on session type environments. The work [14] studies typed equivalences for a theory of binary sessions based on linear logic, without shared names. None of [9, 8, 14] consider session processes with higher-order communication, as we do here. Our results have important consequences in the relative expressivity of higher-order sessions; see [7] for details.

Our approach to typed equivalences builds upon techniques developed by Sangiorgi [15, 16] and Jeffrey and Rathke [5]. As we have discussed, although contextual bisimilarity has a satisfactory discriminative power, its use is hindered by the universal quantification on output. To deal with this, Sangiorgi proposes *normal bisimilarity*, a tractable equivalence without universal quantification. To prove that context and normal bisimilarities coincide, [15] uses triggered processes. Triggered bisimulation is also defined on first-order labels where the context bisimulation is restricted to arbitrary trigger substitution. This characterisation of context bisimilarity was refined in [5] for calculi with recursive types, not addressed in [16, 15] and quite relevant in session-based concurrency. The bisimulation in [5] is based on an LTS extended with trigger meta-notation. As in [16, 15], the LTS in [5] observes first-order triggered values instead of higher-order values, offering a more direct characterisation of contextual equivalence and lifting the restriction to finite types. We briefly contrast the approach in [5] and ours based on characteristic bisimilarity ($\approx^{\mathsf{C}}$):

- The LTS in [5] is enriched with extra labels for triggers; an output action transition emits a trigger and introduces a parallel replicated trigger. Our approach retains usual labels/transitions; in case of output, $\approx^{\mathsf{C}}$ introduces a parallel *non-replicated* trigger.
- Higher-order input in [5] involves the input of a trigger which reduces after substitution. Rather than a trigger name, $\approx^{\mathsf{C}}$ decrees the input of a trigger value $\lambda z.\, t?(x).(x\, z)$.
- Unlike [5], $\approx^{\mathsf{C}}$ treats first- and higher-order values uniformly. As the typed LTS distinguishes linear and shared values, replicated closures are used only for shared values.
- In [5] name matching is crucial to prove completeness of bisimilarity. In our case, HOπ lacks name matching and we use session types: a characteristic value inhabiting a type enables the simplest form of interactions with the environment.

We have compared our approach to that in [5] using a representative example. We considered the transitions and resulting processes involved in checking bisimilarity of process $n!\langle \lambda x.\, x\, (\lambda y.\, y!\langle m\rangle.\mathbf{0})\rangle.\mathbf{0}$ with itself. This comparison, detailed in [7], reveals that our approach requires less visible transitions and replicated processes. Therefore, linearity information does simplify analyses, as it enables simpler witnesses in coinductive proofs.

Environmental bisimulations [17] use a higher-order LTS to define a bisimulation that stores the observer's knowledge; hence, observed actions are based on this knowledge at any given time. This approach is enhanced in [6] with a mapping from constants to higher-order values. This allows to observe first-order values instead of higher-order values. It differs from [16, 5] in that the mapping between higher- and first-order values is no longer implicit.

Acknowledgments. This work has been partially sponsored by the The Doctoral Prize Fellowship, EPSRC EP/K011715/1, EPSRC EP/K034413/1, and EPSRC EP/L00058X/1, EU project FP7-612985 UpScale, and EU COST Action IC1201 BETTY. Pérez is also affiliated to the NOVA Laboratory for Computer Science and Informatics (NOVA LINCS), Universidade Nova de Lisboa, Portugal.

References

1 Giovanni Bernardi, Ornela Dardha, Simon J. Gay, and Dimitrios Kouzapas. On duality relations for session types. In *TGC 2014*, volume 8902 of *LNCS*, pages 51–66. Springer, 2014.
2 Simon J. Gay and Vasco Thudichum Vasconcelos. Linear type theory for asynchronous session types. *J. Funct. Program.*, 20(1):19–50, 2010.
3 Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 22–138, 1998.
4 Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *TCS*, 151(2):437–486, 1995.
5 Alan Jeffrey and Julian Rathke. Contextual equivalence for higher-order pi-calculus revisited. *LMCS*, 1(1), 2005.
6 Vasileios Koutavas and Matthew Hennessy. First-order reasoning for higher-order concurrency. *Computer Languages, Systems & Structures*, 38(3):242–277, 2012.
7 Dimitrios Kouzapas. Full version of this paper, 2015. `http://arxiv.org/abs/1502.02585`.
8 Dimitrios Kouzapas and Nobuko Yoshida. Globally governed session semantics. *LMCS*, 10(4), 2014.
9 Dimitrios Kouzapas, Nobuko Yoshida, Raymond Hu, and Kohei Honda. On asynchronous eventful session semantics. *MSCS*, 2015.
10 Ivan Lanese, Jorge A. Pérez, Davide Sangiorgi, and Alan Schmitt. On the expressiveness and decidability of higher-order process calculi. *Inf. Comput.*, 209(2):198–226, 2011.
11 Robin Milner and Davide Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *19th ICALP*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.
12 Dimitris Mostrous and Nobuko Yoshida. Two session typing systems for higher-order mobile processes. In *TLCA*, volume 4583 of *LNCS*, pages 321–335. Springer, 2007.
13 Dimitris Mostrous and Nobuko Yoshida. Session typing and asynchronous subtyping for the higher-order π-calculus. *Inf. Comput.*, 241:227–263, 2015.
14 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.
15 Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher Order Paradigms*. PhD thesis, University of Edinburgh, 1992.
16 Davide Sangiorgi. Bisimulation for Higher-Order Process Calculi. *Inf. & Comp.*, 131(2):141–178, 1996.
17 Davide Sangiorgi, Naoki Kobayashi, and Eijiro Sumii. Environmental bisimulations for higher-order languages. In *LICS*, pages 293–302. IEEE, 2007.

Multiparty Session Types as Coherence Proofs

Marco Carbone[1], Fabrizio Montesi[2], Carsten Schürmann[1], and Nobuko Yoshida[3]

1 **IT University of Copenhagen, Denmark**
2 **University of Southern Denmark, Denmark**
3 **Imperial College London, UK**

Abstract

We propose a Curry-Howard correspondence between a language for programming multiparty sessions and a generalisation of Classical Linear Logic (CLL). In this framework, propositions correspond to the local behaviour of a participant in a multiparty session type, proofs to processes, and proof normalisation to executing communications. Our key contribution is generalising duality, from CLL, to a new notion of n-ary compatibility, called *coherence*. Building on coherence as a principle of compositionality, we generalise the cut rule of CLL to a new rule for composing many processes communicating in a multiparty session. We prove the soundness of our model by showing the admissibility of our new rule, which entails deadlock-freedom via our correspondence.

1998 ACM Subject Classification F1.1 Models of Computation

Keywords and phrases Programming languages, Type systems, Session Types, Linear Logic

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.412

1 Introduction

Session types are protocols for communications in concurrent systems [13, 22]. A recent line of work investigates Curry-Howard correspondences between the type theory of session types and linear logic, where proofs correspond to processes, propositions to types, and proof normalisation to communications [4, 23]. An important consequence of such correspondences is that several notions that usually require complex additional definitions and proofs, e.g., dependency relations for deadlock-freedom [9, 19], follow for free from the theory of linear logic, yielding a succinct formulation of the formal foundations of sessions.

The aforementioned correspondences cover only session types with exactly two participants, called *binary session types*. In practice, however, protocols often describe the behaviour of multiple participants [21]. *Multiparty Session Types (MPSTs)* have been proposed to capture such protocols, by matching the communications enacted by many participants with a global scenario [14]. Unfortunately, MPSTs are more involved than binary session types, since they include complex analyses on the structure of protocols and a mapping from *global types*, which describe multiparty protocols, to *local types*, which describe the local behaviour of each single participant. So far, it has been unclear whether a succinct logical formulation of MPSTs can be developed, as done for binary session types. Therefore, we ask:

Can we design a proof theory for reasoning about multiparty sessions?

A positive answer to our question would lead to a clearer understanding of the principles that underpin multiparty session programming. The main challenge lies in the foundational notion of duality found in linear logic, which, in a Curry-Howard interpretation of propositions as types, checks whether the session types of two respective participants are compatible. It is an

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 412–426

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

open question how to generalise the notion of type duality to that of "multiparty compatibility" found in MPSTs, which allows to compose an arbitrary number of participants [14, 11, 16]. Therefore, differently from previous work, we are in a situation where the existing logic does not provide us with natural tools for dealing with the types we desire to capture.

The **main contribution** of this work is the development of Multiparty Classical Processes (MCP), a proof theory for reasoning on multiparty communications. The key aspect of MCP is that it generalises Classical Linear Logic (CLL) [12], by building on a new notion of type compatibility, called *coherence*, that replaces duality. Using MCP, we can provide a concise reconstruction of the foundations of MPSTs. In the following, we outline our investigation:

- *Coherence.* We start by formalising a language for *local types* and *global types* (§ 3, Types). As in MPSTs, a local type denotes the I/O actions of a single participant in a session, whereas a global type denotes the desired interactions among all participants in a session. We then present *coherence*, a proof system for determining whether a set of local types follow the scenario denoted by a global type (§ 3, Coherence). We prove the adequacy of coherence by showing that global types are proof terms for coherence proofs (§ 3, Figure 2); equivalences between coherence proofs correspond to the equivalences between global types originally formulated with an auxiliary definition in [6] (§ 3, Proposition 2); and, the coherence proof system yields projection and extraction procedures from global types to local types and vice versa (§ 3, Proposition 3 and Proposition 4). Finally, we show that coherence generalises the notion of duality in CLL (§ 3, Proposition 7). Our extraction procedure is the first not requiring auxiliary conditions (e.g., dependency relations as in [15]) and capturing nested protocols [10].
- *Multiparty Classical Processes.* We present Multiparty Classical Processes (MCP), a proof theory that is in a Curry-Howard correspondence with a language for multiparty sessions (§ 4). The key aspect of MCP is using coherence as a new principle for compositionality in order to generalise the standard cut rule of linear logic, by allowing an arbitrary number of proofs to be composed (§ 4, Figure 6). Such a generalisation allows for the first time to specify cyclic inter-connected networks using (a generalisation of) linear logic whilst preserving its normalisation properties (§ 7). From the proof theory of MCP, we derive logically-founded notions of structural equivalences and reductions for multiparty processes (§ 4, Figure 7 and Figure 8). Driven by the correspondence between processes and proofs, we show that: communications among processes always follow their session types (§ 5, Theorem 10); communications never get stuck (§ 5, Corollary 12), improving on previous techniques for analysing progress with multiparty session types (§ 7); and that protocols used to type processes are always eventually executed (§ 5, Theorem 13).

2 Preview

We give an informal introduction to MCP with the 2-buyer protocol [14], where two buyers buy a book together from a seller. This can be described by the following global type:

$$\begin{array}{ll} 1. & \mathsf{B1} \to \mathsf{S} : \langle \mathbf{str} \rangle;\ \mathsf{S} \to \mathsf{B1} : \langle \mathbf{int} \rangle;\ \mathsf{S} \to \mathsf{B2} : \langle \mathbf{int} \rangle;\ \mathsf{B1} \to \mathsf{B2} : \langle \mathbf{int} \rangle; \\ 2. & \mathsf{B2} \to \mathsf{S} : \&(\mathsf{B2} \to \mathsf{S} : \langle \mathbf{addr} \rangle; \mathsf{end},\quad \mathsf{end}) \end{array} \tag{1}$$

Above, B1 (the first buyer), B2 (the second buyer) and S (the seller) are *roles*. In Line 1, B1 sends the book title to S, then S sends a quote to B1 and B2. At this point, B1 sends to B2 the fraction of the price it wishes to pay. In Line 2, B2 communicates to S whether (&) to proceed with the purchase and, if so, also an address for the delivery.

In multiparty session types, each role in a global type is implemented by a different

process. For example, the following three programs implement the roles in (1):

$$\begin{aligned}
\text{Buyer1} &\stackrel{\text{def}}{=} \quad \overline{x}^{\,\mathsf{B1}\,\mathsf{S}}(\mathit{title});\ x^{\mathsf{B1}\,\mathsf{S}}(\mathit{quote});\ \overline{x}^{\,\mathsf{B1}\,\mathsf{B2}}(\mathit{contr}) \\
\text{Buyer2} &\stackrel{\text{def}}{=} \quad x^{\mathsf{B2}\,\mathsf{S}}(\mathit{quote});\ x^{\mathsf{B2}\,\mathsf{B}_1}(\mathit{contr});\ \big(\ x^{\mathsf{B2}\,\mathsf{S}}.\mathsf{inl};\ \overline{x}^{\,\mathsf{B2}\,\mathsf{S}}(\mathit{addr}) \quad + \quad x^{\mathsf{B2}\,\mathsf{S}}.\mathsf{inr}\ \big) \\
\text{Seller} &\stackrel{\text{def}}{=} \quad x^{\mathsf{S}\,\mathsf{B1}}(\mathit{title});\ \overline{x}^{\,\mathsf{S}\,\mathsf{B1}}(\mathit{quote});\ \overline{x}^{\,\mathsf{S}\,\mathsf{B2}}(\mathit{quote});\ x^{\mathsf{S}\,\mathsf{B2}}\&.\mathsf{case}\big(\ x^{\mathsf{S}\,\mathsf{B2}}(\mathit{addr}),\quad \mathbf{0}\ \big)
\end{aligned}$$

The three processes above are defined in the π-calculus with multiparty sessions [9], and communicate using the session (or channel) x. In term Buyer1, $x^{\mathsf{B1}\,\mathsf{S}}(\mathit{title})$ means "as role B1, send the book *title* over channel x to the process implementing role S"; $\overline{x}^{\,\mathsf{B1}\,\mathsf{S}}(\mathit{quote})$ means "as role B1, receive a quote over channel x from the process implementing role S"; finally, $x^{\mathsf{B1}\,\mathsf{B2}}(\mathit{contr})$ means "as role B1, send to the process implementing role B2, over channel x, the amount the first buyer is willing to contribute with". Note that Buyer2 makes a choice after receiving the contribution from Buyer1, i.e., it either accepts or rejects the purchase by respectively selecting the left or right branch of the case construct in the code of Seller.

Following the approach in [23], we can type channel x using CLL propositions (differently from [23], we use $\parr$ to type outputs and $\otimes$ to type inputs, see § 7):

$$\begin{aligned}
&\text{usage of } x \text{ in Buyer1:} && \mathbf{str} \parr \mathbf{int} \otimes \mathbf{int} \parr \mathsf{end} \\
&\text{usage of } x \text{ in Buyer2:} && \mathbf{int} \otimes \mathbf{int} \otimes \big((\mathbf{addr} \parr \mathsf{end}) \oplus \mathsf{end}\big) \\
&\text{usage of } x \text{ in Seller:} && \mathbf{str} \otimes \mathbf{int} \parr \mathbf{int} \parr \big((\mathbf{addr} \otimes \mathsf{end}) \,\&\, \mathsf{end}\big)
\end{aligned} \tag{2}$$

Above, each proposition states how x is used by each process. For instance, Buyer1 outputs ($\parr$) a string, receives ($\otimes$) an integer, sends another integer and finally terminates (end).

CLL cannot compose our three processes using the above specifications, since its composition rule Cut can only compose two processes, which communicate over a same channel x with compatible binary session types A and $A^{\perp}$:

$$\frac{P \vdash \Delta, x{:}A \qquad Q \vdash \Delta', x{:}A^{\perp}}{(\boldsymbol{\nu} x{:}A)\,(P \mid Q) \vdash \Delta, \Delta'}\ \mathsf{Cut}$$

Using the same channel among our three processes is essential for tracking the dependencies expressed by the global type in (1): for example, we need to ensure that Seller sends a quote to Buyer2 only after it has received a request for a book from Buyer1. Such constraints cannot be tracked by binary session types [14]. To overcome this issue, we annotate each connective in propositions with roles. For example, the type of x for Buyer1 would become:

$$\begin{aligned}
&\text{annotated usage of } x \text{ in Buyer1:} && \mathbf{str} \parr^{\mathsf{S}} \mathbf{int} \otimes^{\mathsf{S}} \mathbf{int} \parr^{\mathsf{B2}} \mathsf{end} \\
&\text{annotated usage of } x \text{ in Buyer2:} && \mathbf{int} \otimes^{\mathsf{S}} \mathbf{int} \otimes^{\mathsf{B1}} \big((\mathbf{addr} \parr^{\mathsf{S}} \mathsf{end}) \oplus^{\mathsf{S}} \mathsf{end}\big) \\
&\text{annotated usage of } x \text{ in Seller:} && \mathbf{str} \otimes^{\mathsf{B1}} \mathbf{int} \parr^{\mathsf{B1}} \mathbf{int} \parr^{\mathsf{B2}} \big((\mathbf{addr} \otimes^{\mathsf{B2}} \mathsf{end}) \,\&^{\mathsf{B2}}\, \mathsf{end}\big)
\end{aligned} \tag{3}$$

Annotations identify the dual role for each action, e.g., the usage for Buyer1 now reads: send a string to S ($\parr^{\mathsf{S}}$); receive an integer from S ($\otimes^{\mathsf{S}}$); send an integer to B2 ($\parr^{\mathsf{B2}}$); and, terminate (end). We can then reformulate rule Cut as:

$$\frac{P_i \vdash \Gamma_i, x^{p_i}{:}A_i \qquad G \vDash \{p_i{:}A_i\}_i}{(\boldsymbol{\nu} x{:}G)\,\big(\textstyle\prod_i P_i\big) \vdash \{\Gamma_i\}_i}\ \mathsf{MCut}$$

In our new *multiparty* cut rule MCut, if some processes P_i use session x as role p_i (denoted x^{p_i}), each according to some respective types A_i, and such types coherently follow a global type specification G (formalised by the judgement $G \vDash \{p_i{:}A_i\}_i$), then we can compose them in parallel within the scope of session x, written $(\boldsymbol{\nu} x{:}G)\,(P_1 \mid \ldots \mid P_n)$. In our example,

$$
\begin{array}{rcll}
A, B, \ldots & ::= & 1 \quad \textit{(unit for } \otimes\textit{)} & \mid \quad \bot \quad \textit{(unit for } \parr\textit{)} \\
& \mid & A \parr^{\tilde{p}} B \quad \textit{(send } A \textit{ to } \tilde{p}\textit{, then } B\textit{)} & \mid \quad A \otimes^{p} B \quad \textit{(receive } A \textit{ from } p\textit{, then } B\textit{)} \\
& \mid & A \oplus^{\tilde{p}} B \quad \textit{(select } A \textit{ or } B \textit{ in } \tilde{p}\textit{)} & \mid \quad A \& ^{p} B \quad \textit{(offer } A \textit{ or } B \textit{ to } p\textit{)} \\
& \mid & !A \quad \textit{(client request)} & \mid \quad ?A \quad \textit{(server accept)}
\end{array}
$$

$$
G \quad ::= \quad p \to \tilde{q} : \langle G' \rangle ; G \quad \mid \quad p \to \tilde{q} : \&(G_1, G_2) \quad \mid \quad ?p \to !\tilde{q} : \langle G \rangle \quad \mid \quad \mathsf{end}^{p\tilde{q}}
$$

Figure 1 Local Types ($A, B, \ldots$) and Global Types (G).

for i ranging from 1 to 3, $\{p_i : A_i\}_i$ would correspond to the types in (3), where p_1, p_2 and p_3 would be, respectively, Buyer1, Buyer2 and Seller. In § 6, we will show that such types coherently follow the global type given in (1).

MCP goes beyond the original multiparty session types [14], capturing also multicasting and nested protocols [9, 10]. For example, we can enhance the 2-buyer protocol as:

$$
\begin{array}{ll}
1. & \mathsf{B1} \to \mathsf{S} : \langle \mathbf{str} \rangle ;\ \mathsf{S} \to \mathsf{B1}, \mathsf{B2} : \langle \mathbf{int} \rangle ;\ \mathsf{B1} \to \mathsf{B2} : \langle \mathbf{int} \rangle ; \\
2. & \mathsf{B2} \to \mathsf{B1}, \mathsf{S} : \&\Big(\mathsf{B2} \to \mathsf{S} : \langle \mathbf{addr} \rangle ;\ \mathsf{end}, \quad \mathsf{B1} \to \mathsf{S} : \langle G_{\mathsf{sub}} \rangle ; \mathsf{B1} \to \mathsf{S} : \langle \mathbf{str} \rangle ; \mathsf{B2} \to \mathsf{S} : \langle \mathbf{str} \rangle ; \mathsf{end} \Big)
\end{array}
\tag{4}
$$

Above, S multicasts the price to both B1 and B2; and B2 multicasts its decision to B1 and S. We have also updated the right branch of the choice using a nested protocol G_{sub}, which is private to B1 and S, where B1 tells S whether it wants to purchase the product alone:

$$
G_{\mathsf{sub}} \quad = \quad \mathsf{B1} \to \mathsf{S} : \&\Big(\mathsf{B1} \to \mathsf{S} : \langle \mathbf{addr} \rangle ;\ \mathsf{end}, \quad \mathsf{B1} \to \mathsf{S} : \langle \mathbf{str} \rangle ; \mathsf{end} \Big)
$$

In MCP, nested protocols can proceed in parallel to their originating protocols. For example, the last two communications, where B1 and B2 inform S of their respective reasons for not completing the purchase, can be executed in parallel to G_{sub}. We will formalise this in § 5.

3 Coherence

We give a proof-theoretical reconstruction of coherence, from [14]. Our theory generalises duality, from CLL, to checking the compatibility of multiple types. We define coherence as a proof system for deriving sets of (compatible) *local types*, which describe the local behaviours of participants in a multiparty session. Global types are proof terms for coherence proofs, yielding a correspondence between sets of compatible local types and their global descriptions.

Types. The syntax of local and global types is given in Figure 1, where p, q range over a set of *roles*. Global types are highlighted, to distinguish them as proof terms. Highlighting is also used in our syntax of local types, to show the difference with CLL. We will adopt the same convention in § 4 when we present more terms.

A local type A describes the local behaviour of a role in a session. Types 1 and $\bot$ denote session termination, respectively representing the request and the acceptance for closing a session (which were informally abstracted by end in our previous examples). A type $A \parr^{\tilde{p}} B$ denotes a multicast output of a session with type A to roles $\tilde{p}$, with a continuation B. A type $A \otimes^{p} B$ represents an input of a session with type A from role p, with continuation B. Types $A \oplus^{\tilde{p}} B$ and $A \&^{p} B$ denote, respectively, the output of a choice between the continuations A and B to roles $\tilde{p}$ and the input of a choice from role p. The replicated type $!A$ offers behaviour A as many times as requested. Finally, type $?A$ requests the execution of a replicated type and proceeds as A.

$$\frac{G \vDash \Theta,\ p{:}B,\ \{q_i{:}D_i\}_i \qquad G' \vDash p{:}A,\ \{q_i{:}C_i\}_i}{p \to \tilde{q} : \langle G' \rangle ; G \vDash \Theta,\ p{:}A \parr^{\tilde{q}} B,\ \{q_i{:}C_i \otimes^{p} D_i\}_i}\ \otimes\parr \qquad \frac{}{\mathsf{end}^{p\tilde{q}} \vDash p{:}\bot, q_1{:}1, \ldots, q_n{:}1}\ 1\bot$$

$$\frac{G_1 \vDash \Theta, p{:}A, \{q_i{:}C_i\}_i \qquad G_2 \vDash \Theta, p{:}B, \{q_i{:}D_i\}_i}{p \to \tilde{q} : \&(G_1, G_2) \vDash \Theta, p{:}A \oplus^{\tilde{q}} B, \{q_i{:}C_i \&^{p} D_i\}_i}\ \oplus\& \qquad \frac{G \vDash p{:}A, \{q_i{:}B_i\}_i}{?p \to !\tilde{q} : \langle G \rangle \vDash p{:}?A, \{q_i{:}!B_i\}_i}\ !?$$

Figure 2 Coherence.

A global type G describes the behaviour of many participants. In $p \to \tilde{q} : \langle G' \rangle ; G$, role p sends to roles $\tilde{q}$ a message to create a new session of type G', and then the protocol proceeds as G. In $p \to \tilde{q} : \&(G_1, G_2)$, role p communicates to roles $\tilde{q}$ its choice of either branch G_1 or G_2. A type $?p \to !\tilde{q} : \langle G \rangle$ denotes that role p may ask roles $\tilde{q}$ to execute G many times. Finally, in $\mathsf{end}^{p\tilde{q}}$, role p asks roles $\tilde{q}$ to terminate the session (for brevity, we often write end).

Judgements. A *role typing* $p{:}A$ states that role p behaves as specified by type A. Our *judgements* for coherence have the form $G \vDash p_1{:}A_1, \ldots, p_n{:}A_n$ which reads as "the types $A_1, \ldots, A_n$ of the respective roles $p_1, \ldots, p_n$ are compatible and follow the global type G". We use Θ to range over sets of role typings, and make the standard assumption that we can write $\Theta, p{:}A$ only if a role typing for p does not appear in Θ. Given some roles $\tilde{p}$, we use the notation $\{p_i{:}A_i\}_i$ to denote the set of role typings $p_1{:}A_1, \ldots, p_n{:}A_n$, assuming $\tilde{p} = p_1, \ldots, p_n$ and i ranging from 1 to n. Given G, we say that G is *valid* if there exists Θ such that $G \vDash \Theta$. Conversely, given Θ, we say that Θ is *coherent* if there exists G such that $G \vDash \Theta$.

We report the rules for deriving coherence judgements in Figure 2.

Rule $\otimes\parr$ matches the output type from role p to roles $\tilde{q}$ with the input types of roles $\tilde{q}$, whenever (i) the types for the newly created session are coherent and (ii) the types of all continuations are also coherent. Rule $\oplus\&$ checks that both possibilities in a choice are coherent, where all roles participating in the communication are allowed to have different behaviour and the other roles are not (a multicast generalisation of [14]). In rule !?, we check that a client requests the creation of a coherent session only from replicated services. Finally, rule $1\bot$ checks that all participants agree on the termination of a protocol. As in CLL, we interpret type 1 as a terminated process and $\bot$ as a process that has terminated its behaviour in a session and proceeds with other sessions. Therefore, we read rule $1\bot$ as "a protocol terminates when one participant waits (type $\bot$) for the termination of all the others (type 1), which execute in parallel". This design choice simplifies our development; we discuss a generalisation in § 7.

▶ **Example 1** (2-Buyer Protocol). We can revisit the local types for the 2-buyer protocol in § 2 (1), where now data types are abstracted by 1's and $\bot$'s.

$$A \stackrel{\text{def}}{=} \bot \parr^{\mathsf{S}} 1 \otimes^{\mathsf{S}} \bot \parr^{\mathsf{B2}} \bot \qquad B \stackrel{\text{def}}{=} 1 \otimes^{\mathsf{S}} 1 \otimes^{\mathsf{B1}} \left((\bot \parr^{\mathsf{S}} 1) \oplus^{\mathsf{S}} 1 \right) \qquad C \stackrel{\text{def}}{=} 1 \otimes^{\mathsf{B1}} \bot \parr^{\mathsf{B1}} \bot \parr^{\mathsf{B2}} \left((1 \otimes^{\mathsf{B2}} 1) \ \&^{\mathsf{B2}} \ 1 \right)$$

Let G be the global type in (1) with end instead of data types; then, $G \vDash \mathsf{B1}{:}A, \mathsf{B2}{:}B, \mathsf{S}{:}C$.

3.1 Properties of Coherence

Swapping. Immediately from our correspondence between global types and coherence proofs, we can reconstruct the standard notion of swapping $\simeq^{\mathsf{g}}$ for global types from [6]. Intuitively, two communications involving different roles can always be swapped, capturing

$$\frac{\{p,\tilde{q}\} \cap \{r,\tilde{s}\} = \emptyset}{p \to \tilde{q} : \langle G \rangle; r \to \tilde{s} : \langle G' \rangle; G'' \simeq^{\mathsf{g}} r \to \tilde{s} : \langle G' \rangle; p \to \tilde{q} : \langle G \rangle; G''} \ (\to\to)$$

$$\frac{\{p,\tilde{q}\} \cap \{r,\tilde{s}\} = \emptyset}{p \to \tilde{q} : \&(r \to \tilde{s} : \langle G \rangle; G_1, r \to \tilde{s} : \langle G \rangle; G_2) \simeq^{\mathsf{g}} r \to \tilde{s} : \langle G \rangle; p \to \tilde{q} : \&(G_1, G_2)} \ (\to\oplus)$$

$$\frac{\{p,\tilde{q}\} \cap \{r,\tilde{s}\} = \emptyset}{p \to \tilde{q} : \&(r \to \tilde{s} : \&(G_1, G_2), r \to \tilde{s} : \&(G_3, G_4)) \simeq^{\mathsf{g}} r \to \tilde{s} : \&(p \to \tilde{q} : \&(G_1, G_3), p \to \tilde{q} : \&(G_2, G_4))} \ (\oplus\oplus)$$

Figure 3 Swapping relation $\simeq^{\mathsf{g}}$ for global types.

the fact that separate roles execute concurrently. For example, the following coherence proof (for p, q, r, s different):

$$\frac{\dfrac{G \vDash \Theta, p{:}A', q{:}B', r{:}C', s{:}D' \quad G'' \vDash \tilde{\Theta}, r{:}C, s{:}D}{r \to s : \langle G'' \rangle; G \vDash \Theta, p{:}A', q{:}B', r{:}C \parr^{s} C', s{:}D \otimes^{r} D'} \otimes\parr \qquad G' \vDash p{:}A, q{:}B}{p \to q : \langle G' \rangle; r \to s : \langle G'' \rangle; G \vDash \Theta,\ p{:}A \parr^{q} A',\ q{:}B \otimes^{p} B',\ r{:}C \parr^{s} C',\ s{:}D \parr^{r} D',} \otimes\parr$$

is equivalent to $(\simeq^{\mathsf{g}})$

$$\frac{\dfrac{G \vDash \Theta, p{:}A', q{:}B', r{:}C', s{:}D' \quad G' \vDash \tilde{\Theta}, p{:}A, q{:}B}{p \to q : \langle G' \rangle; G \vDash \Theta, p{:}A \parr^{q} A', q{:}B \otimes^{p} B', r{:}C', s{:}D'} \otimes\parr \qquad G'' \vDash r{:}C, s{:}D}{r \to s : \langle G'' \rangle; p \to q : \langle G' \rangle; G \vDash \Theta,\ p{:}A \parr^{q} A',\ q{:}B \otimes^{p} B',\ r{:}C \parr^{s} C',\ s{:}D \parr^{r} D'} \otimes\parr$$

proving that $p \to q : \langle G' \rangle; r \to s : \langle G'' \rangle; G$ is equivalent to $r \to s : \langle G'' \rangle; p \to q : \langle G' \rangle; G$. Figure 3 reports all cases for $\equiv$, derived from the proof system in Figure 2.

In general, two global types are proof terms for the same set of local typings if and only if they are equivalent.

▶ **Proposition 2** (Swapping). *Let $G \vDash \Theta$. Then, $G \simeq^{\mathsf{g}} G'$ if and only if $G' \vDash \Theta$.*

Projection and Extraction. The hallmark of the theory of multiparty session types is projection: developers can write protocols as global types, and then automatically project a global type onto a set of local types that can be used to modularly verify the behaviour of each participant. As there is only one possible rule application for each production in the syntax of global types, we can construct an algorithm that traverses the structure of G:

▶ **Proposition 3** (Projection). *For G valid, Θ such that $G \vDash \Theta$ is computable in linear time.*

We can also use coherence for the inverse procedure, i.e., the extraction of a global type from a set of local typings Θ. If Θ is coherent, we can just apply the first applicable coherence rule, noting that the sizes of the local types in the premises always get smaller:

▶ **Proposition 4** (Extraction). *For Θ coherent, G such that $G \vDash \Theta$ is computable.*

▶ **Example 5.** In the 2-buyer protocol, $G \vDash \mathsf{B1}{:}A, \mathsf{B2}{:}B, \mathsf{S}{:}C$ implies: (i) we can infer A, B and C from G (proposition 3) and (ii) we can extract G from $\mathsf{B1}{:}A, \mathsf{B2}{:}B, \mathsf{S}{:}C$ (proposition 4).

Global reductions. We define reductions for global types, denoted $\tilde{G} \rightsquigarrow \tilde{G}'$, where $\tilde{G}$ is a set $\{G_1, \ldots, G_n\}$. *Global type reductions are just a convention* (recalling [6]), which we use in § 5 to concisely formalise how processes follow their protocols. Formally, $\rightsquigarrow$ is the smallest relation satisfying the rules in Figure 4.

Rule $\mathsf{g}_{\otimes\parr}$ models a communication that creates a new session of type G', which will then proceed in parallel to the continuation G. Rule $\mathsf{g}_{1\bot}$ models session termination. Rules

$$(\mathsf{g}_{\otimes\parr}) \quad \{\, p \to \tilde{q} : \langle G' \rangle; G \,\} \rightsquigarrow \{\, G, G' \,\} \qquad (\mathsf{g}_{!C}) \quad \{\, ?p \to !\tilde{q} : \langle G \rangle \,\} \rightsquigarrow \{\, G, ?p \to !\tilde{q} : \langle G \rangle \,\}$$
$$(\mathsf{g}_{\oplus\&1}) \quad \{\, p \to \tilde{q} : \&(G_1, G_2) \,\} \rightsquigarrow \{\, G_1 \,\} \qquad (\mathsf{g}_{\oplus\&2}) \quad \{\, p \to \tilde{q} : \&(G_1, G_2) \,\} \rightsquigarrow \{\, G_2 \,\}$$
$$(\mathsf{g}_{!?}) \quad \{\, ?p \to !\tilde{q} : \langle G \rangle \,\} \rightsquigarrow \{\, G \,\} \qquad (\mathsf{g}_{!W}) \quad \{\, ?p \to !\tilde{q} : \langle G \rangle \,\} \rightsquigarrow \emptyset \qquad (\mathsf{g}_{1\bot}) \quad \{\, \mathsf{end}^{p\tilde{q}} \,\} \rightsquigarrow \emptyset$$
$$(\mathsf{g}_{\mathsf{ctx}}) \quad \tilde{G}_1 \rightsquigarrow \tilde{G}_2 \;\Rightarrow\; \tilde{G} \cup \tilde{G}_1 \rightsquigarrow \tilde{G} \cup \tilde{G}_2 \qquad (\mathsf{g}_{\mathsf{eq}}) \quad \tilde{G}_0 \simeq^{\mathsf{g}} \tilde{G}_1,\ \tilde{G}_1 \rightsquigarrow \tilde{G}_2,\ \tilde{G}_2 \simeq^{\mathsf{g}} \tilde{G}_3 \;\Rightarrow\; \tilde{G}_0 \rightsquigarrow \tilde{G}_3$$

Figure 4 Global Types, reduction semantics.

$\mathsf{g}_{\oplus\&1}$ and $\mathsf{g}_{\oplus\&2}$ model the execution of a choice. In rules $\mathsf{g}_{!?}$, $\mathsf{g}_{!C}$ and $\mathsf{g}_{!W}$, a replicated protocol can be respectively executed exactly once, multiple, or zero times. Rule $\mathsf{g}_{\mathsf{ctx}}$ lifts the behaviour of a protocol to a set of protocols executing concurrently. Finally, rule $\mathsf{g}_{\mathsf{swap}}$ allows for swappings in a global type, where $\tilde{G} \simeq^{\mathsf{g}} \tilde{G}'$ is the point-wise extension of the swapping relation $\simeq^{\mathsf{g}}$ to sets. Our semantics preserves validity:

▶ **Theorem 6** (Coherence Preservation). *If $\tilde{G}$ is valid and $\tilde{G} \rightsquigarrow \tilde{G}'$, then $\tilde{G}'$ is valid.*

▶ Remark. Rule $\mathsf{g}_{!?}$ can be derived from rules $\mathsf{g}_{!C}$ and $\mathsf{g}_{!W}$. Including it simplifies our presentation, since each global type reduction corresponds to a communication in MCP (§ 5).

Coherence as generalised duality. Coherence is a generalisation of duality (from CLL [12]): in the degenerate case of a session with two participants, the two notions coincide. We recall the definition of duality $X^\bot$, defined inductively on the syntax of linear logic propositions:

$$(X \otimes Y)^\bot = X^\bot \parr Y^\bot \qquad (X \parr Y)^\bot = X^\bot \otimes Y^\bot \qquad 1^\bot = \bot \qquad \bot^\bot = 1$$
$$(X \oplus Y)^\bot = X^\bot \& Y^\bot \qquad (X \& Y)^\bot = X^\bot \oplus Y^\bot \qquad (!X)^\bot = ?X^\bot \qquad (?X)^\bot = !X^\bot$$

We define a partial encoding $[\![\cdot]\!]$ from local types into linear logic propositions:

$$[\![1]\!] = 1 \quad [\![\bot]\!] = \bot \qquad [\![!A]\!] = !\,[\![A]\!] \quad [\![?A]\!] = ?\,[\![A]\!] \quad [\![A \otimes B]\!] = [\![A]\!] \otimes [\![B]\!]$$
$$[\![A \parr^q B]\!] = [\![A]\!] \parr [\![B]\!] \quad [\![A \oplus^p B]\!] = [\![A]\!] \oplus [\![B]\!] \qquad [\![A \&^p B]\!] = [\![A]\!] \& [\![B]\!]$$

The encoding $[\![\cdot]\!]$ is defined only when $\parr$ and $\oplus$ are annotated with a single role. We get:

▶ **Proposition 7** (Coherence as Duality). *Let A, B be propositions where all subterms of the form $C \parr^{\tilde{p}} D$ or $C \oplus^{\tilde{p}} D$ are such that $\tilde{p} = q$ for some q. Then, $G \vDash p{:}A, q{:}B$ iff $[\![A]\!] = [\![B]\!]^\bot$.*

4 Multiparty Classical Processes

In this section, we present Multiparty Classical Processes (MCP). MCP captures dependencies among actions performed by different participants in a multiparty session, whereas, in previous work, actions among different pairs of participants must be independent [4, 23].

Environments. Let Γ, Δ range over typing environments: $\Gamma, \Delta \;::=\; \cdot \;\mid\; \Gamma, x^p{:}A$. Intuitively, $x^p{:}A$ means that role p in session x follows behaviour A. We write $?\Delta$ whenever Δ contains only types of the form $?A$, and write $\Delta, x^p{:}A$ only when x^p does not appear in Δ.

Processes. We report the syntax of processes in Figure 5.

In MCP, both input and output names are bound, as in [23]. Term *(send)* creates a new session y and sends it, as role p, to the processes respectively playing roles $\tilde{q}$ in session x; then, the process proceeds as P. The dual operation *(recv)* receives, as role p in session x, a fresh session y from the process playing role q; the process then proceeds as the parallel

$$
\begin{array}{rcll}
P, Q, R & ::= & \overline{x}^{pq}(y); P & \textit{(send)} \\
& \mid & x^{pq}.\mathsf{inl}; P & \textit{(left sel)} \\
& \mid & x^{pq}.\mathsf{case}(P, Q) & \textit{(case)} \\
& \mid & \mathsf{close}\, x^p & \textit{(close)} \\
& \mid & !x^p(y); P & \textit{(service)} \\
& \mid & P + Q & \textit{(choice)} \\
& \mid & x^{pq}(y); (P \mid Q) & \textit{(recv)} \\
& \mid & x^{pq}.\mathsf{inr}; P & \textit{(right sel)} \\
& \mid & (\boldsymbol{\nu} x{:}G)\left(\prod_i P_i\right) & \textit{(res)} \\
& \mid & \mathsf{wait}\, x^p; P & \textit{(wait)} \\
& \mid & ?x^p(y); P & \textit{(client)}
\end{array}
$$

Figure 5 MCP, syntax of processes.

$$
\frac{P \vdash \Gamma, y^p{:}A \quad Q \vdash \Delta, x^p{:}B}{x^{pq}(y); (P \mid Q) \vdash \Gamma, \Delta, x^p{:}A \otimes^q B}\ \otimes
\qquad
\frac{P \vdash \Gamma, y^p{:}A, x^p{:}B}{\overline{x}^{p\tilde{q}}(y); P \vdash \Gamma, x^p{:}A \parr^{\tilde{q}} B}\ \parr
$$

$$
\frac{P_i \vdash \Gamma_i, x^{p_i}{:}A_i \quad G \vDash \{p_i{:}A_i\}_i}{(\boldsymbol{\nu} x{:}G)\left(\prod_i P_i\right) \vdash \{\Gamma_i\}_i}\ \mathsf{MCut}
\qquad
\frac{P \vdash \Gamma, x^p{:}A \quad Q \vdash \Gamma, x^p{:}B}{x^{pq}.\mathsf{case}(P, Q) \vdash \Gamma, x^p{:}A \,\&^q\, B}\ \&
$$

$$
\frac{P \vdash \Gamma \quad Q \vdash \Gamma}{P + Q \vdash \Gamma}\ +
\qquad
\frac{P \vdash \Gamma, x^p{:}A}{x^{p\tilde{q}}.\mathsf{inl}; P \vdash \Gamma, x^p{:}A \oplus^{\tilde{q}} B}\ \oplus_1
\qquad
\frac{P \vdash \Gamma, x^p{:}B}{x^{p\tilde{q}}.\mathsf{inr}; P \vdash \Gamma, x^p{:}A \oplus^{\tilde{q}} B}\ \oplus_2
$$

$$
\frac{}{\mathsf{close}\, x^p \vdash x^p{:}1}\ 1
\qquad
\frac{P \vdash \Gamma}{\mathsf{wait}\, x^p; P \vdash \Gamma, x^p{:}\bot}\ \bot
\qquad
\frac{P \vdash \Gamma}{P \vdash \Gamma, x^p{:}\ ?A}\ \mathsf{Weaken}
$$

$$
\frac{P \vdash ?\Gamma, y^p{:}A}{!x^p(y); P \vdash ?\Gamma, x^p{:}\ !A}\ !
\qquad
\frac{P \vdash\ \Gamma, y^p{:}A}{?x^p(y); P \vdash\ \Gamma, x^p{:}\ ?A}\ ?
\qquad
\frac{P \vdash\ \Gamma, y^p{:}\ ?A, z^p{:}\ ?A}{P[x/y][x/z] \vdash\ \Gamma, x^p{:}\ ?A}\ \mathsf{Contract}
$$

Figure 6 MCP, typing rules.

composition of P (dedicated to session y) and Q (dedicated to continuing session x). Similarly, terms *(left sel)* and *(right sel)* multicast a selection of a left or right branch respectively to the processes playing roles $\tilde{q}$ in session x, as role p. A selection is received by term *(case)*, which offers the two selectable branches. Terms *(close)* and *(wait)* terminate a session. Term *(choice)* is the standard non-deterministic choice. In a restriction *(res)*, x is bound in the processes P_i; we use the standard type annotation (as in [23]) to show the relation between the semantics of processes and global types in § 5. In term $x^{p\,q}(y); (P \mid Q)$, y is bound in P but not in Q. In terms $\overline{x}^{\,p\,\tilde{q}}(y)P$, $!x^p(y); P$, and $?x^p(y); P$, y is bound in P.

Judgements. Judgements in MCP have the form $P \vdash x_1^{p_1}{:}A_1, \ldots, x_n^{p_n}{:}A_n$, meaning that process P implements roles p_i in the respective session x_i with behaviour A_i.

Rules. We report the rules of MCP in Figure 6. Intuitively, a process is typed with local types; then, we use coherence to check that the local types of composed processes (rule MCut) coherently implement a global type. All rules are defined up to context exchange.

Rule MCut is central: it extends the Cut of CLL to composing in parallel an arbitrary number of P_i that communicate using session x. The rule checks that the composition of the respective local behaviours of the composed processes is coherent ($G \vDash \{p_i{:}A_i\}_i$). In the conclusion, $\{\Gamma_i\}_i$ is the disjoint union of all Γ_i in the premise.

Rule $\otimes$ types an input $x^{pq}(y); (P \mid Q)$, where the subprocess P plays role p with behaviour A in the received multiparty session y; session x then proceeds by following behaviour B for role p in Q. Observe that the $\otimes$ is annotated with the role q that p wishes

to receive from. The multicast output $\overline{x}^{p\tilde{q}}(y); P$ in rule $\parr$ creates a new session y and sends it, as role p in session x, to roles $\tilde{q}$. The new session y is used by P as role p with type A, assuming that the other processes receiving it implement the other roles (this assumption is checked by coherence in MCut, when processes are composed). We discuss in § 7 how to relax the constraint that the role p played in session y is the same.

Rules $\oplus_1$ and $\oplus_2$ type, respectively, the multicast of a left and right selection, by checking that the process continuation follows the expected local type. Similarly, rule & types a branching by checking that the continuations implement the respective expected local types.

Rule + types the nondeterministic process $P + Q$, by checking that both P and Q implement the same local behaviours. Observe that P and Q may still be substantially different, since they may (i) perform different selections on some sessions (as rules $\oplus_1$ and $\oplus_2$ can yield the same typing), and (ii) have different inner compositions of processes whose types have been hidden by rule MCut.

Rules 1 and $\bot$ type, respectively, the request and the acceptance for closing a multiparty session. Rules ! and ? type, respectively, the replicated offering of a service and its repeated usage (a client). Since a service typed by ! may be used multiple times, we require that its continuation does not use any linear behaviour ($?\Delta$). Rules Weaken and Contract type, respectively, the absence of clients or the presence of multiple clients. In rule Contract, sessions y and z are contracted into a single session x with a standard name substitution, provided that they have the same type $?A$.

▶ Remark. Removing proof terms from MCP yields a pure logic that differs from CLL only for rule MCut. Since we will prove in § 5 that MCut is admissible, just like Cut in CLL, the two systems subsume the same set of valid judgements. Nevertheless, as shown in next section, MCP yields a different operational meaning: reductions of MCut correspond to multiparty communications, whereas reductions of Cut in CLL correspond to binary communications.

5 Semantics

In this section, we demonstrate the consistency of MCP, by establishing a cut-elimination result that yields an operational semantics and important properties, e.g., deadlock-freedom.

5.1 Structural Equivalences as Commuting Conversions

MCP supports *commuting conversions*, permutations of applications of MCut that maintain the validity of judgements. As an example, consider the following proof equivalence ($\equiv$):

$$\dfrac{\dfrac{P \vdash \Delta, y^p{:}A, x^p{:}B, z^r{:}C}{\overline{x}^{p\tilde{q}}(y); P \vdash \Delta, x^p{:}A \parr^{\tilde{q}} B, z^r{:}C}\,\parr \qquad Q_i \vdash \Gamma_i, z^{s_i}{:}D_i \qquad G \vDash r{:}C, \{s_i{:}D_i\}_i}{(\boldsymbol{\nu} z{:}G)\left(\overline{x}^{p\tilde{q}}(y); P \mid \prod_i Q_i\right) \vdash \{\Gamma_i\}_i, \Delta, x^p{:}A \parr^{\tilde{q}} B}\,\mathsf{MCut}$$

$$\equiv$$

$$\dfrac{P \vdash \Delta, y^p{:}A, x^p{:}B, z^r{:}C \qquad \dfrac{Q_i \vdash \Gamma_i, z^{s_i}{:}D_i \qquad G \vDash r{:}C, \{s_i{:}D_i\}_i}{(\boldsymbol{\nu} z{:}G)\left(P \mid \prod_i Q_i\right) \vdash \{\Gamma_i\}_i, \Delta, y^p{:}A, x^p{:}B}\,\mathsf{MCut}}{\overline{x}^{p\tilde{q}}(y); (\boldsymbol{\nu} z{:}G)\left(P \mid \prod_i Q_i\right) \vdash \{\Gamma_i\}_i, \Delta, x^p{:}A \parr^{\tilde{q}} B}\,\parr$$

Above, an output is moved out of a restriction of a different session (or in it, reading in the other direction), as in [23]. In this example, the output process is the first in the parallel under the restriction; in general, this is not always the case since the process may be any of those in the parallel composition. In order to represent equivalences independently of the position of processes in a parallel, we use process contexts [20]. A context, denoted by $\mathcal{C}$, is a parallel composition with a hole: $\mathcal{C}[\cdot] ::= \cdot \mid \mathcal{C}[\cdot] \mid P \mid P \mid \mathcal{C}[\cdot]$. All equivalences are reported in Figure 7.

$$
\begin{array}{lrcll}
(\kappa_{\mathsf{par}}) & (\boldsymbol{\nu} z{:}G)\left(\prod_{i\in\tilde{k}} P_i\right) & \equiv & (\boldsymbol{\nu} z{:}G)\left(\prod_{j\in\tilde{k'}} P_j\right) & (\tilde{k} \text{ is a permutation of } \tilde{k'}) \\
(\kappa_{\mathsf{cut}}) & (\boldsymbol{\nu} x{:}G)\left(\mathcal{C}\left[(\boldsymbol{\nu} y{:}G')\,\mathcal{C}'[P]\right]\right) & \equiv & (\boldsymbol{\nu} y{:}G')\left(\mathcal{C}'\left[(\boldsymbol{\nu} x{:}G)\,\mathcal{C}[P]\right]\right) & (\text{if } x, y \in \mathsf{fn}(P)) \\
(\kappa_{\parr}) & (\boldsymbol{\nu} z{:}G)\left(\mathcal{C}\left[\overline{x}^{pq}(y); P\right]\right) & \equiv & \overline{x}^{pq}(y); (\boldsymbol{\nu} z{:}G)\,\mathcal{C}[P] & \\
(\kappa_{\otimes}) & (\boldsymbol{\nu} z{:}G)\left(\mathcal{C}\left[x^{pq}(y); (P \mid Q)\right]\right) & \equiv & x^{pq}(y); (P \mid (\boldsymbol{\nu} z{:}G)\,(\mathcal{C}[Q])) & (\text{if } z \notin \mathsf{fn}(P)) \\
(\kappa_{\oplus 1}) & (\boldsymbol{\nu} z{:}G)\,\mathcal{C}[x^{pq}.\mathsf{inl}; P] \equiv x^{pq}.\mathsf{inl}; (\boldsymbol{\nu} z{:}G)\,\mathcal{C}[P] & & (\kappa_{\oplus 2}) & (\boldsymbol{\nu} z{:}G)\,\mathcal{C}[x^{pq}.\mathsf{inr}; P] \equiv x^{pq}.\mathsf{inr}; (\boldsymbol{\nu} z{:}G)\,\mathcal{C}[P] \\
(\kappa_{\&}) & (\boldsymbol{\nu} z{:}G)\,\mathcal{C}[x^{pq}.\mathsf{case}(P, Q)] & \equiv & x^{pq}.\mathsf{case}((\boldsymbol{\nu} z{:}G)\,\mathcal{C}[P], (\boldsymbol{\nu} z{:}G)\,\mathcal{C}[Q]) & \\
(\kappa_{!}) & (\boldsymbol{\nu} z{:}G)\,\mathcal{C}[!x^{p}(y); P] \equiv !x^{p}(y); (\boldsymbol{\nu} z{:}G)\,\mathcal{C}[P] & & (\kappa_{?}) & (\boldsymbol{\nu} z{:}G)\,\mathcal{C}[?x^{p}(y); P] \equiv ?x^{p}(y); (\boldsymbol{\nu} z{:}G)\,\mathcal{C}[P] \\
(\kappa_{\bot}) & (\boldsymbol{\nu} z{:}G)\,\mathcal{C}[\mathsf{wait}\, x^{p}; P] & \equiv & \mathsf{wait}\, x^{p}; (\boldsymbol{\nu} z{:}G)\,\mathcal{C}[P] &
\end{array}
$$

Figure 7 MCP, Structural Equivalences.

$$
\begin{array}{ll}
(\beta_{\otimes\parr}) & (\boldsymbol{\nu} x{:}p\text{->}\tilde{q} : \langle G'\rangle;G)\left(\prod_i x^{q_i\,p}(y); (P_i \mid Q_i) \mid \overline{x}^{p\tilde{q}}(y); R \mid \prod_j P_j\right) \\
& \quad\to\quad (\boldsymbol{\nu} y{:}G')\left(\prod_i P_i \mid (\boldsymbol{\nu} x{:}G)\,(\prod_i Q_i \mid R \mid \prod_j P_j)\right) \\
(\beta_{\oplus\&1}) & (\boldsymbol{\nu} x{:}p\text{->}\tilde{q} : \&(G_1, G_2))\left(x^{p\tilde{q}}.\mathsf{inl}; P \mid \prod_i x^{q_i p}.\mathsf{case}(Q_i, R_i) \mid \prod_j P_j\right) \to (\boldsymbol{\nu} x{:}G_1)\left(P \mid \prod_i Q_i \mid \prod_j P_j\right) \\
(\beta_{\oplus\&2}) & (\boldsymbol{\nu} x{:}p\text{->}\tilde{q} : \&(G_1, G_2))\left(x^{p\tilde{q}}.\mathsf{inr}; P \mid \prod_i x^{q_i p}.\mathsf{case}(Q_i, R_i) \mid \prod_j P_j\right) \to (\boldsymbol{\nu} x{:}G_2)\left(P \mid \prod_i R_i \mid \prod_j P_j\right) \\
(\beta_{!?}) & (\boldsymbol{\nu} x{:}?p\text{->}!\tilde{q} : \langle G\rangle)\left(?x^{p}(y); P \mid \prod_i !x^{q_i}(y); Q_i\right) \to (\boldsymbol{\nu} y{:}G)\left(P \mid \prod_i Q_i\right) \\
(\beta_{!W}) & (\boldsymbol{\nu} x{:}?p\text{->}!\tilde{q} : \langle G\rangle)\left(\prod_i !x^{q_i}(y); Q_i \mid P\right) \to P \qquad \text{if } x \notin \mathsf{fn}(P) \\
(\beta_{!C}) & (\boldsymbol{\nu} x{:}?p\text{->}!\tilde{q} : \langle G\rangle)\left(\prod_i !x^{q_i}(w); Q_i \mid P[x/y][x/z]\right) \\
& \quad\to\quad (\boldsymbol{\nu} y{:}?p\text{->}!\tilde{q} : \langle G\rangle)\left(\prod_i !y^{q_i}(w); Q_i \mid (\boldsymbol{\nu} z{:}?p\text{->}!\tilde{q} : \langle G\rangle)\left(\prod_i !z^{q_i}(w); Q_i \mid P\right)\right) \\
(\beta_{1\bot}) & (\boldsymbol{\nu} x{:}\mathsf{end}^{p\tilde{q}})\left(\mathsf{wait}\, x^{p}; P \mid \prod_i \mathsf{close}\, x^{q_i}\right) \to P \\
(\beta_{+}) & (\boldsymbol{\nu} x{:}G)\left((P_1 + P_2) \mid \prod_i Q_i\right) \to (\boldsymbol{\nu} x{:}G)\left(P_j \mid \prod_i Q_i\right) \qquad j \in \{1, 2\}
\end{array}
$$

Figure 8 MCP, Cut Reductions.

The equivalence κ_{par} permutes processes in a parallel, since the premises of rule MCut can be in any order. In κ_{cut}, we can swap two restrictions, which corresponds to swapping two applications of rule MCut. The equivalence $\kappa_{\parr}$ shows that a restriction can always be swapped with an output on a different session. Similarly, the equivalence $\kappa_{\otimes}$ swaps a restriction with an input, requiring that the restricted name (z in this case) occurs free in P. In the case of $\oplus$, we have two equivalences, corresponding to the right and left selection respectively. For $\kappa_{\&}$, we can move a restriction to each branch of a case construct, also duplicating the context $\mathcal{C}$. Equivalences $\kappa_{!}$ and $\kappa_{?}$ allow to swap a restriction with a service and a client respectively. Finally, $\kappa_{\bot}$ is the case for $\mathsf{wait}\, x^{p}$. There is no equivalence for the process $\mathsf{close}\, x^{p}$ since it is only typable with the axiom 1.

5.2 Process Reductions as MCut Reductions

As for equivalences, we use our proof theory to derive reductions for processes, given in Figure 8.

In the reduction $\beta_{\otimes\parr}$, the output from role p to roles $\tilde{q}$ on session x is matched with the inputs at such roles, creating a new session y, following the global type of x. Reductions $\beta_{\oplus\&1}$ and $\beta_{\oplus\&2}$ capture the left and right multicast selection of a branching, respectively. In $\beta_{!?}$, a set of services with a single client is reduced to the composition of the bodies of such

services with that of the client; the type $?p \texttt{->} !\tilde{q} : \langle G \rangle$ of x is correspondingly reduced to G. Reduction $\beta_{!W}$ garbage collects a set of unused services. In $\beta_{!C}$, instead, a set of services is replicated to handle multiple clients. Finally, reduction $\beta_{1\perp}$ terminates a session x.

5.3 Properties

In the remainder, we abuse the notation $P \to P'$ to refer to process reductions closed up to our structural equivalence $\equiv$, as in standard process calculi.

Processes and Types. Since both equivalences and reductions are derived from judgement-preserving proof transformations, we immediately obtain the following two properties:

▶ **Theorem 8** (Subject Congruence). *$P \vdash \Delta$ and $P \equiv Q$ imply that $Q \vdash \Delta$.*

▶ **Theorem 9** (Subject Reduction). *$P \vdash \Delta$ and $P \to Q$ imply that $Q \vdash \Delta$.*

In Figure 8, global type annotations should not be mistaken for a requirement of our reductions; they are rather a guarantee given by our proof theory: if a process is reducible, then its sessions are surely typed with the respective global types reported in the rule. We use this property to reconstruct the result of session fidelity from multiparty session types [14]. In the following, $\mathsf{gt}(P)$ denotes the set of global types used in the restrictions inside P.

▶ **Theorem 10** (Session Fidelity). *$P \vdash \Delta$ and $P \to P'$ imply that either $\mathsf{gt}(P) \rightsquigarrow \mathsf{gt}(P')$ or $\mathsf{gt}(P) \simeq^{\mathsf{g}} \mathsf{gt}(P')$.*

Deadlock Freedom. Processes in MCP are guaranteed to be deadlock-free. We use the standard methodology from [4, 23]. First, we prove that the MCut rule in MCP is admissible:

▶ **Theorem 11** (MCut Admissibility). *$P_i \vdash \Gamma_i, x^{p_i} : A_i$, for $i \in [1, n]$, and $G \vDash \{p_i : A_i\}_i$ imply that there exists Q such that $Q \vdash \{\Gamma_i\}_i$.*

The admissibility of MCut gives us a methodology for removing cuts from a proof, corresponding to executing communications in a process until all restrictions are eliminated:

▶ **Corollary 12** (Deadlock-Freedom). *$P \vdash \Delta$ and P has a restriction imply $P \to Q$ for some Q.*

Protocol Progress. Our correspondence between process and global type reductions goes both ways, a novel result for Multiparty Session Types. Below, $\to^+$ denotes one or more applications of $\to$:

▶ **Theorem 13** (Protocol Progress). *$P \vdash \Delta$ and $\mathsf{gt}(P) \rightsquigarrow \tilde{G}$ imply $P \to^+ P'$ and $\tilde{G} = \mathsf{gt}(P')$.*

6 The 2-Buyer Protocol Example

We now formalise the 2-buyer protocol from § 2 and expand it further.

Processes and Types. Roles B1, B2 and S are implemented as the processes:

$$\overline{x}^{\mathsf{B1\,S}}(title);\ \mathsf{wait}\ title^{\mathsf{B1}};\ x^{\mathsf{B1\,S}}(quote);\Big(\mathsf{close}\ quote^{\mathsf{B1}}\ \mid\ \overline{x}^{\mathsf{B1\,B2}}(contrib);\mathsf{wait}\ contrib^{\mathsf{B1}};\mathsf{wait}\ x^{\mathsf{B1}};\mathsf{close}\ z^{\mathsf{Z}}\Big)$$
$$x^{\mathsf{B2\,S}}(quote);\ \Big(\mathsf{close}\ quote^{\mathsf{B2}}\ \mid\ x^{\mathsf{B2\,B1}}(contrib);\ \big(\mathsf{close}\ contrib^{\mathsf{B2}}\ \mid\ P_{B2}\big)\Big)$$
$$x^{\mathsf{S\,B1}}(title);\ \big(\mathsf{close}\ title^{\mathsf{S}}\ \mid\ \overline{x}^{\mathsf{S\,B1}}(quote);\ \mathsf{wait}\ quote^{\mathsf{S}};\ \overline{x}^{\mathsf{S\,B2}}(quote);\ \mathsf{wait}\ quote^{\mathsf{S}};\ \ P_S\big)\Big)$$

The first process is the first buyer Buyer1. In the second process, the second buyer Buyer2, subterm P_{B2} implements the choice of whether to accept or reject the purchase:

$$\left(x^{\mathsf{B2\,S}}.\mathsf{inl};\ \overline{x}^{\,\mathsf{B2\,S}}(addr);\ \mathsf{wait}\ addr^{\mathsf{S}};\ \mathsf{close}\ x^{\mathsf{B2}}\right)\ +\ \left(x^{\mathsf{B2\,S}}.\mathsf{inr};\mathsf{close}\ x^{\mathsf{B2}}\right)$$

Finally, in the third process, the implementation of the seller, P_S is the process:

$$x^{\mathsf{S\,B2}}.\mathsf{case}\ (\ x^{\mathsf{S\,B2}}(addr);\ \left(\mathsf{close}\ addr^{\mathsf{S}}\ \mid\ \mathsf{close}\ x^{\mathsf{S}}\right),\qquad \mathsf{close}\ x^{\mathsf{S}}\ )$$

At the level of types, the local types in Example 1 from § 3 can be used to type the three processes above: Buyer1 $\vdash x^{\mathsf{B1}}{:}A, z^{\mathsf{Z}}{:}1$, Buyer2 $\vdash x^{\mathsf{B2}}{:}B$ and Seller $\vdash x^{\mathsf{S}}{:}C$. If we apply our new cut rule, we obtain $(\boldsymbol{\nu} x : G)\left(\text{Buyer1}\ \mid\ \text{Buyer2}\ \mid\ \text{Seller}\right) \vdash z^{\mathsf{Z}}{:}1$ where the global type G, corresponding to equation (1) in § 2, is such that $G\ \vDash\ \mathsf{B1}{:}A,\ \mathsf{B2}{:}B,\ \mathsf{S}{:}C$.

Nested Multiparty Sessions. We can extend the example above by implementing the global type (4) in § 2, where the first buyer creates a sub-session with the seller if the second buyer decides not to contribute to the purchase. Below, we give an excerpt of the new seller:

$$\ldots\overline{x}^{\mathsf{S\,B1,B2}}(quote);\mathsf{wait}\ quote^{\mathsf{S}};x^{\mathsf{S\,B2}}.\mathsf{case}\left(\ldots,x^{\mathsf{S\,B1}}(y);\left(P_{\mathsf{sub}}\mid x^{\mathsf{S\,B1}}(why);\left(\mathsf{close}\ why^{\mathsf{S}}\mid x^{\mathsf{S\,B2}}(why);\ldots\right)\right)\right)$$

where $P_{\mathsf{sub}} = y^{\mathsf{S\,B1}}.\mathsf{case}\left(y^{\mathsf{S\,B1}}(addr);(\mathsf{close}\ addr^{\mathsf{S}}\ \mid\ \mathsf{close}\ y^{\mathsf{S}}),\quad y^{\mathsf{S\,B1}}(why);(\mathsf{close}\ why^{\mathsf{S}}\ \mid\ \mathsf{close}\ y^{\mathsf{S}})\right)$. Hence, the type of channel x, from the seller's viewpoint, becomes:

$$1\otimes^{\mathsf{B1}}\bot\ \invamp^{\mathsf{B1,B2}}\left((1\otimes^{\mathsf{B2}}1)\ \ \&^{\mathsf{B2}}\ \ \left(\left((1\otimes^{\mathsf{B1}}1)\ \ \&^{\mathsf{B1}}\ \ (1\otimes^{\mathsf{B1}}1)\right)\otimes^{\mathsf{B1}}1\otimes^{\mathsf{B1}}1\otimes^{\mathsf{B2}}1\right)\right)$$

We can then use coherence to infer the global type (4) in § 2.

Services. We extend the example to support multiple clients on a replicated session a:

$$(\boldsymbol{\nu} a{:}?\mathsf{B1}\ \texttt{->}\ !\mathsf{B2},\mathsf{S}:\langle G\rangle)\ (\ \text{Buyers}\ \mid\ !a^{\mathsf{B2}}(x);\text{Buyer2}\ \mid\ !a^{\mathsf{S}}(x);\text{Seller}\ )$$

where Buyers consists of two buyers: $(\boldsymbol{\nu} z:\mathsf{end})\ (\ ?a^{\mathsf{B1}}(x);\text{Buyer1}\ \mid\ ?a^{\mathsf{B1}}(x);\text{Buyer1}')$. Process Buyer1′ initially behaves as Buyer1, but we replace $\mathsf{close}\ z^{\mathsf{Z}}$ with $\mathsf{wait}\ z^{\mathsf{Z}};\mathsf{close}\ w^{\mathsf{W}}$. By applying $\beta_{!C}$ once, $\beta_{!?}$ twice, and commuting conversions, the process above can be reduced to the parallel composition of two sessions that follow the 2-buyer protocol:

$$(\boldsymbol{\nu} x{:}G)\left(\text{Buyer2}\ \mid\ \text{Seller}\ \mid\ (\boldsymbol{\nu} z:\mathsf{end})\left(\text{Buyer1}\ \mid\ (\boldsymbol{\nu} x{:}G)\,(\text{Buyer2}\ \mid\ \text{Seller}\ \mid\ \text{Buyer1'})\right)\right)$$

7 Related Work and Discussion

Curry-Howard correspondences for session types. The works closest to ours are the Curry-Howard correspondences between binary session types and linear logic [4, 23]. We extended this line of work considerably by introducing multiparty sessions, which required generalising the notion of type compatibility in linear logic to address multiple types (coherence). Coherence reconstructs the standard relationship between the global and local views found in multiparty session types. We then used coherence to develop a new proof theory that conservatively extends linear logic to capture multiparty interactions (all derivable judgements in linear logic are derivable also in our framework, and vice versa). Furthermore, our work provides, for the first time, a notion of session fidelity in the context of a Curry-Howard

correspondence between linear logic and session types (§ 5, Theorem 10). In this work we have not treated polymorphism and existential/universal quantification, which we believe can be naturally added to MCP following the lines presented in [23, 3] for binary sessions.

Our work inverts the interpretation of $\otimes$ as output and $\parr$ as input given in [2]. This makes our process terms in line with previous developments of multiparty session types, where communications go from one sender to many receivers [9]. Using the standard interpretation would yield a join mechanism where multiple senders synchronise with a single receiver; note that there would be no need to re-prove our results, since the proof theory would not change.

The standard cut rule in CLL forces the graph of connections among processes to be a tree [1], a known sufficient condition for deadlock-freedom in session types [5]. A multi-cut rule is proposed in [1] to allow two processes to share multiple channels. This enables reasoning on networks with cyclic inter-connections, but breaks the deadlock-freedom property guaranteed by linear logic, since duality is no longer a sufficient condition when multiple resources are involved (also noted in [23]). For the first time, MCP processes can have cyclic inter-connections (e.g., our example in § 2), but they are still guaranteed to be deadlock-free. The key twist is to use coherence as a principle to check that the inter-connections are safely resolved by communications. This suggests that coherence may be useful also in other settings related to linear logic, for reasoning about the sharing of resources among multiple entities (in our case, sessions). We leave this investigation as interesting future work.

Multiparty Session Types (MPSTs). Our work concisely unifies many of the ideas found in separate developments of multiparty session types. Our global types with multicasting are inspired from [9], to which we added nested and replicated types; both additions arise naturally from our proof theory. Our nesting of global types can be seen as a logical reconstruction of (a simplification of) those originally presented in [10], while repetitions in global types reconstruct the concept presented in [8].

Our proof system for coherence is inspired by the notion of well-formedness found in MPSTs [14, 9]. Since coherence is a proof system, projection and extraction are derived from proof equivalences, rather than being defined separately as in [14, 15]. A benefit is that our projection and extraction are guaranteed to be correct by construction, whereas in previous works they have to be proven correct separately wrt the auxiliary notion of well-formedness.

In [9], MPSTs are combined with an ordering on session names to guarantee deadlock-freedom. Our deadlock-freedom result, instead, is based on the structure of our proofs. In some cases, our technique is more precise; for example, consider the deadlock-free system:

$$?a^{p}(x_a);\ ?b^{r}(x_b);\ \overline{x_a}^{\,pq}(w_1);\ {x_b}^{rs}(w_2);\ (\ \overline{x_a}^{\,pt}(w_3);\ P_1\ \mid\ P_2\ )$$
$$!a^{q}(x_a);\ {x_a}^{qp}(w_1);\ (P_3\ \mid\ P_4) \qquad !a^{t}(x_a);\ {x_a}^{tp}(w_3);\ (P_5\ \mid\ P_6) \qquad !b^{s}(x_b);\ \overline{x_b}^{\,sr}(w_2);\ P_7$$

If we compose these processes in parallel, restricting sessions a and b accordingly, we obtain a typable MCP process. Instead, the system in [9] rejects it, since the actions performed by the first process create a cycle between the names x_a and x_b. In [19], the approach in [9] is refined to type processes such as the one above by ordering the I/O actions of each session.

We conjecture that MCP can be used to naturally extend the work in [7], where linear logic is used to type choreography programs, obtaining a Curry-Howard correspondence for the calculus of compositional choreographies typed with multiparty session types [18].

Coherence. Coherence can be generalised, e.g., in Figure 2: (i) rule !? could allow for more than one client; (ii) similarly, rule $1\bot$ could be relaxed to allow for more than one $\bot$ type; (iii) rule $\otimes\parr$ could allow the involved participants to play different roles in the nested session they

create, as in [10] (adding such roles as an extra annotation to each type respectively). We leave these extensions as interesting future work. Point (ii) influences greatly the complexity of the cut admissibility proof for MCP (Theorem 11), because it would imply that the cut reduction of a terminated session could lead to having more than one process in the reductum (all the processes typed with $\bot$), whereas now we have only one. This means that we would have to type a parallel composition of processes without restriction, requiring to extend our framework in the fashion of the logic presented in [7]. While extending the proof theory of MCP would be easy, (extending coherence to allow for missing participants to be added later, as in [18]), it would also cause an explosion in the number of cases to consider in the proof [7]. As future work, we will investigate how our rule MCut and the notion of coherence can affect the mapping from the functional language GV [23, 17].

Acknowledgements. Montesi was supported by CRC, grant no. DFF–4005-00304 from the Danish Council for Independent Research. Schürmann was partly supported by DemTech, grant no. 10-092309 from the Danish Council for Strategic Research. Yoshida was partially sponsored by the EPSRC EP/K011715/1, EP/K034413/1, EP/L00058X/1, and EU project FP7-612985 UpScale. This work is also supported by the COST Action IC1201 BETTY.

References

1 Samson Abramsky, Simon J. Gay, and Rajagopal Nagarajan. Interaction categories and the foundations of typed concurrent programming. In *NATO ASI DPD*, pages 35–113, 1996.

2 Gianluigi Bellin and Philip J. Scott. On the pi-calculus and linear logic. *Theor. Comput. Sci.*, 135(1):11–65, 1994.

3 Luís Caires, Jorge A. Pérez, Frank Pfenning, and Bernardo Toninho. Behavioral polymorphism and parametricity in session-based communication. In *ESOP*, pages 330–349, 2013.

4 Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *CONCUR*, pages 222–236, 2010.

5 Marco Carbone and Søren Debois. A graphical approach to progress for structured communication in web services. In *Proc. of ICE'10*, 2010.

6 Marco Carbone and Fabrizio Montesi. Deadlock-freedom-by-design: multiparty asynchronous global programming. In *POPL*, pages 263–274, 2013.

7 Marco Carbone, Fabrizio Montesi, and Carsten Schürmann. Choreographies, logically. In *CONCUR*, pages 47–62, 2014.

8 Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani. On global types and multi-party session. *LMCS*, 8(1), 2012.

9 Mario Coppo, Mariangiola Dezani-Ciancaglini, Nobuko Yoshida, and Luca Padovani. Global progress for dynamically interleaved multiparty sessions. *MSCS*, 760:1–65, 2015.

10 Romain Demangeon and Kohei Honda. Nested protocols in session types. In *CONCUR*, pages 272–286, 2012.

11 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty compatibility in communicating automata: Characterisation and synthesis of global session types. In *ICALP (2)*, pages 174–186, 2013.

12 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

13 Kohei Honda, Vasco Vasconcelos, and Makoto Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP*, pages 22–138, 1998.

14 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proc. of POPL*, volume 43(1), pages 273–284. ACM, 2008.

15 Julien Lange and Emilio Tuosto. Synthesising choreographies from local session types. In *CONCUR*, pages 225–239, 2012.

16 Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From communicating machines to graphical choreographies. In *POPL 2015*, pages 221–232. ACM, 2015.

17 Sam Lindley and J. Garrett Morris. A semantics for propositions as sessions. In *ESOP*, pages 560–584, 2015.

18 Fabrizio Montesi and Nobuko Yoshida. Compositional choreographies. In *CONCUR*, pages 425–439, 2013.

19 Luca Padovani, Vasco Thudichum Vasconcelos, and Hugo Torres Vieira. Typing liveness in multiparty communicating systems. In *COORDINATION*, pages 147–162, 2014.

20 D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

21 Scribble project home page. `http://www.scribble.org`.

22 Vasco T. Vasconcelos. Fundamentals of session types. *Inf. Comput.*, 217:52–70, 2012.

23 Philip Wadler. Propositions as sessions. In *ICFP*, pages 273–286, 2012.

On Coinduction and Quantum Lambda Calculi*

Yuxin Deng[1], Yuan Feng[2], and Ugo Dal Lago[3]

1 **Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, China**
2 **University of Technology Sydney, Australia**
3 **Università di Bologna, Italy, and INRIA, France**

Abstract

In the ubiquitous presence of linear resources in quantum computation, program equivalence in linear contexts, where programs are used or executed once, is more important than in the classical setting. We introduce a linear contextual equivalence and two notions of bisimilarity, a state-based and a distribution-based, as proof techniques for reasoning about higher-order quantum programs. Both notions of bisimilarity are sound with respect to the linear contextual equivalence, but only the distribution-based one turns out to be complete. The completeness proof relies on a characterisation of the bisimilarity as a testing equivalence.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages

Keywords and phrases Quantum lambda calculi, contextual equivalence, bisimulation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.427

1 Introduction

Since two decades ago, the theory of quantum computing has attracted considerable research efforts. Benefiting from the superposition of quantum states, quantum computing may provide remarkable speedup over its classical analogue [32, 16, 17]. As a consequence, a wealth of models and programming languages for describing quantum computation have been introduced.

For many reasons, the functional paradigm fits very well in the picture. One successful attempt in this direction is QUIPPER [15], an expressive functional higher-order language that can be used to program a diverse set of non-trivial quantum algorithms and can generate quantum gate representations using trillions of gates. The group led by Svore introduced LIQUi$|\rangle$ as a modular software architecture designed to control quantum hardware [34]: it enables easy programming, compilation, and simulation of quantum algorithms and circuits. In spite of the success of language design, the semantic foundation of quantum programming languages is not well established. In a series of papers, Selinger and co-authors try to find a denotational semantics for higher-order quantum computation [28, 29, 30, 31]. In the most recent one [24], they propose a denotational model that is adequate with respect to an operational semantics. However, full abstraction for a higher-order language with both classical and quantum resources still remains an open problem.

In quantum mechanics, a fundamental principle is the no-cloning theorem of quantum resources. From a type-theoretic point of view, quantum resources are linear and can be

* Partially supported by the National Natural Science Foundation of China (61173033, 61261130589, 61428208) and ANR 12IS02001 PACE. The second author is also supported by Australian Research Council under grant No. DP130102764 and the CAS/SAFEA International Partnership Program for Creative Research Team.

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 427–440

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

described by linear types in quantum programming languages. How to define appropriate program equivalences for this kind of languages is an interesting problem. Some preliminary results towards this direction have been obtained by omitting quantum effects and only considering nondeterminism and linearity in a functional language [8]. For that restricted setting, a notion of *linear contextual equivalence* is introduced and shown to be nicely related to trace equivalence. Linear contextual equivalence is a special form of contextual equivalence [23] in which the observable behaviour of programs is tested by executing them (at most) once.

In the current work we investigate the operational semantics of the typed quantum λ-calculus proposed in [24]. We aim to develop coinductive proof techniques for linear contextual equivalence (written $\simeq$) of quantum programs. We first define a labelled transition system for the quantum λ-calculus. It is in fact a probabilistic labelled transition system (pLTS) because probability distributions arise naturally when quantum systems are measured. In the underlying pLTS, we consider two notions of probabilistic bisimilarity: one (written $\sim_s$) is state-based because it is directly defined over states and then lifted to distributions; the other ($\sim_d$) is distribution-based as it is a relation between distributions. The relation $\sim_s$ is essentially the probabilistic bisimilarity originally defined by Larsen and Skou [22], representing a branching time semantics. In contrast, the relation $\sim_d$ is strictly coarser. It is in the style of [13, 18], representing a linear time semantics. Both $\sim_s$ and $\sim_d$ are sound proof techniques for $\simeq$, which requires to prove that they are congruence relations. We show the congruence property by adapting Howe's method to the quantum setting. We also find that $\sim_d$ provides a complete proof technique for $\simeq$. In order to prove full-abstraction, we first characterise $\sim_d$ as a testing equivalence $=^{\mathcal{T}}$ given by a simple testing language. Since all the tests in the testing language can be simulated by linear contexts, we obtain that $\simeq \subseteq =^{\mathcal{T}}$, which implies that $\simeq \subseteq \sim_d$. To some extent this is a generalisation of the aforementioned coincidence result obtained in [8] because the distribution-based probabilistic bisimilarity $\sim_d$ captures a notion of trace equivalence in the probabilistic setting very intuitively.

1.1 Other Related Work

Reasoning about program equality in higher-order languages is challenging. Most of the time equivalence between two programs requires them to exhibit the same observable behaviour under any context. To alleviate the burden of dealing with all the contexts, a useful way out is to develop operational methods for proving program equivalence. For example, Abramsky's *applicative bisimulation* [1] has attracted a lot of attention, not only in the classical setting [14, 19, 25, 26], but also in the probabilistic setting [4, 2]. In [4] a notion of probabilistic applicative bisimulation is shown to be a sound technique for proving contextual equivalence. However, completeness fails and can only be recovered when pure, deterministic λ-terms are considered and a coupled logical bisimulation is used in place of applicative bisimulation. In [2] a probabilistic call-by-value λ-calculus is considered, where a probabilistic applicative bisimilarity is shown to be a sound *and complete* proof technique for contextual equivalence. Recently, the third author and Rioli have studied applicative bisimulation in a purely linear quantum λ-calculus, obtaining a soundness result [3]. Following this line of research, our work is carried out in a quantum setting and uses a distribution-based bisimilarity to characterise linear contextual equivalence. We also examine a state-based bisimilarity. It corresponds to the probabilistic applicative bisimilarity discussed above. The characterisation of state-based probabilistic bisimilarity by a set of tests, shown with an involved proof in [33] and with a tradition dated back to [22], is essential for the completeness proof of [2]. For distribution-based bisimilarity, however, a much simpler characterisation exists.

Variants of probabilistic bisimulation have already been used to compare the behaviour of quantum processes [21, 11, 5, 12, 10]. However, as far as we know, the current work is the first to explore operational techniques based on probabilistic bisimulation to reason about contextual equivalence of fully-featured higher-order quantum programs.

1.2 Structure of the Paper

In Section 2, we introduce the syntax, the reduction semantics, and the labelled transition semantics of a quantum λ-calculus. A linear contextual equivalence and two bisimilarities are defined. In Section 3 we show that the two notions of bisimilarity are congruence relations included in linear contextual equivalence. The distribution-based bisimilarity is shown to coincide with a testing equivalence in Section 4. By exploiting this result, we show that the distribution-based bisimilarity is complete with respect to linear contextual equivalence. Finally, we conclude in Section 5.

2 A Quantum λ-Calculus

Following [24] we introduce the syntax and operational semantics of a quantum λ-calculus.

2.1 Syntax

In this typed language, types are given by the following grammar:

$$A, B, C ::= \mathsf{qubit} \mid A \multimap B \mid !(A \multimap B) \mid 1 \mid A \otimes B \mid A \oplus B \mid A^{l}.$$

Here $A \multimap B$ is the usual linear function type and $(!A \multimap B)$ is a non-linear function type. For any arbitrary type A, the type $!A$ can be simulated by $!(1 \multimap A)$. The qubit type is used to classify terms that represent qubit information. Tensor and sum types are standard. We use the notation $A^{\otimes n}$ for A tensored n times. The type A^{l} denotes finite lists of type A.

Terms are built up from constants and variables, using the following constructs:

M, N, P	$::=$	x	Variables
	$\mid$	$\lambda x^{A}.M \mid M\,N$	Abstractions and applications
	$\mid$	$\mathtt{skip} \mid M; N$	Skip and sequential compositions
	$\mid$	$M \otimes N \mid \mathtt{let}\ x^{A} \otimes y^{B} = M\ \mathtt{in}\ N$	Tensor products and projections
	$\mid$	$\mathtt{in}_{l}\, M \mid \mathtt{in}_{r}\, M$	Sums
	$\mid$	$\mathtt{match}\ P\ \mathtt{with}\ (x^{A} : M \mid y^{B} : N)$	Matches
	$\mid$	$\mathtt{split}^{A}$	Split
	$\mid$	$\mathtt{letrec}\ f^{A \multimap B} x = M\ \mathtt{in}\ N$	Recursions
	$\mid$	$\mathtt{new} \mid \mathtt{meas} \mid \mathtt{U}$	Quantum operators

Most of the language constructs are standard. The tensor product and tensor projection are related to linearity. The quantum operators are used to prepare quantum systems. The constant $\mathtt{U}$ ranges over a set of elementary unitary transformations on quantum bits. Two typical examples are the Hadamard gate $\mathtt{H}$ and the controlled-not gate $\mathtt{N_c}$.

Variables appearing in the λ-binder, the $\mathtt{let}$-binder, the $\mathtt{match}$-binder, and the $\mathtt{letrec}$-binder are bound variables. We write $fv(M)$ for the set of free variables in term M. We will not distinguish *α-equivalent* terms, which are terms syntactically identical up to renaming of bound variables. If M and N are terms and x is a variable, then $M\{N/x\}$ denotes the term resulting from substituting N for all free occurrences of x in M. More generally, given a list $N_1, \ldots, N_n$ of terms and a list $x_1, \ldots, x_n$ of distinct variables, we write $M\{N_1/x_1, \ldots, N_n/x_n\}$

or simply $M\{\tilde{N}/\tilde{x}\}$ for the result of simultaneously substituting each N_i for free occurrences in M of the corresponding variable x_i.

Values are special terms in the following form

$$V, W ::= x \mid c \mid \lambda x^A.M \mid V \otimes W \mid \mathtt{in}_l\, V \mid \mathtt{in}_r\, W,$$

where c ranges over the set of constants $\{\mathtt{skip}, \mathtt{split}^A, \mathtt{meas}, \mathtt{new}, \mathtt{U}\}$. As syntactic sugar we write $\mathsf{bit} = 1 \oplus 1$, $\mathtt{tt} = \mathtt{in}_r\ \mathtt{skip}$, and $\mathtt{ff} = \mathtt{in}_l\ \mathtt{skip}$.

A *typing assertion* takes the form $\Delta \vdash M : A$, where Δ is a finite partial function from variables to types, M is a term, and A is a type. We write $dom(\Delta)$ for the domain of Δ. We call Δ *exponential* (resp. *linear*) whenever $\Delta(x)$ is (resp. is not) a !-type for each $x \in dom(\Delta)$. We write $!\Delta$ for a context that is exponential. The *type assignment relation* consists of all typing assertions that can be derived from the axioms and rules in Figure 1, where the contexts Δ' and Δ'' are assumed to be linear. The notation $\Delta, x : A$ denotes the partial function which properly extends Δ by mapping x to A, so it is assumed that $x \notin dom(\Delta)$. Similarly, in the notation Δ, Δ' the domains of Δ and Δ' are assmued to be disjoint. We write $\mathcal{P}rog(A) = \{M \mid \emptyset \vdash M : A\}$ for the set of all closed programs of type A.

For any typing assertion $\Delta \vdash M : A$, it is not difficult to see that $fv(M) \subseteq dom(\Delta)$. Let $fqv(M)$ be the set of free variables of qubit type in term M and $fcv(M)$ collects all other types of free variables. Thus we have $fv(M) = fcv(M) \cup fqv(M)$ for any term M. We often separate the free quantum variables in M from the type environment Δ and write $\Delta' \rhd M : A$ where $\Delta', x_1 : \mathsf{qubit}, \ldots, x_n : \mathsf{qubit} = \Delta$ with $\{x_1, \ldots, x_n\} = fqv(M)$. Let a *proved expression* be a triple (Δ, M, A) such that $\Delta \rhd M : A$. If $\Delta = x_1 : A_1, \ldots, x_n : A_n$, a Δ-*closure* is a substitution $\{\tilde{M}/\tilde{x}\}$ where each $M_i \in \mathcal{P}rog(A_i)$. If $\mathcal{R}$ is a relation on terms M with $fcv(M) = \emptyset$, its *open extension*, $\mathcal{R}^\circ$, is the least relation between proved expressions such that

$$(\Delta, M, A)\ \mathcal{R}^\circ\ (\Delta, N, A) \text{ iff } M\{\tilde{P}/\tilde{x}\}\ \mathcal{R}\ N\{\tilde{P}/\tilde{x}\} \text{ for any } \Delta\text{-closure } \{\tilde{P}/\tilde{x}\}\ .$$

We will write $\Delta \rhd M\ \mathcal{R}\ N : A$ to mean that $(\Delta, M, A)\ \mathcal{R}^\circ\ (\Delta, N, A)$. Sometimes we omit the type information if it is not important and simply write $\Delta \rhd M\ \mathcal{R}\ N$.

2.2 The Reduction Semantics

The reduction semantics is defined in terms of an abstract machine simulating the behaviour of the QRAM model [20].

▶ **Definition 1.** A *quantum closure* is a triple $[q, l, M]$ where

- q is a normalized vector of $\mathbb{C}^{2^n}$, for some integer $n \geq 0$. It is called the *quantum state*;
- M is a term, not necessarily closed;
- l is a *linking function* that is an injective map from $fqv(M)$ to the set $\{1, \ldots, n\}$.

We write $dom(l)$ for the domain of l. The notation $l \uplus m$ stands for the union of two linking functions l and m (viewed as two sets of pairs) if their domains are disjoint, otherwise it is undefined. A closure $[q, l, M]$ is *total* if l is surjective, thus a bijection. In that case we write l as $\langle x_1, \ldots, x_n\rangle$ if $dom(l) = \{x_1, \ldots, x_n\}$ and $l(x_i) = i$ for all $i \in \{1 \ldots n\}$. A quantum closure $C = [q, l, M]$ is well typed and has type A in Δ whenever $dom(l) = \{x_1, \ldots, x_m\}$ and $\Delta, x_1 : \mathsf{qubit}, \ldots, x_m : \mathsf{qubit} \vdash M : A$. In this case we write $\Delta \rhd C : A$. The notion of α-equivalence extends naturally to quantum closures. So, e.g., the two states $[q, \langle y\rangle, \lambda x^A.y]$ and $[q, \langle z\rangle, \lambda x^A.z]$ are deemed equivalent. With a slight abuse of language, we call a closure $[q, l, V]$ a *value* when the term V is a value. Most often we will work with closed quantum

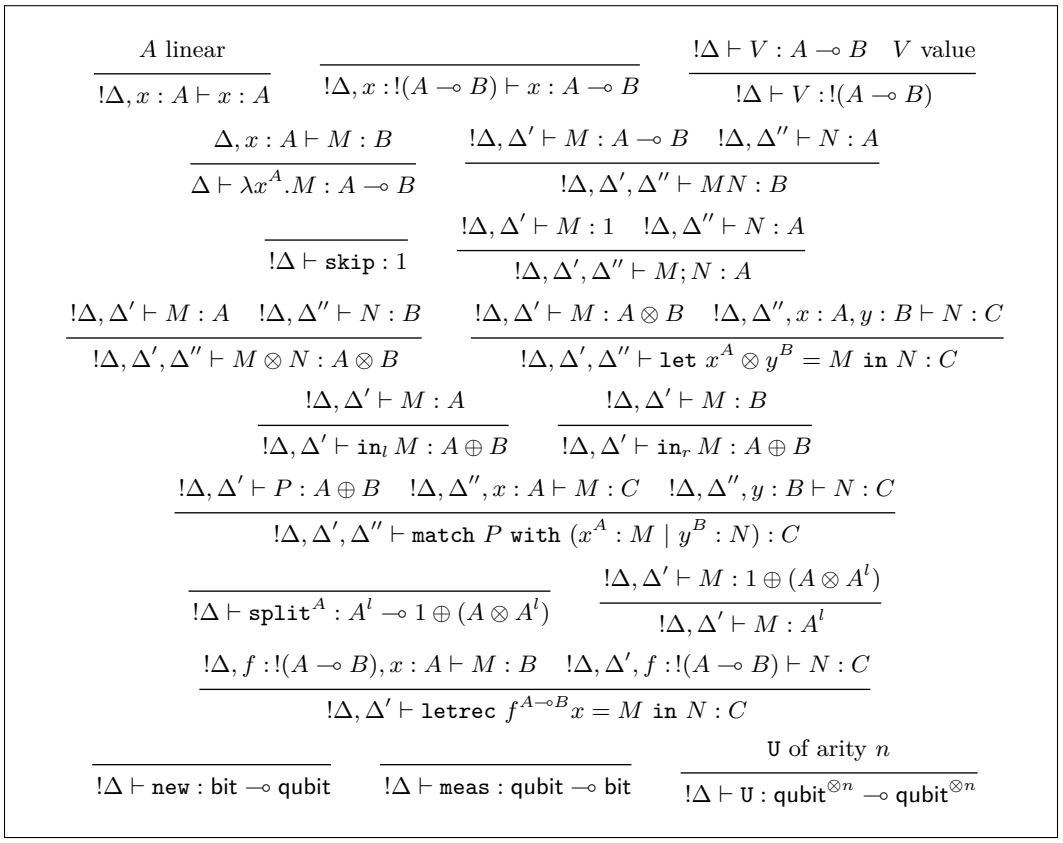


Figure 1 Typing Rules.

closures, i.e. with those closures C such that $\emptyset \rhd C : A$. We let Cl be the set of closed quantum closures. For the sake of simplicity, we often assume that the quantum closures we work with are typable without stating it explicitly.

We now introduce the notion $C \overset{p}{\rightsquigarrow} D$ to mean that closed quantum closure C reduces to D immediately with probability $p \in [0, 1]$. Formally, the one-step reduction $\overset{p}{\rightsquigarrow}$ between quantum closures is the smallest relation including the axioms from Figure 2 together with the structural rule

$$\frac{[q,\, l,\, M] \overset{p}{\rightsquigarrow} [r,\, i,\, N]}{[q,\, j \uplus l,\, \mathcal{E}[M]] \overset{p}{\rightsquigarrow} [r,\, j \uplus i,\, \mathcal{E}[N]]}$$

where $\mathcal{E}$ is any *evaluation context* generated by the grammar

$$\begin{aligned}\mathcal{E} ::= &[\,] \mid \mathcal{E}\, M \mid V\, \mathcal{E} \mid \mathcal{E}; M \mid \mathcal{E} \otimes M \mid V \otimes \mathcal{E} \mid \mathtt{in}_l\, \mathcal{E} \mid \mathtt{in}_r\, \mathcal{E} \\ &\mid \mathtt{let}\ x^A \otimes y^B = \mathcal{E}\ \mathtt{in}\ M \mid \mathtt{match}\ \mathcal{E}\ \mathtt{with}\ (x^A : M \mid y^B : N).\end{aligned}$$

In the two reduction rules for `new`, the quantum state q has size n, and x is a fresh variable. In the rule for unitary transformations, the quantum state r is obtained by applying the k-ary unitary gate $\mathtt{U}$ to the qubits $l(x_1), \ldots, l(x_k)$. In other words, $r = (\sigma \circ (\mathtt{U} \otimes id) \circ \sigma^{-1})(q)$, where σ is the action on $\mathbb{C}^{2^n}$ of any permutation over $\{1, \ldots, n\}$ such that $\sigma(i) = l(x_i)$ whenever $i \leq k$. In the rules for measurements, we assume that if q_0 and q_1 are normalized quantum states of the form $\sum_j \alpha_j |\varphi_j\rangle \otimes |0\rangle \otimes |\phi_j\rangle$, $\sum_j \beta_j |\varphi_j\rangle \otimes |1\rangle \otimes |\phi_j\rangle$, then r_0 and r_1 are

$$
\begin{array}{l}
[q,\, l,\, (\lambda x^A.M)V] \overset{1}{\rightsquigarrow} [q,\, l,\, M\{V/x\}] \\
[q,\, l,\, \mathtt{let}\ x^A \otimes y^B = V \otimes W\ \mathtt{in}\ N] \overset{1}{\rightsquigarrow} [q,\, l,\, N\{V/x, W/y\}] \\
[q,\, l,\, \mathtt{skip}; N] \overset{1}{\rightsquigarrow} [q,\, l,\, N] \\
[q,\, l,\, \mathtt{split}^A\, V] \overset{1}{\rightsquigarrow} [q,\, l,\, V] \\
[q,\, l,\, \mathtt{match}\ \mathtt{in}_l\, V\ \mathtt{with}\ (x^A : M \mid y^B : N)] \overset{1}{\rightsquigarrow} [q,\, l,\, M\{V/x\}] \\
[q,\, l,\, \mathtt{match}\ \mathtt{in}_r\, V\ \mathtt{with}\ (x^A : M \mid y^B : N)] \overset{1}{\rightsquigarrow} [q,\, l,\, N\{V/y\}] \\
[q,\, l,\, \mathtt{letrec}\ f^{A\multimap B}x = M\ \mathtt{in}\ N] \overset{1}{\rightsquigarrow} [q,\, l,\, N\{(\lambda x^A.\mathtt{letrec}\ f^{A\multimap B}x = M\ \mathtt{in}\ M)/f\}] \\
[q,\, \emptyset,\, \mathtt{new}\, \mathtt{ff}] \overset{1}{\rightsquigarrow} [q \otimes |0\rangle,\, \{x \mapsto n+1\},\, x] \\
[q,\, \emptyset,\, \mathtt{new}\, \mathtt{tt}] \overset{1}{\rightsquigarrow} [q \otimes |1\rangle,\, \{x \mapsto n+1\},\, x] \\
[\alpha q_0 + \beta q_1,\, \{x \mapsto i\},\, \mathtt{meas}\, x] \overset{|\alpha|^2}{\rightsquigarrow} [r_0,\, \emptyset,\, \mathtt{ff}] \\
[\alpha q_0 + \beta q_1,\, \{x \mapsto i\},\, \mathtt{meas}\, x] \overset{|\beta|^2}{\rightsquigarrow} [r_1,\, \emptyset,\, \mathtt{tt}] \\
[q,\, l,\, \mathtt{U}(x_1 \otimes \cdots \otimes x_k)] \overset{1}{\rightsquigarrow} [r,\, l,\, (x_1 \otimes \cdots \otimes x_k)]
\end{array}
$$

Figure 2 Small-Step Axioms.

respectively $\sum_j \alpha_j|\varphi_j\rangle \otimes |\phi_j\rangle$, $\sum_j \beta_j|\varphi_j\rangle \otimes |\phi_j\rangle$, where the vectors φ_j has dimension $l(x) - 1$. The reduction semantics defined above employs a *call-by-value* evaluation strategy.

▶ **Lemma 2** (Totality). *Let C and D be two quantum closures and $C \overset{p}{\rightsquigarrow} D$. If C is total then so is D.* ◀

▶ **Lemma 3** (Type safety). *Let $C = [q, l, M]$ be a closed quantum closure. Then either M is a value or the total probability of all one-step reductions from C is 1.* ◀

By Lemma 3 we see that the reduction semantics induces a Markov chain $(Cl, \to)$, where Cl is the set of all closed quantum closures and $\to \subseteq Cl \times \mathcal{D}(Cl)$ is the transition relation with $C \to \mu$ satisfying $\mu(D) = p$ iff $C \overset{p}{\rightsquigarrow} D$ for some $p > 0$. Here $\mathcal{D}(Cl)$ stands for all probability subdistributions over Cl and μ is a full distribution over all successor quantum closures of C.

For any $\mu = \sum_i p_i \cdot [q_i, l_i, M_i]$, let $env(\mu) = \sum_i p_i \cdot tr_{fqv(M)} q_i q_i^\dagger$ be the reduced quantum state of the qubits not referred to by M. In particular, if each $[q_i, l_i, M_i]$ in the support of μ is a total quantum closure, we then have $env(\mu) = |\mu|$.

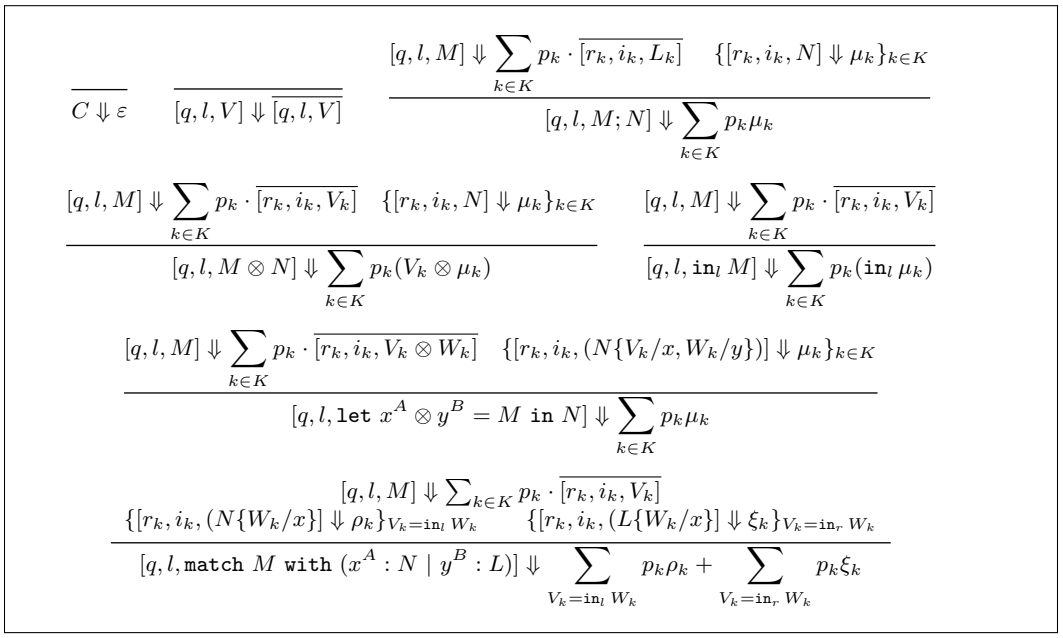
In order to investigate the long-term behaviour of a Markov chain, we introduce the notion of extreme derivative from [7]. We first need to lift the relation $\to$ to be a transition relation between subdistributions: $\mu \to \nu$ if $\nu = \sum_{C \in \lceil \mu \rceil} \mu(C) \cdot \mu_C$ and $C \to \mu_C$ for each C in $\lceil \mu \rceil$, the support of the subdistribution μ.

▶ **Definition 4** (Extreme derivative). Suppose we have subdistributions μ, $\mu_n^{\rightarrow}$, $\mu_n^{\times}$ for $n \geq 0$ with the following properties:

$$\mu = \mu_0^{\rightarrow} + \mu_0^{\times}; \qquad \forall k \geq 0.\ \mu_k^{\rightarrow} \to \mu_{k+1}^{\rightarrow} + \mu_{k+1}^{\times};$$

and each $\mu_k^{\times}$ is stable in the sense that $C \not\to$, for all $C \in \lceil \mu_k^{\times} \rceil$. Then we call $\rho := \sum_{k=0}^{\infty} \mu_k^{\times}$ an *extreme derivative* of μ, and write $\mu \Rightarrow \rho$.

Let C be a quantum closure in the Markov chain $(Cl, \to)$. The extreme derivative of the point distribution on C that assigns probability 1 to C, written $\overline{C}$, is unique, and we use it for the denotation of C, indicated as $[\![C]\!]$. So we always have $\overline{C} \Rightarrow [\![C]\!]$. Note that in the presence of divergence $[\![C]\!]$ may be a proper subdistribution.



$$\frac{}{C \Downarrow \varepsilon} \qquad \frac{}{[q,l,V] \Downarrow \overline{[q,l,V]}}$$

$$\frac{[q,l,M] \Downarrow \sum_{k\in K} p_k \cdot \overline{[r_k,i_k,L_k]} \quad \{[r_k,i_k,N] \Downarrow \mu_k\}_{k\in K}}{[q,l,M;N] \Downarrow \sum_{k\in K} p_k\mu_k}$$

$$\frac{[q,l,M] \Downarrow \sum_{k\in K} p_k \cdot \overline{[r_k,i_k,V_k]} \quad \{[r_k,i_k,N] \Downarrow \mu_k\}_{k\in K}}{[q,l,M\otimes N] \Downarrow \sum_{k\in K} p_k(V_k\otimes\mu_k)} \qquad \frac{[q,l,M] \Downarrow \sum_{k\in K} p_k \cdot \overline{[r_k,i_k,V_k]}}{[q,l,\mathtt{in}_l\, M] \Downarrow \sum_{k\in K} p_k(\mathtt{in}_l\,\mu_k)}$$

$$\frac{[q,l,M] \Downarrow \sum_{k\in K} p_k \cdot \overline{[r_k,i_k,V_k\otimes W_k]} \quad \{[r_k,i_k,(N\{V_k/x,W_k/y\})] \Downarrow \mu_k\}_{k\in K}}{[q,l,\mathtt{let}\ x^A\otimes y^B = M\ \mathtt{in}\ N] \Downarrow \sum_{k\in K} p_k\mu_k}$$

$$\frac{\begin{array}{c}[q,l,M] \Downarrow \sum_{k\in K} p_k \cdot \overline{[r_k,i_k,V_k]} \\ \{[r_k,i_k,(N\{W_k/x\}] \Downarrow \rho_k\}_{V_k=\mathtt{in}_l\, W_k} \qquad \{[r_k,i_k,(L\{W_k/x\}] \Downarrow \xi_k\}_{V_k=\mathtt{in}_r\, W_k}\end{array}}{[q,l,\mathtt{match}\ M\ \mathtt{with}\ (x^A:N \mid y^B:L)] \Downarrow \sum_{V_k=\mathtt{in}_l\, W_k} p_k\rho_k + \sum_{V_k=\mathtt{in}_r\, W_k} p_k\xi_k}$$

Figure 3 Big-Step Semantics Rules – Selection.

Extreme derivatives can also be defined by a *big-step* semantics given by using a binary relation $\Downarrow$ between quantum closures and value distributions. Some of the rules of it can be found in Figure 3 (the others are similar). In the rules, ε stands for the empty subdistribution, and the notation $|\mu|$ stands for the size of the subdistribution μ, i.e. $\sum_{C\in\lceil\mu\rceil}\mu(C)$. Finally, term constructors are, with abuse of notation, applied to quantum closures and subdistributions in the natural way, e.g., $\mathtt{in}_l\,(\sum_{k\in K} p_k\cdot\overline{[q,l,M]})$ stands for $\sum_{k\in K} p_k\cdot\overline{[q,l,\mathtt{in}_l\,M]}$.

▶ **Lemma 5.** *$[\![C]\!] = \sup\{\mu \mid C \Downarrow \mu\}$, where the supremum of subdistributions are computed component-wisely.* ◀

Following [8] we would like to give an alternative characterisation of linear contextual equivalence. Intuitively, as usual, a context is a term with a unique hole, and a linear context is a context where programs under examination will be evaluated and used *exactly once*. We are interested in *closing* contexts.

▶ **Definition 6.** A *linear context* (or simply a *context*) is a term with a hole, written $\mathcal{C}(\Delta;A)$, such that $\mathcal{C}[M]$ is a closed program whenever the hole is filled in by a term M, where $\Delta\rhd M:A$, and the hole lies in linear position.

Following [2], we require that the observable behaviour of a quantum closure C is its probability of convergence $|[\![C]\!]|$.

▶ **Definition 7.** The *linear contextual preorder* is the typed relation $\sqsubseteq$ defined as follows: $\Delta\rhd M\sqsubseteq N:A$ if for every linear context $\mathcal{C}$, quantum state q and linking function l such that $\emptyset\rhd\mathcal{C}(\Delta;A):B$, and both $[q,l,\mathcal{C}[M]]$ and $[q,l,\mathcal{C}[N]]$ are total quantum closures, it holds that $|[\![[q,l,\mathcal{C}[M]]]\!]| \le |[\![[q,l,\mathcal{C}[N]]]\!]|$. *Linear contextual equivalence* is the typed relation $\simeq$ by letting $\Delta\rhd M\simeq N:A$ just when $\Delta\rhd M\sqsubseteq N:A$ and $\Delta\rhd N\sqsubseteq M:A$.

$$\frac{}{[q, l, x_1 \otimes \cdots \otimes x_n] \xrightarrow{\mathtt{i\,U}} [q, l, \mathtt{U}(x_1 \otimes \cdots \otimes x_n)]} \qquad \frac{}{[q, l, x] \xrightarrow{\mathtt{i\,meas}} [q, l, \mathtt{meas}\, x]}$$

$$\frac{}{[q, \emptyset, \mathtt{skip}] \xrightarrow{\mathtt{skip}} [q, \emptyset, \mathbf{\Omega}]} \qquad \frac{\emptyset \rhd V : A \multimap B \quad \emptyset \rhd W : A}{[q,\ l,\ V] \xrightarrow{@[r,W]} [q,\ l \uplus r,\ VW]}$$

$$\frac{\emptyset \rhd \mathtt{in}_l\, V : A \oplus B \quad x : A \rhd M : C}{[q, l, \mathtt{in}_l\, V] \xrightarrow{\mathtt{l}\,[r,M]} [q, l \uplus r, M\{V/x\}]} \qquad \frac{\emptyset \rhd \mathtt{in}_r\, V : A \oplus B \quad y : B \rhd M : C}{[q, l, \mathtt{in}_r\, V] \xrightarrow{\mathtt{r}\,[r,M]} [q, l \uplus r, M\{V/y\}]}$$

$$\frac{\emptyset \rhd V \otimes W : A \otimes B \qquad x : A, y : B \rhd M : C}{[q, l, V \otimes W] \xrightarrow{\otimes[r,M]} [l \uplus r, M\{V/x, W/y\}]} \qquad \frac{}{C \xrightarrow{eval} [\![C]\!]}$$

Figure 4 Labelled Transition Rules for Quantum Closures.

2.3 A Probabilistic Labelled Transition System

In [14], Gordon defines explicitly a labelled transition system in order to illustrate the bisimulation technique in PCF. We follow this idea to define a *probabilistic* labelled transition system for the quantum λ-calculus, upon which we can define probabilistic bisimulations.

Transition rules are listed in Figure 4: we make the typing of terms explicit in the rules as the type system plays an important role in defining the operational semantics of typed terms. A transition takes the form $C \xrightarrow{a} \mu$, where C is a quantum closure, μ is a subdistribution over quantum closures, and a is an action. If μ is a point distribution $\overline{D}$, we simply write the transition as $C \xrightarrow{a} D$. Note that non-total quantum closures are needed here to specify the operational semantics: we cannot work only with total quantum closures due to entanglement. For example, we should allow for a non-total quantum closure like $[\frac{|00\rangle+|11\rangle}{\sqrt{2}}, \{x \mapsto 1\}, x]$, where the quantum variable x refers to the first qubit of an entangled quantum state.

The last rule in Figure 4 says that term reductions are considered as internal transitions that are abstracted away; external transitions are labelled by *actions*. Intuitively, external transitions represent the way terms interact with environments (or contexts). For instance, a λ-abstraction can "consume" (application of itself to) a term, which is supplied by the environment as an argument, and forms a β-reduction. The rule for `skip` says that what it can provide to the environment is the value of itself, and after that it cannot provide any information, hence no external transitions can occur any more. We represent this by a transition, labelled by the value of the constant, into a non-terminating program $\mathbf{\Omega}$ of appropriate type.

The set of quantum closures $\mathcal{C}l$ together with the transition rules in Figure 4 yields a probabilistic labelled transition system (pLTS). It is in fact a reactive system in the sense that if $C \xrightarrow{a} \mu_1$ and $C \xrightarrow{a} \mu_2$ then $\mu_1 = \mu_2$ for all $C \in \mathcal{C}l$, that is no two outgoing transitions leaving a quantum closure are labelled by the same action. Below we recall Larsen and Skou's probabilistic bisimulation [22]. We first review a way of lifting a binary relation $\mathcal{R}$ over a set S to be a relation $\mathcal{R}^\dagger$ over the set of subdistributions on S given in [9][1].

▶ **Definition 8.** Let S, T be two countable sets and $\mathcal{R} \subseteq S \times T$ be a binary relation. The lifted relation $\mathcal{R}^\dagger \subseteq \mathcal{D}(S) \times \mathcal{D}(T)$ is defined by letting $\mu\ \mathcal{R}^\dagger\ \nu$ iff $\mu(X) \leq \nu(\mathcal{R}(X))$ for all

[1] There are several different but equivalent formulations of the lifting operation; see e.g. [27, 6].

$X \subseteq S$. Here we write $\mathcal{R}(X)$ for the set $\{t \in T \mid \exists s \in X.\ s\ \mathcal{R}\ t\}$ and $\mu(X)$ is the accumulation probability $\sum_{s \in X} \mu(s)$.

The definition by Larsen and Skou, when instantiated on the pLTS of closed quantum closures, looks as follows:

▶ **Definition 9.** A *probabilistic simulation* is a preorder $\mathcal{R}$ on closed quantum closures such that whenever $(C, D) \in \mathcal{R}$ we have that:

- $env(C) = env(D)$;
- $[\![C]\!]\ \mathcal{R}^{\dagger}\ [\![D]\!]$;
- if C, D are values then $C \xrightarrow{a} \mu$ implies $D \xrightarrow{a} \nu$ with $\mu\ \mathcal{R}^{\dagger}\ \nu$.

A *probabilistic bisimulation* is a relation $\mathcal{R}$ such that both $\mathcal{R}$ and $\mathcal{R}^{-1}$ are probabilistic simulations. Let $\preceq$ and $\sim_s$ be the largest probabilistic simulation and bisimulation, called similarity and bisimilarity, respectively. Bisimilarity and similarity are relations on closed quantum closures, but can be generalized to open closures as follows:

Suppose $\Delta \triangleright M, N : A$. We write $\emptyset \triangleright M \preceq N : A$ if $[q, l, M] \preceq [q, l, N]$, and $\Delta \triangleright M \sim_s N : A$ if $[q, l, M] \sim_s [q, l, N]$, for any q and l such that $[q, l, M]$ and $[q, l, N]$ are both typable quantum closures.

For reactive pLTSs, the kernel of probabilistic similarity is probabilistic bisimilarity. That is, $\sim_s = \preceq \cap \preceq^{-1}$. This of course also applies to the specific pLTS we are working with. The probabilistic bisimilarity defined above is a binary relation between states, and thus sometimes called state-based bisimilarity. Alternatively, it is possible to directly define a (sub)distribution-based bisimilarity by comparing actions emitted from subdistributions. In order to do so, we first define a transition relation between subdistributions.

▶ **Definition 10.** We write $\mu \xrightarrow{a} \rho$ if $\rho = \sum_{s \in \lceil \mu \rceil} \mu(s) \cdot \mu_s$, where μ_s is determined as follows:

- either $s \xrightarrow{a} \mu_s$
- or there is no ν with $s \xrightarrow{a} \nu$, and in this case we set $\mu_s = \varepsilon$.

Note that this is a weaker notion of transition relation between subdistributions, compared with that defined on Page 432. If $\mu \xrightarrow{a} \mu'$ then some (not necessarily all) states in the support of μ can perform action a. For example, consider the two states s_2 and s_3 in Figure 5. Since $s_2 \xrightarrow{c} \overline{s_4}$ and s_3 cannot perform action c, the distribution $\mu = \frac{1}{2}\overline{s_2} + \frac{1}{2}\overline{s_3}$ can make the transition $\mu \xrightarrow{c} \frac{1}{2}\overline{s_4}$ to reach the subdistribution $\frac{1}{2}\overline{s_4}$. Let μ be a subdistribution over a reactive pLTS. After performing any action, it can reach a unique subdistribution. That is, if $\mu \xrightarrow{a} \nu$ and $\mu \xrightarrow{a} \rho$ then $\nu = \rho$. Given a subdistribution μ, we denote by $[\![\mu]\!]$ the subdistribution $\sum_{C \in \lceil \mu \rceil} \mu(C) \cdot [\![C]\!]$.

▶ **Definition 11.** A *distribution-based bisimulation* is a binary relation $\mathcal{R}$ on subdistributions such that $\mu\ \mathcal{R}\ \nu$ implies

- $env(\mu) = env(\nu)$;
- $[\![\mu]\!]\ \mathcal{R}\ [\![\nu]\!]$;
- if μ and ν are value distributions and $\mu \xrightarrow{a} \rho$, then $\nu \xrightarrow{a} \xi$ for some ξ with $\rho\ \mathcal{R}\ \xi$, and vice-versa.

Let $\sim_d$ be the largest distribution-based bisimulation. Suppose $\emptyset \triangleright M, N : A$. We write $\emptyset \triangleright M \sim_d N : A$ if $[\![[q, l, M]]\!] \sim_d [\![[q, l, N]]\!]$ for any q and l such that $[q, l, M]$ and $[q, l, N]$ are quantum closures.

It is not difficult to see that $s \sim_s t$ implies $\overline{s} \sim_d \overline{t}$ but not the other way around, as witnessed by the following example.

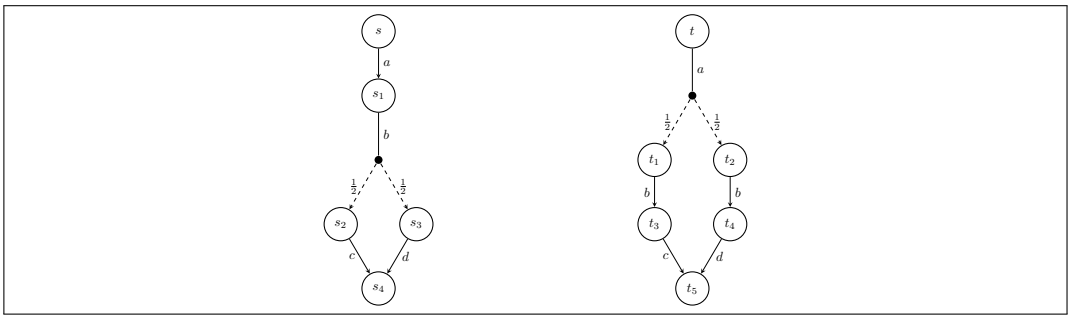


Figure 5 $s \not\sim_s t$.

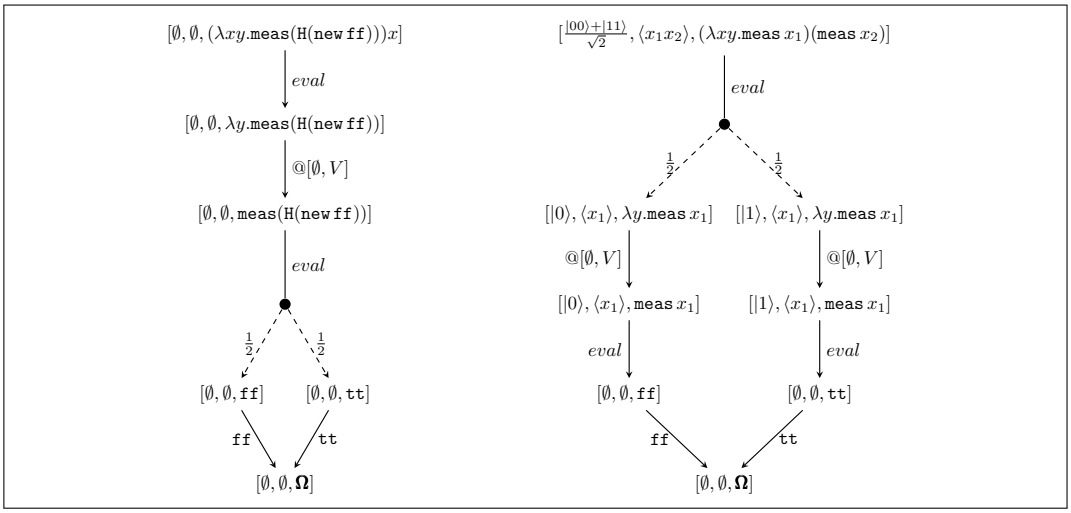


Figure 6 Two Quantum Closures not Related by $\sim_s$.

▶ **Example 12.** Consider the two states s and t in Figure 5. We can construct the relation

$$\mathcal{R} = \{(\overline{s}, \overline{t}), \left(\overline{s_1}, \frac{1}{2}\overline{t_1} + \frac{1}{2}\overline{t_2}\right), \left(\frac{1}{2}\overline{s_2} + \frac{1}{2}\overline{s_3}, \frac{1}{2}\overline{t_3} + \frac{1}{2}\overline{t_4}\right), (\overline{s_2}, \overline{t_3}), (\overline{s_3}, \overline{t_4}), (\overline{s_4}, \overline{t_5}))\}$$

and check that $\mathcal{R}$ is a distribution-based bisimulation. Therefore, we have $\overline{s} \sim_d \overline{t}$. Note that the point distribution at state s_1 is related to the distribution $\frac{1}{2}\overline{t_1} + \frac{1}{2}\overline{t_2}$. However, we have $s \not\sim_s t$ because after performing action a the state s evolves into the point distribution $\overline{s_1}$, and the only candidate transition from t to match this is $t \xrightarrow{a} \mu$ where $\mu = \frac{1}{2}\overline{t_1} + \frac{1}{2}\overline{t_2}$. But then the condition $\overline{s_1} \sim_s{}^{\dagger} \mu$ is invalid because there is no way to split s_1 into two different states such that they are bisimilar to t_1 and t_2 respectively. In the quantum λ-calculus this distinction between state-based and distribution-based bisimulations also exists. For example, the quantum closures $[\emptyset, \emptyset, (\lambda xy.\mathtt{meas}(\mathtt{H}(\mathtt{new}\,\mathtt{ff})))x]$ and $[\frac{|00\rangle+|11\rangle}{\sqrt{2}}, \langle x_1 x_2\rangle, (\lambda xy.\mathtt{meas}\,x_1)(\mathtt{meas}\,x_2)]$ exhibit similar behaviour as states s and t, respectively, as depicted in Figure 6. ◀

3 Congruence

In this section we show that both $\sim_s$ and $\sim_d$ are congruence relations. The proof for $\sim_s$ is more complicated, so we take it as an example and give the details. The case for $\sim_d$ follows

$$\frac{\Delta \rhd c \quad c \in \{x, \mathtt{skip}, \mathtt{split}^A, \mathtt{new}, \mathtt{meas}, \mathtt{U}\}}{\Delta \rhd [q,l,c]\ \hat{\mathcal{R}}\ [q,l,c]} \qquad \frac{\Delta, x : A \rhd [q,l,M]\ \ \mathcal{R}\ \ [r,j,N]}{\Delta \rhd [q,l,(\lambda x^A.M)]\ \hat{\mathcal{R}}\ [r,j,(\lambda x^A.N)]}$$

$$\frac{!\Delta, \Delta' \rhd [q,l,M]\ \ \mathcal{R}\ \ [r,j,N] \quad !\Delta, \Delta'' \rhd [q,i,L]\ \ \mathcal{R}\ \ [r,m,P]}{!\Delta, \Delta', \Delta'' \rhd [q, l \uplus i, ML]\ \hat{\mathcal{R}}\ [r, j \uplus m, NP]}$$

$$\frac{!\Delta, \Delta' \rhd [q,l,M]\ \ \mathcal{R}\ \ [r,j,N] \quad !\Delta, \Delta'' \rhd [q,i,L]\ \ \mathcal{R}\ \ [r,m,P]}{!\Delta, \Delta', \Delta'' \rhd [q, l \uplus i, M \otimes L]\ \hat{\mathcal{R}}\ [q, j \uplus m, N \otimes P]}$$

$$\frac{!\Delta, \Delta' \rhd [q,l,M]\ \ \mathcal{R}\ \ [r,j,N]}{!\Delta, \Delta' \rhd [q,l,\mathtt{in}_l\, M]\ \hat{\mathcal{R}}\ [r,j,\mathtt{in}_l\, N]} \qquad \frac{!\Delta, \Delta' \rhd [q,l,M]\ \ \mathcal{R}\ \ [r,j,N]}{!\Delta, \Delta' \rhd [q,l,\mathtt{in}_r\, M]\ \hat{\mathcal{R}}\ [r,j,\mathtt{in}_r\, N]}$$

Figure 7 Compatible Refinement Rules – Selection.

the same schema. The basic idea is to make use of Howe's method [19, 26], which requires to start from an initial relation $\mathcal{R}$, define a precongruence candidate $\mathcal{R}^H$, a precongruence relation by construction, and then to show the coincidence of that relation with the initial relation.

▶ **Definition 13.** Let $\mathcal{R}$ be a typed relation on quantum closures. Its *compatible refinement* $\hat{\mathcal{R}}$ is defined by some natural rules, a selection of which is in Figure 7. A relation $\mathcal{R}$ is a *precongruence* iff it contains its own compatible refinement, that is $\hat{\mathcal{R}} \subseteq \mathcal{R}$. Let a *congruence* be an equivalence relation that is a precongruence.

Let $\mathcal{R}$ be a typed relation on quantum closures. The typed relation $\mathcal{R}^H$ is defined by the rules in Figure 8. Note that if $\mathcal{R}$ is reflexive then $\mathcal{R} \subseteq \mathcal{R}^H$, and $\mathcal{R}^H$ is a precongruence. Therefore, in order to show that $\mathcal{R}$ is a precongruence (or congruence if $\mathcal{R}$ is also symmetric), it suffices to establish $\mathcal{R}^H \subseteq \mathcal{R}$ because we then have the coincidence of $\mathcal{R}$ with $\mathcal{R}^H$. In order to show $(\sim_s)^H \subseteq \sim_s$, we need the following two technical lemmas.

▶ **Lemma 14.** *If* $\emptyset \rhd [q,l,M] \preceq^H [r,j,N]$ *then* $[\![[q,l,M]]\!]\ (\preceq^H)^\dagger\ [\![[r,j,N]]\!]$. ◀

▶ **Lemma 15.** *If* $\emptyset \rhd [q,l,V] \preceq^H [r,j,W]$ *then we have that* $[q,l,V] \xrightarrow{a} \mu$ *implies* $[r,j,W] \xrightarrow{a} \nu$ *and* $\mu\ (\preceq^H)^\dagger\ \nu$. ◀

Consequently, we can establish the coincidence of $\preceq$ with $\preceq^H$, from which it is easy to show that $\sim_s$ is a congruence. Similar arguments apply to $\sim_d$.

▶ **Theorem 16.** *Both* $\sim_s$ *and* $\sim_d$ *are included in* $\simeq$. ◀

4 Completeness of Distribution-Based Bisimilarity

In this section we show that distribution-based bisimilarity is complete for linear contextual equivalence. The basic idea is to first characterise bisimilarity by a very simple testing framework. Let $\mathcal{T}$ be the set of tests of the two forms: ω and $a \cdot \mathtt{t}$, where ω is used to indicate success and a ranges over the set of all possible labels in the transition rules in Figure 4. In other words, the testing language is given by the grammar: $\mathtt{t} ::= \omega \mid a \cdot \mathtt{t}$.

Below we define the function Pr that calculates the probability of passing a test for a distribution of states in a reactive pLTS.

$$\begin{array}{rcl} Pr(\mu, \omega) & = & |\mu| \\ Pr(\mu, a \cdot \mathtt{t}) & = & Pr(\rho, \mathtt{t}) \text{ where } \mu \xrightarrow{a} \rho \end{array}$$

$$\frac{\Delta, x : A \rhd [q, l, M]\ \mathcal{R}^H\ [r, j, N] \quad \Delta \rhd [r, j, \lambda x^A.N]\ \ \mathcal{R}\ \ [p, i, L]}{\Delta \rhd [q, l, \lambda x^A.M]\ \mathcal{R}^H\ [p, i, L]}$$

$$\frac{\begin{array}{c}!\Delta, \Delta' \rhd [q, l, M]\ \mathcal{R}^H\ [r, j, N] \\ !\Delta, \Delta'' \rhd [q, i, L]\ \mathcal{R}^H\ [r, m, P] \\ !\Delta, \Delta', \Delta'' \rhd [r, j \uplus m, NP]\ \ \mathcal{R}\ \ [s, n, Q]\end{array}}{!\Delta, \Delta', \Delta'' \rhd [q, l \uplus i, ML]\ \mathcal{R}^H\ [s, n, Q]}$$

$$\frac{\begin{array}{c}!\Delta, \Delta' \rhd [q, l, M]\ \mathcal{R}^H\ [r, j, N] \\ !\Delta, \Delta'' \rhd [q, i, L]\ \mathcal{R}^H\ [r, m, P] \\ !\Delta, \Delta', \Delta'' \rhd [r, j \uplus m, N \otimes P]\ \ \mathcal{R}\ \ [s, n, Q]\end{array}}{!\Delta, \Delta', \Delta'' \rhd [q, l \uplus i, M \otimes L]\ \mathcal{R}^H\ [s, n, Q]}$$

$$\frac{!\Delta, \Delta' \rhd [q, l, M]\ \mathcal{R}^H\ [r, j, N] \quad !\Delta, \Delta' \rhd [r, j, \mathtt{in}_l\ N]\ \mathcal{R}\ [p, i, L]}{!\Delta, \Delta' \rhd [q, l, \mathtt{in}_l\ M]\ \mathcal{R}^H\ [p, i, L]}$$

$$\frac{!\Delta, \Delta' \rhd [q, l, M]\ \mathcal{R}^H\ [r, j, N] \quad !\Delta, \Delta' \rhd [r, j, \mathtt{in}_r\ N]\ \mathcal{R}\ [p, i, L]}{!\Delta, \Delta' \rhd [q, l, \mathtt{in}_r\ M]\ \mathcal{R}^H\ [p, i, L]}$$

Figure 8 Howe's Construction Rules – Selection.

If μ is a point distribution $\overline{s}$, we will write $Pr(s, \mathtt{t})$ for $Pr(\mu, \mathtt{t})$. We define a testing equivalence $=^{\mathcal{T}}$ by letting $\mu =^{\mathcal{T}} \nu$ iff $\forall \mathtt{t} \in \mathcal{T} : Pr(\mu, \mathtt{t}) = Pr(\nu, \mathtt{t})$. It turns out that the tests in $\mathcal{T}$ are sufficient to characterise $\sim_d$ as far as reactive pLTSs are concerned.

▶ **Theorem 17.** *Let μ and ν be two distributions in a reactive pLTS. Then $\mu \sim_d \nu$ if and only if $\mu =^{\mathcal{T}} \nu$.* ◀

Following [2], we turn each test into a corresponding context. That is, for a given test $\mathtt{t}$ and a given type A, there exists a linear context $\mathcal{C}^A_{\mathtt{t}}$ such that for all terms M of type A, the success probability of $\mathtt{t}$ applied to any total quantum closure $[q, l, M]$ is exactly the convergence probability of $[q, l, \mathcal{C}^A_{\mathtt{t}}[M]]$.

▶ **Lemma 18.** *Let A be a type and $\mathtt{t}$ a test. There is a context $\mathcal{C}^A_{\mathtt{t}}$ such that $\emptyset \rhd \mathcal{C}^A_{\mathtt{t}}(\emptyset; A) : \mathsf{bit}$ and for every M with $\emptyset \rhd M : A$, we have $Pr([q, l, M], \mathtt{t}) = |[\![[q, l, \mathcal{C}^A_{\mathtt{t}}[M]]]\!]|$, where $[q, l, M]$ and $[q, l, \mathcal{C}^A_{\mathtt{t}}[M]]$ are quantum closures for any q and l.* ◀

As a consequence of the previous lemma, we can show that the distribution-based bisimilarity $\sim_d$ is complete with respect to the linear contextual equivalence $\simeq$.

▶ **Theorem 19** (Full Abstraction)**.** *$\simeq$ coincides with $\sim_d$.* ◀

5 Concluding Remarks

We have presented two notions of bisimilarity for reasoning about equivalence of higher-order quantum programs in linear contexts, based on an appropriate labelled transition system for specifying the operational behaviour of programs. Both bisimilarities are sound with respect to the linear contextual equivalence, but only the distribution-based one turns out to be complete. Since linear resources are widely used in quantum computation, we believe that linear contextual equivalence will be a useful notion of behavioural equivalence for quantum

programs. The coinductive proof techniques developed in the current work can help to reason about quantum programs.

In the future, it would be interesting to seek a denotational model fully abstract with respect to the linear contextual equivalence. As recently shown in [35], Fock spaces can be useful to interpret quantum computation and they are close to the categorical semantics studied in [24], so it seems promising to start from there.

References

1 Samson Abramsky. The lazy lambda calculus. In *Research Topics in Functional Programming*, pages 65–116. Addison-Wesley, 1999.

2 Raphaëlle Crubillé and Ugo Dal Lago. On probabilistic applicative bisimulation and call-by-value λ-calculi. In *ESOP'14*, volume 8410 of *LNCS*, pages 209–228. Springer, 2014.

3 Ugo Dal Lago and Alessandro Rioli. Applicative bisimulation and quantum λ-calculi (long version). *CoRR*, abs/1506.06661, 2015.

4 Ugo Dal Lago, Davide Sangiorgi, and Michele Alberti. On coinductive equivalences for higher-order probabilistic functional programs. In *POPL'14*, pages 297–308. ACM, 2014.

5 Timothy A.S. Davidson. *Formal verification techniques using quantum process calculus*. PhD thesis, University of Warwick, 2011.

6 Yuxin Deng. *Semantics of Probabilistic Processes: An Operational Approach*. Springer, 2014.

7 Yuxin Deng, Rob van Glabbeek, Matthew Hennessy, and Carroll Morgan. Testing finitary probabilistic processes (extended abstract). In *CONCUR'09*, volume 5710 of *LNCS*, pages 274–288. Springer, 2009.

8 Yuxin Deng and Yu Zhang. Program equivalence in linear contexts. *Theoretical Computer Science*, 585:71–90, 2015.

9 Josée Desharnais. *Labelled Markov Processes*. PhD thesis, McGill University, 1999.

10 Yuan Feng, Yuxin Deng, and Mingsheng Ying. Symbolic bisimulation for quantum processes. *ACM Transactions on Computational Logic*, 15(2):1–32, 2014.

11 Yuan Feng, Runyao Duan, Zheng-Feng Ji, and Mingsheng Ying. Probabilistic bisimulations for quantum processes. *Information and Computation*, 205(11):1608–1639, 2007.

12 Yuan Feng, Runyao Duan, and Mingsheng Ying. Bisimulation for quantum processes. *ACM Transactions on Programming Languages and Systems*, 34(4):17, 2012.

13 Yuan Feng and Lijun Zhang. When equivalence and bisimulation join forces in probabilistic automata. In *FM'14*, volume 8442 of *LNCS*, pages 247–262. Springer, 2014.

14 Andrew M. Gordon. A tutorial on co-induction and functional programming. In *Glasgow Workshop on Functional Programming*, pages 78–95. Springer, 1995.

15 Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: a scalable quantum programming language. In *PLDI'13*, pages 333–342. ACM, 2013.

16 Lov K. Grover. A fast quantum mechanical algorithm for database search. In *STOC'96*, pages 212–219. ACM, 1996.

17 Lov K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physical Review Letters*, 79:325–328, 1997.

18 Holger Hermanns, Jan Krcál, and Jan Kretínský. Probabilistic bisimulation: Naturally on distributions. In *CONCUR'14*, volume 8704 of *LNCS*, pages 249–265. Springer, 2014.

19 Douglas J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.

20 Emanuel Knill. Conventions for quantum pseudocode. Technical Report LAUR-96-2724, Los Alamos National Laboratory, 1996.

21 Marie Lalire. Relations among quantum processes: bisimilarity and congruence. *Mathematical Structures in Computer Science*, 16(3):407–428, 2006.

22 Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

23 James H. Morris. *Lambda Calculus Models of Programming Languages*. PhD thesis, MIT, 1969.

24 Michele Pagani, Peter Selinger, and Benoît Valiron. Applying quantitative semantics to higher-order quantum computing. In *POPL'14*, pages 647–658. ACM, 2014.

25 Andrew M. Pitts. Operationally-based theories of program equivalence. In *Semantics and Logics of Computation*, pages 241–298. Cambridge University Press, 1997.

26 Andrew M. Pitts. Howe's method for higher-order languages. In *Advanced Topics in Bisimulation and Coinduction*, pages 197–232. Cambridge University Press, 2011.

27 Joshua Sack and Lijun Zhang. A general framework for probabilistic characterizing formulae. In *VMCAI'12*, volume 7148 of *LNCS*, pages 396–411. Springer, 2012.

28 Peter Selinger. Towards a quantum programming language. *Mathematical Structures in Computer Science*, 14(4):527–586, 2004.

29 Peter Selinger and Benoît Valiron. A lambda calculus for quantum computation with classical control. *Mathematical Structures in Computer Science*, 16(3):527–552, 2006.

30 Peter Selinger and Benoît Valiron. A linear-non-linear model for a computational call-by-value lambda calculus (extended abstract). In *FOSSACS'08*, volume 4962 of *LNCS*, pages 81–96. Springer, 2008.

31 Peter Selinger and Benoît Valiron. On a fully abstract model for a quantum linear functional language (extended abstract). *Electronic Notes in Theoretical Computer Science*, 210:123–137, 2008.

32 Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *FOCS'94*, pages 124–134. IEEE Computer Society, 1994.

33 Franck van Breugel, Michael W. Mislove, Joël Ouaknine, and James Worrell. Domain theory, testing and simulation for labelled markov processes. *Theoretical Computer Science*, 333(1-2):171–197, 2005.

34 Dave Wecker and Krysta M. Svore. LIQUi|>: A software design architecture and domain-specific language for quantum computing. *CoRR*, abs/1402.4467, 2014.

35 Mingsheng Ying. Quantum recursion and second quantisation: Basic ideas and examples. *CoRR*, abs/1405.4443, 2014.

Toward Automatic Verification of Quantum Cryptographic Protocols

Yuan Feng[1,3] and Mingsheng Ying[1,2]

1 **University of Technology Sydney, Australia,**
{Yuan.Feng,Mingsheng.Ying}@uts.edu.au
2 **Department of Computer Science and Technology, National Laboratory for Information Science and Technology, Tsinghua University, China**
3 **AMSS-UTS Joint Research Laboratory for Quantum Computation, Chinese Academy of Sciences, China**

Abstract

Several quantum process algebras have been proposed and successfully applied in verification of quantum cryptographic protocols. All of the bisimulations proposed so far for quantum processes in these process algebras are state-based, implying that they only compare individual quantum states, but not a combination of them. This paper remedies this problem by introducing a novel notion of distribution-based bisimulation for quantum processes. We further propose an approximate version of this bisimulation that enables us to prove more sophisticated security properties of quantum protocols which cannot be verified using the previous bisimulations. In particular, we prove that the quantum key distribution protocol BB84 is sound and (asymptotically) secure against the intercept-resend attacks by showing that the BB84 protocol, when executed with such an attacker concurrently, is approximately bisimilar to an ideal protocol, whose soundness and security are obviously guaranteed, with at most an exponentially decreasing gap.

1998 ACM Subject Classification C.2.2 Protocol verification

Keywords and phrases Quantum cryptographic protocols, Verification, Bisimulation, Security

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.441

1 Introduction

Quantum cryptography can provide unconditional security; it allows the realisation of cryptographic tasks that are proven or conjectured to be impossible in classical cryptography. The security of quantum cryptographic protocols is mathematically provable, based on the principles of quantum mechanics, without imposing any restrictions on the computational capacity of attackers. The proof is, however, often notoriously difficult, which is evidenced by the 50 pages long security proof of the quantum key distribution protocol BB84 [20]. It is hard to imagine such an analysis being carried out for more sophisticated quantum protocols. Thus, techniques for (semi-)automated verification of quantum protocols will be indispensable, given that quantum communication systems are already commercially available.

Process algebra has been successfully applied in the verification of classical (non-quantum) cryptographic protocols [21, 25]. One key step for such a process algebraic approach is a suitable notion of *bisimulation* which has appropriate distinguishing power and is preserved by various process constructs. Intuitively, two systems are bisimilar if and only if each observable action of one of them can be simulated by the other by performing the same observable action (possibly preceded and/or followed by some unobservable internal actions), and furthermore,

the resultant systems are again bisimilar. To verify a cryptographic protocol, we first give a *specification* which is an ideal protocol with obvious correctness and security, and then show that the given protocol is bisimilar (or approximately bisimilar with a small perturbation) to the specification.

In the last 10 years, several quantum process algebras like CQP [13], QPAlg [16] and qCCS [10] have been introduced, which provide an intuitive but rigorous way to model and reason about quantum communication systems. In particular, they have been adopted in verification of several popular quantum communication protocols such as Teleportation, Superdense Coding, etc. Similar to the classical case, the notion of bisimulation is crucial in the process algebra-based verification of quantum protocols. Actually, several different versions of bisimulation have been proposed for quantum processes in the recent literature [19, 27, 11, 5, 6]. A key feature of all of them is that they are state-based in the sense that they only compare individual configurations but not a combination of them. More explicitly, they are defined to be relations over configurations which are pairs of a quantum process and a density operator describing the state of environment quantum systems. However, when distributions of configurations are considered (which is inevitable for protocols where randomness is employed or quantum measurement is involved), state-based bisimulations are too discriminative – they distinguish some distributions which will never be distinguished by any outside observers, thereby providing the potential attacker of a cryptographic protocol with unrealistic power. As an extreme example, a state-based bisimulation distinguishes the distribution $p\langle \mathbf{nil}, \rho\rangle + (1-p)\langle \mathbf{nil}, \sigma\rangle$ from the single configuration $\langle \mathbf{nil}, p\rho + (1-p)\sigma\rangle$ if $\rho \neq \sigma$, where **nil** is the *dead* process incapable of performing any action.

In this paper, we propose a novel bisimulation for quantum processes which is defined directly on distributions of quantum configurations. Compared with existing bisimulations in the literature, our definition is strictly coarser (in particular, equates the two distributions presented above) and takes into account the combination of accompanied quantum states. We further define a pseudo-metric to characterise the extent to which two quantum processes are bisimilar. Note that we only consider quantum processes written in qCCS, but the main results can be generalised to other quantum process algebras like CQP and QPAlg easily.

To illustrate the utility of distribution-based bisimulation and the pseudo-metric in verification of quantum cryptographic protocols, we analyse the soundness and security of the well-known BB84 quantum key distribution protocol [4]. For the soundness, we show that when executed alone (without the presence of an attacker), BB84 is bisimilar to an ideal protocol which always returns a uniformly distributed (conditioning on a given key size) key. For the security analysis, we prove that when BB84 is executed concurrently with an intercept-resend attacker, the whole system is approximately bisimilar, with at most an exponentially decreasing gap, to an ideal protocol which never reports failure or information leakage. To the best of our knowledge, this is the first time (a weak notion of) security of BB84 is formally described and verified in the quantum process algebra approach.

Related works. The problem of existing bisimulations, as pointed out in the third paragraph of this section, was also noted by Kubota et al. [17]. To deal with it, they adopted two different semantics for quantum measurements. When a measurement induces a probability distribution in which all configurations have the same observable actions, it is represented semantically as a super-operator obtained by discarding the measurement outcome (thus no probabilistic branching is produced, and all post-measurement quantum states are merged). Otherwise, the measurement has the same semantics as in the original qCCS. This treatment solves the problem when probabilistic behaviours are only induced by quantum measurements. However, it does not work when probabilistic choice is included

in the syntax level, as we do in describing BB84 protocol in this paper. Furthermore, it brings difficulty in deciding the right semantics of a quantum process where a measurement is involved, as determining if the observable actions of the post-measurement configurations are all the same might not be easy; sometimes it even depends on the later input from the environment. In this paper, we solve this problem by revising the definition of bisimulation, instead of the definition of semantics.

In the same paper [17], Kubota et al. applied qCCS (with the semantic modification mentioned above) to show the security of BB84. They proved that BB84 is bisimilar to an EDP-based protocol, following the proof of Shor and Preskill [26]. However, this should not be regarded as a complete security proof, as it relies on the security of the EDP-based protocol. In contrast, our approach shows the security of BB84 directly. Note that for this purpose, a notion of approximate bisimulation, which was not presented in [17], is necessary, as BB84 is secure only in the sense that the eavesdropper's information about the secure key obtained by the legitimate parties is arbitrarily small (but still can be strictly positive) when the number of qubits transmitted (called the *security parameter*) goes to infinity.

Software tools based on the quantum process algebra CQP have been developed in [2] and [3] to check the equivalence between quantum sequential programs as well as concurrent protocols. These tools were applied to verify the correctness of protocols like Teleportation, Bit Flip Error Correction Code, and Quantum Secret Sharing. However, verification of security properties in cryptographic protocols such as BB84 has not been reported yet.

Besides the process algebra approach, model-checking is another promising approach for verification of quantum cryptographic protocols. For example, by observing the fact that the quantum states appearing in BB84, when only intercept-resend eavesdroppers are considered, are all the so-called stabiliser states which can be efficiently encoded in a classical way, Nagarajan et al. [22] analysed the security of BB84 by using the probabilistic model checker PRISM [18].

2 Preliminaries

In this section we review the model of probabilistic labelled transition systems (pLTSs) and the notion of lifted relations. Later on we will interpret the behaviour of quantum processes in terms of pLTSs.

2.1 Probabilistic labelled transition systems

A (finite-support) *probability distribution* over a set S is a function $\mu : S \to [0,1]$ with $\mu(s) > 0$ for finitely many $s \in S$ and $\sum_{s\in S} \mu(s) = 1$; the support of such a μ is the set $\lceil \mu \rceil = \{\, s \in S \mid \mu(s) > 0 \,\}$. The *point distribution* $\overline{s}$ assigns probability 1 to s and 0 to all other elements of S, so that $\lceil \overline{s} \rceil = \{s\}$. We use $D(S)$ to denote the set of probability distributions over S, ranged over by μ, ν etc. If $\sum_{i\in I} p_i = 1$ for some collection of $p_i \geq 0$, and $\mu_i \in D(S)$, then $\sum_{i\in I} p_i \cdot \mu_i \in D(S)$ is a *combined* probability distribution with $(\sum_{i\in I} p_i \cdot \mu_i)(s) = \sum_{i\in I} p_i \cdot \mu_i(s)$. We always assume the index set I to be finite.

▶ **Definition 1.** A *probabilistic labelled transition system* (pLTS) is a triple $\langle S, \mathsf{Act}, \longrightarrow \rangle$, where S is a set of *states*, Act is a set of *transition labels* with a special element τ included, and the *transition relation* $\longrightarrow$ is a subset of $S \times \mathsf{Act} \times D(S)$.

2.2 Lifting relations

In a pLTS actions are only performed by states, in that they are given by relations from states to distributions. But in general we allow distributions over states to perform an action. For this purpose, we *lift* these relations to distributions [7, 6].

▶ **Definition 2** (Lifting). Let $\mathcal{R} \subseteq S \times D(S)$ be a relation. The lifted relation, denoted by $\mathcal{R}$ again for simplicity, is the smallest relation $\mathcal{R} \subseteq D(S) \times D(S)$ that satisfies

1. $s\mathcal{R}\nu$ implies $\overline{s}\mathcal{R}\nu$, and
2. (Linearity) $\mu_i \mathcal{R} \nu_i$ for $i \in I$ implies $(\sum_{i\in I} p_i \cdot \mu_i)\mathcal{R}(\sum_{i\in I} p_i \cdot \nu_i)$ for any $p_i \in [0,1]$ with $\sum_{i\in I} p_i = 1$.

We apply this operation to the relations $\xrightarrow{\alpha}$ in a pLTS for $\alpha \in \mathsf{Act}$. Thus as source of a relation $\xrightarrow{\alpha}$ we also allow distributions. But $\overline{s} \xrightarrow{\alpha} \mu$ is more general than $s \xrightarrow{\alpha} \mu$, because if $\overline{s} \xrightarrow{\alpha} \mu$ then there is a collection of distributions μ_i and probabilities p_i such that $s \xrightarrow{\alpha} \mu_i$ for each $i \in I$ and $\mu = \sum_{i\in I} p_i \cdot \mu_i$ with $\sum_{i\in I} p_i = 1$; that is, we allow different transitions to be combined together, provided that they have the same source s and the same label α.

Sometimes we also need to lift a relation on states, say a state-based bisimulation, to distributions. This can be done by the following two steps. Let $\mathcal{R} \subseteq S \times S$ be such a relation. First, it induces a relation $\hat{\mathcal{R}} \subseteq S \times D(S)$ between states and distributions: $\hat{\mathcal{R}} := \{(s, \overline{t}) \mid s\mathcal{R}t\}$. Then we can use Definition 2 to lift $\hat{\mathcal{R}}$ to distributions. Note that when $\mathcal{R}$ is an equivalence relation over S, the lifted relation over $D(S)$ coincides with the lifting defined in [15].

In Definition 2, linearity tells us how to compare two linear combinations of distributions. Sometimes we need a dual notion of decomposition. Intuitively, if a relation $\mathcal{R}$ is *left-decomposable* and $\mu\mathcal{R}\nu$, then for any decomposition of μ there exists some corresponding decomposition of ν.

▶ **Definition 3** (Left-decomposable). A binary relation over distributions, $\mathcal{R} \subseteq D(S) \times D(S)$, is called *left-decomposable* if $(\sum_{i\in I} p_i \cdot \mu_i)\mathcal{R}\nu$ implies that ν can be written as $(\sum_{i\in I} p_i \cdot \nu_i)$ such that $\mu_i \mathcal{R} \nu_i$ for every $i \in I$.

The next lemma shows that any lifted relation is left-decomposable.

▶ **Lemma 4** ([6]). *For any $\mathcal{R} \subseteq S \times D(S)$ or $S \times S$, the lifted relation over distributions is left-decomposable.*

With the help of lifted relations, we are now able to define various (weak) transitions between distributions for a pLTS.

▶ **Definition 5.** Given a pLTS $\langle S, \mathsf{Act}, \longrightarrow\rangle$, we define the following transitions over distributions:

1. $\xrightarrow{\hat{\tau}}$. Let $s \xrightarrow{\hat{\tau}} \mu$ if either $s \xrightarrow{\tau} \mu$ or $\mu = \overline{s}$, and lift it to distributions;
2. $\xrightarrow{\hat{\alpha}}$ for $\alpha \neq \tau$. Let $s \xrightarrow{\hat{\alpha}} \mu$ if $s \xrightarrow{\alpha} \mu$, and lift it to distributions;
3. $\stackrel{\hat{\tau}}{\Longrightarrow}$. Let $\stackrel{\hat{\tau}}{\Longrightarrow} = (\xrightarrow{\hat{\tau}})^*$ be the reflexive and transitive closure of $\xrightarrow{\hat{\tau}}$;
4. $\stackrel{\hat{\alpha}}{\Longrightarrow}$ for $\alpha \neq \tau$. Let $\stackrel{\hat{\alpha}}{\Longrightarrow} = \stackrel{\hat{\tau}}{\Longrightarrow}\xrightarrow{\hat{\alpha}}\stackrel{\hat{\tau}}{\Longrightarrow}$. For point distributions, we often write $s \stackrel{\hat{\alpha}}{\Longrightarrow} \nu$ instead of $\overline{s} \stackrel{\hat{\alpha}}{\Longrightarrow} \nu$.

Note that here $\stackrel{\hat{\alpha}}{\Longrightarrow}$ is not a lifted transition. However, the next lemma shows that it is still both linear and left-decomposable.

▶ **Lemma 6** ([6]). *The transition relations $\stackrel{\hat{\alpha}}{\Longrightarrow}$ are both linear and left-decomposable.*

3 qCCS: Syntax and Semantics

In this section, we review the syntax and semantics of qCCS, a quantum extension of value-passing CCS introduced in [10, 27], and a notion of state-based bisimulation for qCCS processes presented in [6]. We assume the readers are familiar with the basic notions in quantum information theory; for those who are not, please refer to [23].

3.1 Syntax

We assume three types of data in qCCS: Bool for booleans, real numbers Real for classical data, and qubits Qbt for quantum data. Let $cVar$, ranged over by $x, y, \ldots$, be the set of classical variables, and $qVar$, ranged over by $q, r, \ldots$, the set of quantum variables. It is assumed that $cVar$ and $qVar$ are both countably infinite. We assume a set Exp, which includes $cVar$ as a subset and is ranged over by $e, e', \ldots$, of classical expressions over Real, and a set of boolean-valued expressions $BExp$, ranged over by $b, b', \ldots$, with the usual set of boolean operators tt, ff, $\neg$, $\wedge$, $\vee$, and $\rightarrow$. In particular, we let $e \bowtie e'$ be a boolean expression for any $e, e' \in Exp$ and $\bowtie \in \{>, <, \geq, \leq, =\}$. We further assume that only classical variables can occur free in data expressions and boolean expressions. Let $cChan$ be the set of classical channel names, ranged over by $c, d, \ldots$, and $qChan$ the set of quantum channel names, ranged over by $\mathsf{c}, \mathsf{d}, \ldots$. Let $Chan = cChan \cup qChan$. A relabeling function f is a one-to-one function from $Chan$ to $Chan$ such that $f(cChan) \subseteq cChan$ and $f(qChan) \subseteq qChan$.

We often abbreviate the indexed set $\{q_1, \ldots, q_n\}$ to $\tilde{q}$ when $q_1, \ldots, q_n$ are distinct quantum variables and the dimension n is understood. Sometimes we also use $\tilde{q}$ to denote the string $q_1 \ldots q_n$. We assume a set of process constant schemes, ranged over by $A, B, \ldots$. Assigned to each process constant scheme A there are two non-negative integers $ar_c(A)$ and $ar_q(A)$. If $\tilde{x}$ is a tuple of classical variables with $|\tilde{x}| = ar_c(A)$, and $\tilde{q}$ a tuple of distinct quantum variables with $|\tilde{q}| = ar_q(A)$, then $A(\tilde{x}, \tilde{q})$ is called a process constant. When $ar_c(A) = ar_q(A) = 0$, we also denote by A the (unique) process constant produced by A.

The syntax of qCCS terms can be given by the Backus-Naur form as

$$\begin{aligned} t &::= \mathbf{nil} \mid A(\tilde{e}, \tilde{q}) \mid \alpha.t \mid t + t \mid t \| t \mid t \backslash L \mid t[f] \mid \mathbf{if}\ b\ \mathbf{then}\ t \\ \alpha &::= \tau \mid c?x \mid c!e \mid \mathsf{c}?q \mid \mathsf{c}!q \mid \mathcal{E}[\tilde{q}] \mid M[\tilde{q}; x] \end{aligned}$$

where $c \in cChan$, $x \in cVar$, $\mathsf{c} \in qChan$, $q \in qVar$, $\tilde{q} \subseteq qVar$, $e \in Exp$, $\tilde{e} \subseteq Exp$, τ is the silent action, A is a process constant scheme, f is a relabeling function, $L \subseteq Chan$, $b \in BExp$, $\mathcal{E}$ and M are respectively a super-operator and a quantum measurement applying on the Hilbert space associated with the systems $\tilde{q}$.

To exclude quantum processes which are not physically implementable, we also require $q \notin qv(t)$ in $\mathsf{c}!q.t$ and $qv(t) \cap qv(u) = \emptyset$ in $t \| u$, where for a process term t, $qv(t)$ is the set of its free quantum variables which are not bound by quantum input $\mathsf{c}?q$. The notion of free classical variables in quantum processes can be defined in the usual way with the only modification that the quantum measurement prefix $M[\tilde{q}; x]$ has binding power on x. A quantum process term t is closed if it contains no free classical variables, *i.e.*, $fv(t) = \emptyset$. We let $\mathcal{T}$, ranged over by $t, u, \cdots$, be the set of all qCCS terms, and $\mathcal{P}$, ranged over by $P, Q, \cdots$, the set of closed terms. To complete the definition of qCCS syntax, we assume that for each process constant $A(\tilde{x}, \tilde{q})$, there is a defining equation $A(\tilde{x}, \tilde{q}) := t$ where $fv(t) \subseteq \tilde{x}$ and $qv(t) \subseteq \tilde{q}$. Throughout the paper we implicitly assume the convention that process terms are identified up to α-conversion.

The process constructs we give here are quite similar to those in classical CCS, and they also have similar intuitive meanings: **nil** stands for a process which does not perform

any action; $c?x$ and $c!e$ are respectively classical input and classical output, while $\mathsf{c}?q$ and $\mathsf{c}!q$ are their quantum counterparts. $\mathcal{E}[\tilde{q}]$ denotes the action of performing the quantum operation $\mathcal{E}$ on the qubits $\tilde{q}$ while $M[\tilde{q};x]$ measures the qubits $\tilde{q}$ according to M and stores the measurement outcome into the classical variable x. $+$ models nondeterministic choice: $t+u$ behaves like either t or u depending on the choice of the environment. $\|$ denotes the usual parallel composition. The operators $\backslash L$ and $[f]$ model restriction and relabeling, respectively: $t\backslash L$ behaves like t but any action through the channels in L is forbidden, and $t[f]$ behaves like t where each channel name is replaced by its image under the relabeling function f. Finally, **if** b **then** t is the standard conditional choice where t can be executed only if b evaluates to tt.

An evaluation ψ is a function from $cVar$ to Real; it can be extended in an obvious way to functions from Exp to Real and from $BExp$ to $\{\mathrm{tt},\mathrm{ff}\}$, and finally, from $\mathcal{T}$ to $\mathcal{P}$. For simplicity, we still use ψ to denote these extensions. Let $\psi\{v/x\}$ be the evaluation which differs from ψ only in that it maps x to v.

3.2 Transitional semantics

For each quantum variable $q \in qVar$, we assume a 2-dimensional Hilbert space $\mathcal{H}_q$ to be the state space of the q-system. For any $V \subseteq qVar$, we denote $\mathcal{H}_V = \bigotimes_{q\in V} \mathcal{H}_q$. In particular, $\mathcal{H} = \mathcal{H}_{qVar}$ is the state space of the whole environment consisting of all the quantum variables. Note that $\mathcal{H}$ is a countably-infinite dimensional Hilbert space. For any $V \subseteq qVar$ we denote by $\overline{V}$ the complement set of V in $qVar$.

Suppose P is a closed quantum process. A pair of the form $\langle P,\rho\rangle$ is called a configuration, where $\rho \in \mathcal{D}(\mathcal{H})$ is a density operator on $\mathcal{H}$.[1] The set of configurations is denoted by Con, and ranged over by $\mathcal{C},\mathcal{D},\ldots$. Let

$$\mathsf{Act} \quad = \quad \{\tau\} \cup \{c?v, c!v \mid c \in cChan, v \in \mathsf{Real}\} \cup \{\mathsf{c}?r, \mathsf{c}!r \mid \mathsf{c} \in qChan, r \in qVar\}.$$

For each $\alpha \in \mathsf{Act}$, we define the bound quantum variables $qbv(\alpha)$ of α as $qbv(\mathsf{c}?r) = \{r\}$ and $qbv(\alpha) = \emptyset$ if α is not a quantum input. The channel names used in action α is denoted by $cn(\alpha)$; that is, $cn(c?v) = cn(c!v) = \{c\}$, $cn(\mathsf{c}?r) = cn(\mathsf{c}!r) = \{\mathsf{c}\}$, and $cn(\tau) = \emptyset$. We also extend the relabelling function to Act in an obvious way. Then the transitional semantics of qCCS can be given by a pLTS $\langle Con, \mathsf{Act}, \longrightarrow\rangle$, where $\longrightarrow \subseteq Con \times \mathsf{Act} \times D(Con)$ is the smallest relation satisfying the inference rules depicted in Fig. 1. The symmetric forms for rules Par, Com_C, Com_Q, and Sum are omitted. We abuse the notation slightly by writing $\mathcal{C} \xrightarrow{\alpha} \mathcal{D}$ if $\mathcal{C} \xrightarrow{\alpha} \overline{\mathcal{D}}$. We also use the obvious extension of the function $\|$ on configurations to distributions. To be precise, if $\mu = \sum_{i\in I} p_i\langle P_i,\rho_i\rangle$ then $\mu\|Q$ denotes the distribution $\sum_{i\in I} p_i\langle P_i\|Q,\rho_i\rangle$. Similar extension applies to $\mu[f]$ and $\mu\backslash L$.

3.3 State-based bisimulation

In this subsection, we recall the basic definitions and properties of the state-based bisimulation introduced in [6]. Let $\mathcal{C} = \langle P,\rho\rangle$ be a configuration and $\mathcal{E}$ a super-operator. We denote $qv(\mathcal{C}) = qv(P)$, $\mathrm{env}(\mathcal{C}) = \mathrm{tr}_{qv(P)}(\rho)$ being the quantum *environment* of process P in $\mathcal{C}$, and $\mathcal{E}(\mathcal{C}) = \langle P,\mathcal{E}(\rho)\rangle$. Furthermore, for distribution $\mu = \sum_i p_i\mathcal{C}_i$ with $p_i > 0$ for each i, we write $qv(\mu) = \bigcup_i qv(\mathcal{C}_i)$, $\mathrm{env}(\mu) = \sum_i p_i \cdot \mathrm{env}(\mathcal{C}_i)$, and $\mathcal{E}(\mu) = \sum_i p_i\mathcal{E}(\mathcal{C}_i)$. For any $V \subseteq qVar$, denote by $SO(\mathcal{H}_V)$ the set of super-operators on $\mathcal{H}_V$.

[1] As $\mathcal{H}$ is infinite dimensional, ρ should be understood as a density operator on some finite dimensional subspace of $\mathcal{H}$ which contains $\mathcal{H}_{qv(P)}$.

$$Tau\quad \frac{}{\langle \tau.P,\rho\rangle \xrightarrow{\tau} \langle P,\rho\rangle} \qquad Inp_C\quad \frac{v\in \mathsf{Real}}{\langle c?x.t,\rho\rangle \xrightarrow{c?v} \langle t\{v/x\},\rho\rangle}$$

$$Out_C\quad \frac{v=[\![e]\!]}{\langle c!e.P,\rho\rangle \xrightarrow{c!v} \langle P,\rho\rangle} \qquad Inp_Q\quad \frac{r\notin qv(\mathsf{c}?q.P)}{\langle \mathsf{c}?q.P,\rho\rangle \xrightarrow{c?r} \langle P\{r/q\},\rho\rangle}$$

$$Out_Q\quad \frac{}{\langle \mathsf{c}!q.P,\rho\rangle \xrightarrow{c!q} \langle P,\rho\rangle} \qquad Oper\quad \frac{}{\langle \mathcal{E}[\tilde{r}].P,\rho\rangle \xrightarrow{\tau} \langle P,\mathcal{E}_{\tilde{r}}(\rho)\rangle}$$

$$Meas\quad \frac{M=\sum_{i\in I}\lambda_i E^i,\ p_i=\mathrm{tr}(E^i_{\tilde{r}}\rho)>0}{\langle M[\tilde{r};x].P,\rho\rangle \xrightarrow{\tau} \sum_{i\in I} p_i\langle P\{\lambda_i/x\},E^i_{\tilde{r}}\rho E^i_{\tilde{r}}/p_i\rangle} \qquad Par\quad \frac{\langle P_1,\rho\rangle \xrightarrow{\alpha} \mu,\ qbv(\alpha)\cap qv(P_2)=\emptyset}{\langle P_1\|P_2,\rho\rangle \xrightarrow{\alpha} \mu\|P_2}$$

$$Com_C\quad \frac{\langle P_1,\rho\rangle \xrightarrow{c?v} \langle P_1',\rho\rangle,\ \langle P_2,\rho\rangle \xrightarrow{c!v} \langle P_2',\rho\rangle}{\langle P_1\|P_2,\rho\rangle \xrightarrow{\tau} \langle P_1'\|P_2',\rho\rangle} \qquad Com_Q\quad \frac{\langle P_1,\rho\rangle \xrightarrow{\mathsf{c}?r} \langle P_1',\rho\rangle,\quad \langle P_2,\rho\rangle \xrightarrow{\mathsf{c}!r} \langle P_2',\rho\rangle}{\langle P_1\|P_2,\rho\rangle \xrightarrow{\tau} \langle P_1'\|P_2',\rho\rangle}$$

$$Sum\quad \frac{\langle P,\rho\rangle \xrightarrow{\alpha} \mu}{\langle P+Q,\rho\rangle \xrightarrow{\alpha} \mu} \qquad Rel\quad \frac{\langle P,\rho\rangle \xrightarrow{\alpha} \mu}{\langle P[f],\rho\rangle \xrightarrow{f(\alpha)} \mu[f]}$$

$$Cho\quad \frac{\langle P,\rho\rangle \xrightarrow{\alpha} \mu,\ [\![b]\!]=\mathrm{tt}}{\langle \textbf{if } b \textbf{ then } P,\rho\rangle \xrightarrow{\alpha} \mu} \qquad Res\quad \frac{\langle P,\rho\rangle \xrightarrow{\alpha} \mu,\ cn(\alpha)\cap L=\emptyset}{\langle P\backslash L,\rho\rangle \xrightarrow{\alpha} \mu\backslash L}$$

$$Def\quad \frac{\langle t\{\tilde{v}/\tilde{x},\tilde{r}/\tilde{q}\},\rho\rangle \xrightarrow{\alpha} \mu,\ A(\tilde{x},\tilde{q}):=t,\ \tilde{v}=[\![\tilde{e}]\!]}{\langle A(\tilde{e},\tilde{r}),\rho\rangle \xrightarrow{\alpha} \mu}$$

Figure 1 Transitional semantics of qCCS.

▶ **Definition 7.** A relation $\mathcal{R}\subseteq Con\times Con$ is closed under super-operator application if $\mathcal{C}\mathcal{R}\mathcal{D}$ implies $\mathcal{E}(\mathcal{C})\mathcal{R}\mathcal{E}(\mathcal{D})$ for all $\mathcal{E}\in SO(\mathcal{H}_{\overline{qv(\mathcal{C})\cup qv(\mathcal{D})}})$. More generally, a relation $\mathcal{R}\subseteq D(Con)\times D(Con)$ is closed under super-operator application if $\mu\mathcal{R}\nu$ implies $\mathcal{E}(\mu)\mathcal{R}\mathcal{E}(\nu)$ for all $\mathcal{E}\in SO(\mathcal{H}_{\overline{qv(\mu)\cup qv(\nu)}})$.

▶ **Definition 8.**

1. A symmetric relation $\mathcal{R}\subseteq Con\times Con$ is called a *state-based ground bisimulation* if $\mathcal{C}\mathcal{R}\mathcal{D}$ implies that
   - **(i)** $qv(\mathcal{C})=qv(\mathcal{D})$, and $\mathrm{env}(\mathcal{C})=\mathrm{env}(\mathcal{D})$,
   - **(ii)** whenever $\mathcal{C}\xrightarrow{\alpha}\mu$, there exists ν such that $\mathcal{D}\overset{\hat{\alpha}}{\Longrightarrow}\nu$ and $\mu\mathcal{R}\nu$.
2. A relation $\mathcal{R}$ is a *state-based bisimulation* if it is a state-based ground bisimulation, and is closed under super-operator application.
3. Two quantum configurations $\mathcal{C}$ and $\mathcal{D}$ are state-based bisimilar, denoted by $\mathcal{C}\approx_s\mathcal{D}$, if there exists a state-based bisimulation $\mathcal{R}$ such that $\mathcal{C}\mathcal{R}\mathcal{D}$;
4. Two quantum process terms t and u are state-based bisimilar, denoted by $t\approx_s u$, if for any quantum state $\rho\in\mathcal{D}(\mathcal{H})$ and any evaluation ψ, $\langle t\psi,\rho\rangle\approx_s\langle u\psi,\rho\rangle$.

Note that in Clause 1.(ii) of the above definition, $\mu\mathcal{R}\nu$ means μ and ν are related by the relation lifted from $\mathcal{R}$. The following theorem is taken from [6].

▶ **Theorem 9.**

1. *The bisimilarity relation $\approx_s$ is the largest state-based bisimulation on Con, and it is an equivalence relation.*
2. *As a lifted relation on $D(Con)$, $\approx_s$ is both linear and left-decomposable.*

4 Distribution-based bisimulation

Note that in [8], it has already been shown by examples that state-based bisimulation is sometimes too discriminative for probabilistic automata. These examples certainly work for quantum processes as well. Furthermore, as the following example indicates, the problem becomes more serious in the quantum setting, as the accompanied quantum states can and should be combined when simulating each other.

▶ **Example 10.** Let $M = \lambda_0|0\rangle\langle 0| + \lambda_1|1\rangle\langle 1|$ be a two-outcome measurement according to the computational basis, and $\mathcal{E}$ a super-operator with the Kraus operators being $|0\rangle\langle 0|$ and $|1\rangle\langle 1|$. Let ρ be a density operator on $\mathcal{H}_{\overline{\{q\}}}$, and $\mathcal{C} := \langle M[q;x].\mathbf{nil}, |+\rangle_q\langle +| \otimes \rho\rangle$ and $\mathcal{D} := \langle \mathcal{E}[q].\mathbf{nil}, |+\rangle_q\langle +| \otimes \rho\rangle$ be two configurations where $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$. Note that in the process $M[q;x].\mathbf{nil}$, the measurement outcome is never used (as $x \notin fv(\mathbf{nil})$), while the effect of $\mathcal{E}[q]$ is exactly measuring the system q according to M, but ignoring the measurement outcome. Thus we definitely would like to regard $\mathcal{C}$ and $\mathcal{D}$ as being bisimilar[2].

However, we can show that $\mathcal{C} \not\approx_s \mathcal{D}$. Let $\mathcal{C}_0 = \langle \mathbf{nil}, |0\rangle_q\langle 0| \otimes \rho\rangle$, $\mathcal{C}_1 = \langle \mathbf{nil}, |1\rangle_q\langle 1| \otimes \rho\rangle$, $\mathcal{C}_I = \langle \mathbf{nil}, I_q/2 \otimes \rho\rangle$, and $\mu = \frac{1}{2}\mathcal{C}_0 + \frac{1}{2}\mathcal{C}_1$. Then obviously $\mu \not\approx_s \overline{\mathcal{C}_I}$, as otherwise by the left-decompositivity of $\approx_s$ we must have both $\mathcal{C}_0 \approx_s \mathcal{C}_I$ and $\mathcal{C}_1 \approx_s \mathcal{C}_I$, which is impossible.

Actually, the argument in Example 10 applies to *any* bisimulation which is state-based: by Lemma 4, any bisimilation between distributions which is lifted from configurations is left-decomposable, hence discriminating $\mathcal{C}$ and $\mathcal{D}$. Therefore, to make these two obviously indistinguishable configurations bisimilar, we have to define bisimulation relation *directly* on distributions, rather than on configurations and then lift it to distributions.

For this purpose, we extend the distribution-based bisimulation introduced in [9] to our quantum setting. A distribution μ is said to be *transition consistent*, if for any $\mathcal{C} \in \lceil\mu\rceil$ and $\alpha \neq \tau$, $\mathcal{C} \stackrel{\hat{\alpha}}{\Longrightarrow} \nu_{\mathcal{C}}$ for some $\nu_{\mathcal{C}}$ implies $\mu \stackrel{\hat{\alpha}}{\Longrightarrow} \nu$ for some ν, i.e., all configurations in its support have the same set of enabled visible actions (possibly after some invisible transitions). Furthermore, a decomposition $\mu = \sum_{i\in I} p_i \cdot \mu_i$, $p_i > 0$ for each $i \in I$, is a *tc-decomposition* of μ if for each $i \in I$, μ_i is transition consistent.

▶ **Definition 11.**

1. A symmetric relation $\mathcal{R} \subseteq D(Con) \times D(Con)$ is called a *(distribution-based) ground bisimulation* if for any $\mu, \nu \in D(Con)$, $\mu\mathcal{R}\nu$ implies that
   - **(i)** $qv(\mu) = qv(\nu)$, and $\mathrm{env}(\mu) = \mathrm{env}(\nu)$,
   - **(ii)** whenever $\mu \xrightarrow{\hat{\alpha}} \mu'$, there exists ν' such that $\nu \stackrel{\hat{\alpha}}{\Longrightarrow} \nu'$ and $\mu'\mathcal{R}\nu'$,
   - **(iii)** if μ is not transition consistent, and $\mu = \sum_{i\in I} p_i \cdot \mu_i$ is a tc-decomposition, then $\nu \stackrel{\hat{\tau}}{\Longrightarrow} \sum_{i\in I} p_i \cdot \nu_i$ such that for each i, $\mu_i\mathcal{R}\nu_i$.
2. A relation $\mathcal{R}$ is a *(distribution-based) bisimulation* if it is a ground bisimulation, and is closed under super-operator application.

In contrast with Definition 8.1, the above definition has an additional requirement Clause 1.(iii). This clause is crucial for distribution-based bisimulation, as the transition $\mu \xrightarrow{\hat{\alpha}} \mu'$ in Clause 1.(ii) is possible only when μ is transition consistent for α. That is, all configurations in the support of μ can perform weak α-transition. For those actions for which μ is not

[2] Note that $\mathcal{C}$ and $\mathcal{D}$ would be regarded as 'semantically identical' in [17], instead of '(distribution-based) bisimilar' as we do in this paper, since the semantics of $M[q;x]$ in this case is represented as $\mathcal{E}[q]$ *by definition*.

transition consistent, we must first split μ into transition consistent components, and then compare them with the corresponding components of ν individually.

The bisimilarity $\approx$ for quantum configurations and for quantum process terms are defined similarly as in the state-based case. The next theorem collects some useful properties of the distribution-based bisimilarity.

▶ **Theorem 12.**
1. *The bisimilarity relation $\approx$ is the largest bisimulation on $D(Con)$, and it is an equivalence relation.*
2. *$\approx$ is linear, but not left-decomposable.*

A direct consequence of Theorem 12 is a deciding algorithm for the bisimilarity between recursion-free quantum configurations, which is sufficient for most practical quantum cryptographic protocols. First, as pointed out in [12], any recursion-free quantum processes can be modified to be free of quantum input, so that the bisimilarity between them can be verified by only examining the ground bisimulation. Second, it has been proved in [14, Lemma 1] that every linear bisimulation $\mathcal{R}$ corresponds to a matrix E, so that two distributions μ and ν are related by $\mathcal{R}$ if and only if $(\mu - \nu)E = 0$, where distributions are seen as vectors. As our ground bisimulation for quantum processes is indeed linear, the algorithm presented in [14], with slight changes, can be used to decide it. For the sake of space limit, we omit the details here, and refer interested readers to [14].

To conclude this section, we would like to show that our distribution-based bisimulation is weaker than its state-based counterpart presented in Definition 8.

▶ **Theorem 13.** *Let $\mu, \nu \in D(Con)$. Then $\mu \approx_s \nu$ implies $\mu \approx \nu$, but $\mu \approx \nu$ does not necessarily imply $\mu \approx_s \nu$. In particular, we have in Example 10 that $\mu \approx \overline{\mathcal{C}_I}$ and $\mathcal{C} \approx \mathcal{D}$.*

5 Bisimulation metric

In the previous section, only *exact* bisimulation is presented where two quantum processes are either bisimilar or non-bisimilar. Obviously, such a bisimulation cannot capture the idea that a quantum process *approximately* implements its specification. To measure the behavioural distance between processes, the notion of approximate bisimulation and the bisimulation distance for qCCS processes were introduced in [27]. This section is devoted to extending this approximate bisimulation to distribution-based case. Note that approximate bisimulation has been investigated in probabilistic process algebra and probabilistic labelled transition systems in the context of security analysis [24, 1].

Recall that the trace distance of $\rho, \sigma \in \mathcal{D}(\mathcal{H})$ is defined to be $d(\rho, \sigma) = \frac{1}{2}\|\rho - \sigma\|_{\mathrm{tr}}$ where $\|\cdot\|_{\mathrm{tr}}$ denotes the trace norm. We have the following definition.

▶ **Definition 14.** Given $\lambda \in [0, 1]$, a symmetric relation $\mathcal{R}$ over $D(Con)$ which is closed under super-operator application is a λ-bisimulation if for any $\mu \mathcal{R} \nu$, we have
1. $qv(\mu) = qv(\nu)$, and $d(\mathrm{env}(\mu), \mathrm{env}(\nu)) \leq \lambda$,
2. whenever $\mu \xrightarrow{\hat{\alpha}} \mu'$, there exists ν' such that $\nu \stackrel{\hat{\alpha}}{\Longrightarrow} \nu'$ and $\mu' \mathcal{R} \nu'$,
3. if μ is not transition consistent, and $\mu = \sum_{i \in I} p_i \cdot \mu_i$ is a tc-decomposition, then $\nu \stackrel{\hat{\tau}}{\Longrightarrow} \sum_{i \in I} p_i \cdot \nu_i$ such that $\sum_{i : \mu_i \mathcal{R} \nu_i} p_i \geq 1 - \lambda$.

By induction, we can show easily that $\mu \xrightarrow{\hat{\alpha}} \mu'$ can be replaced by $\mu \stackrel{\hat{\alpha}}{\Longrightarrow} \mu'$ in Clause (2).

The approximate bisimilarity $\overset{\lambda}{\approx}$ for quantum configurations and for quantum process terms are defined similarly as in the exact bisimulation case. Furthermore, we define the bisimulation

distance between distributions as $d_b(\mu,\nu) = \inf\{\lambda \geq 0 \mid \mu \overset{\lambda}{\approx} \nu\}$ and the bisimulation distance between process terms as $d_b(t,u) = \inf\{\lambda \geq 0 \mid \forall \psi \text{ and } \rho \in \mathcal{D}(\mathcal{H}), \langle t\psi, \rho\rangle \overset{\lambda}{\approx} \langle u\psi, \rho\rangle\}$. Here we assume that $\inf \emptyset = 1$. The next theorem shows that d_b is indeed a pseudo-metric with $\approx$ being its kernel.

▶ **Theorem 15.**

1. *The bisimulation distance d_b is a pseudo-metric on $D(Con)$.*
2. *For any $\mu, \nu \in D(Con)$, $\mu \approx \nu$ if and only if $d_b(\mu,\nu) = 0$.*

6 An illustrative example

For the ease of notations, we extend the syntax of qCCS a little bit by allowing probabilistic choice in the syntax level[3]; that is, we assume $\sum_{i\in I} p_i t_i \in \mathcal{T}$ whenever $t_i \in \mathcal{T}$ and $p_i \geq 0$ for each $i \in I$ with $\sum_{i\in I} p_i = 1$. We further extend the transitional semantics in Fig. 1 by adding the following transition rule:

$$Dist\ \frac{}{\langle \sum_{i\in I} p_i t_i, \rho\rangle \xrightarrow{\tau} \sum_{i\in I} p_i \langle t_i, \rho\rangle}.$$

We also introduce the syntax sugar **if** b **then** t **else** u to be the abbreviation of **if** b **then** t + **if** $\neg b$ **then** u.

BB84, the first quantum key distribution protocol developed by Bennett and Brassard in 1984 [4], provides a provably secure way to create a private key between two parties, say, Alice and Bob, with the help of a classical authenticated channel and a quantum insecure channel between them. Its security relies on the basic property of quantum mechanics that information gain about a quantum state is only possible at the expense of changing the state, if all the possible states are not orthogonal. The basic BB84 protocol with security parameter n goes as follows:

(1) Alice randomly generates two strings $\tilde{B}_a$ and $\tilde{K}_a$ of bits, each with size n.
(2) Alice prepares a string of qubits $\tilde{q}$, with size n, such that the ith qubit of $\tilde{q}$ is $|x_y\rangle$ where x and y are the ith bits of $\tilde{B}_a$ and $\tilde{K}_a$, respectively, and $|0_0\rangle = |0\rangle$, $|0_1\rangle = |1\rangle$, $|1_0\rangle = |+\rangle$, and $|1_1\rangle = |-\rangle$. Here $|+\rangle := (|0\rangle + |1\rangle)/\sqrt{2}$ and $|-\rangle := (|0\rangle - |1\rangle)/\sqrt{2}$.
(3) Alice sends the qubit string $\tilde{q}$ to Bob.
(4) Bob randomly generates a string of bits $\tilde{B}_b$ with size n.
(5) Bob measures each qubit received from Alice according to a basis determined by the bits he generated: if the ith bit of $\tilde{B}_b$ is k then he measures with $\{|k_0\rangle, |k_1\rangle\}$, $k = 0, 1$. Let the measurement results be $\tilde{K}_b$, again a string of bits with size n.
(6) Bob sends his measurement bases $\tilde{B}_b$ back to Alice, and upon receiving the information, Alice sends her bases $\tilde{B}_a$ to Bob.
(7) Alice and Bob determine at which positions the bit strings $\tilde{B}_a$ and $\tilde{B}_b$ are equal. They discard the bits in $\tilde{K}_a$ and $\tilde{K}_b$ where the corresponding bits of $\tilde{B}_a$ and $\tilde{B}_b$ do not match.

After the execution of the basic BB84 protocol above, the remaining bits of $\tilde{K}_a$ and $\tilde{K}_b$, denoted by $\tilde{K}'_a$ and $\tilde{K}'_b$ respectively, should be the same, provided that the channels used are perfect, and no eavesdropper exists.

To detect a potential eavesdropper Eve, Alice and Bob proceed as follows:

[3] Note that this extension will not change the expressive power of qCCS and all the results obtained in this paper, as probabilistic choices can be simulated by quantum measurements preceded by appropriate quantum state preparation.

(8) Alice randomly chooses $\lceil|\tilde{K}'_a|/2\rceil$ bits of $\tilde{K}'_a$, denoted by $\tilde{K}''_a$, and sends to Bob $\tilde{K}''_a$ and its indexes in $\tilde{K}'_a$.

(9) Upon receiving the information from Alice, Bob sends back to Alice his substring $\tilde{K}''_b$ of $\tilde{K}'_b$ at the indexes received from Alice.

(10) Alice and Bob check if the strings $\tilde{K}''_a$ and $\tilde{K}''_b$ are equal. If yes, then the remaining substring $\tilde{K}^f_a$ (resp. $\tilde{K}^f_b$) of $\tilde{K}'_a$ (resp. $\tilde{K}'_b$) by deleting $\tilde{K}''_a$ (resp. $\tilde{K}''_b$) is the secure key shared by Alice (reps. Bob). Otherwise, an eavesdropper (or too much noise in the channels) is detected, and the protocol halts without generating any secure keys.

For simplicity, we omit the processes of information reconciliation and privacy amplification. Now we describe the basic BB84 protocol [Steps (1)–(7)] in qCCS as follows.

$$\begin{aligned}
Alice(n) &:= \sum_{\tilde{B}_a,\tilde{K}_a\in\{0,1\}^n} \frac{1}{2^{2n}} Set_{\tilde{K}_a}[\tilde{q}].H_{\tilde{B}_a}[\tilde{q}].\mathsf{A2B}!\tilde{q}.Wait_A(\tilde{B}_a,\tilde{K}_a)\\
Wait_A(\tilde{B}_a,\tilde{K}_a) &:= b2a?\tilde{B}_b.a2b!\tilde{B}_a.key_a!cmp(\tilde{K}_a,\tilde{B}_a,\tilde{B}_b).\mathbf{nil}\\
Bob(n) &:= \mathsf{A2B}?\tilde{q}.\sum_{\tilde{B}_b\in\{0,1\}^n} \frac{1}{2^n} M_{\tilde{B}_b}[\tilde{q};\tilde{K}_b].Set_{\tilde{0}}[\tilde{q}].b2a!\tilde{B}_b.Wait_B(\tilde{B}_b,\tilde{K}_b)\\
Wait_B(\tilde{B}_b,\tilde{K}_b) &:= a2b?\tilde{B}_a.key_b!cmp(\tilde{K}_b,\tilde{B}_a,\tilde{B}_b).\mathbf{nil}\\
BB84(n) &:= Alice(n)\|Bob(n)
\end{aligned}$$

where $Set_{\tilde{K}_a}[\tilde{q}]$ sets the ith qubit of $\tilde{q}$ to the state $|\tilde{K}_a(i)\rangle$, $H_{\tilde{B}_a}[\tilde{q}]$ applies H or does nothing on the ith qubit of $\tilde{q}$ depending on whether the ith bit of $\tilde{B}_a$ is 1 or 0, and $M_{\tilde{B}_b}[\tilde{q};\tilde{K}_b]$ is the quantum measurement on $\tilde{q}$ according to the bases determined by $\tilde{B}_b$, i.e., for each $1\le i\le n$, it measures q_i with respect to the basis $\{|0\rangle,|1\rangle\}$ (resp. $\{|+\rangle,|-\rangle\}$) if $\tilde{B}_b(i)=0$ (resp. 1), and stores the result into $\tilde{K}_b(i)$. The function cmp takes a triple of bit-strings $\tilde{x},\tilde{y},\tilde{z}$ with the same size as inputs, and returns the substring of $\tilde{x}$ where the corresponding bits of $\tilde{y}$ and $\tilde{z}$ match. When $\tilde{y}$ and $\tilde{z}$ match nowhere, we let $cmp(\tilde{x},\tilde{y},\tilde{z})=\epsilon$, the empty string. We add the operation $Set_{\tilde{0}}[\tilde{q}]$ in $Bob(n)$ for technical reasons: it makes the ideal specifications defined below simple.

To show the correctness of basic BB84 protocol, we first put $BB84(n)$ in a *test environment* defined as follows

$$\begin{aligned}
Test &:= key_a?k_a.key_b?k_b.\mathbf{if}\ k_a=k_b\ \mathbf{then}\ key!k_a.\mathbf{nil}\ \mathbf{else}\ fail!0.\mathbf{nil}\\
BB84_{test}(n) &:= (BB84(n)\|Test)\backslash\{a2b,b2a,\mathsf{A2B},key_a,key_b\}
\end{aligned}$$

For the ideal *specification* of $BB84_{test}(n)$, we would like it to satisfy the following three conditions: (1) it is correct, in the sense that it will never perform $fail!0$; (2) the generated key $\tilde{x}$ with $|\tilde{x}|=i$ is uniformly distributed for each $i\le n$. That is, for any $\tilde{x}$ with $|\tilde{x}|=i$, $\Pr(\tilde{x}$ is the key obtained $|$ key-length $=i)=1/2^i$; (3) The length of the obtained key follows the unbiased binomial distribution. That is, for each $i\le n$, $\Pr(\text{key-length}=i)=\binom{n}{i}/2^n$. Thus we can let

$$BB84_{spec}(n) := \sum_{i=0}^{n}\sum_{\tilde{x}\in\{0,1\}^i}\frac{\binom{n}{i}}{2^{n+i}} Set_{\tilde{0}}[\tilde{q}].key!\tilde{x}.\mathbf{nil}.$$

It is tedious but routine to check that $BB84_{test}(n)\approx BB84_{spec}(n)$ for any n.

Now we proceed to describe the protocol that detects potential eavesdroppers [Steps

(1)–(10)]. Let

$$\begin{aligned}
Alice'(n) &:= (Alice(n)\|key_a?\tilde{K}'_a.\sum_{\tilde{x}\subseteq\{1,\ldots,m\}}^{|\tilde{x}|=k}\frac{1}{\binom{m}{k}}a2b!\tilde{x}.a2b!SubStr(\tilde{K}'_a,\tilde{x}).b2a?\tilde{K}''_b.\\
&\quad (\textbf{if } SubStr(\tilde{K}'_a,\tilde{x})=\tilde{K}''_b \textbf{ then } key'_a!RemStr(\tilde{K}'_a,\tilde{x}).\textbf{nil}))\backslash\{key_a\}\\
Bob'(n) &:= (Bob(n)\|key_b?\tilde{K}'_b.a2b?\tilde{x}.a2b?\tilde{K}''_a.b2a!SubStr(\tilde{K}'_b,\tilde{x}).\\
&\quad (\textbf{if } SubStr(\tilde{K}'_b,\tilde{x})=\tilde{K}''_a \textbf{ then } key'_b!RemStr(\tilde{K}'_b,\tilde{x}).\textbf{nil}))\backslash\{key_b\}\\
BB84'(n) &:= Alice'(n)\|Bob'(n)
\end{aligned}$$

where $m=|\tilde{K}'_a|$ and $k=\lceil m/2\rceil$, the function $SubStr(\tilde{K}'_a,\tilde{x})$ returns the substring of $\tilde{K}'_a$ at the indexes specified by $\tilde{x}$, and $RemStr(\tilde{K}'_a,\tilde{x})$ returns the remaining substring of $\tilde{K}'_a$ by deleting $SubStr(\tilde{K}'_a,\tilde{x})$.

To get a taste of the security of BB84 protocol, we consider a special case where Eve's strategy is to simply measure the qubits sent by Alice, according to randomly guessed bases, to get the keys and resend these qubits to Bob. That is, we define

$$Eve(n) := \mathsf{A2E}?\tilde{q}.\sum_{\tilde{B}_e\in\{0,1\}^n}\frac{1}{2^n}M_{\tilde{B}_e}[\tilde{q};\tilde{K}_e].key'_e!\tilde{K}_e.\mathsf{E2B}!\tilde{q}.\textbf{nil}$$

Again, we put $BB84'(n)$ in a test environment, but now the environment includes the presence of Eve:

$$\begin{aligned}
Test' &:= key'_a?\tilde{x}.key'_b?\tilde{y}.key'_e?\tilde{z}.(\textbf{if } \tilde{x}\neq\tilde{y} \textbf{ then } fail!0.\textbf{nil}\\
&\qquad\qquad \textbf{else } (\textbf{if } \tilde{x}=\tilde{z} \textbf{ then } hacked!0.\textbf{nil}))\\
BB84'_{test}(n) &:= (Alice'(n)[f_a]\|Bob'(n)[f_b]\|Eve(n)\|Test')\backslash L
\end{aligned}$$

where $L=\{a2b,b2a,\mathsf{A2E},\mathsf{E2B},key'_a,key'_b,key'_e\}$, $f_a(\mathsf{A2B})=\mathsf{A2E}$, and $f_b(\mathsf{A2B})=\mathsf{E2B}$.

Now, to show the *security* of BB84,[4] it suffices to prove the following property:

$$BB84'_{test}(n)\overset{c^n}{\approx} Set_{\tilde{0}}[\tilde{q}].\textbf{nil} \tag{1}$$

where $c=1/2+\sqrt{3}/4<1$. Thus $d_b(BB84'_{test}(n),Set_{\tilde{0}}[\tilde{q}].\textbf{nil})\leq c^n$. That is, the testing system is just like a protocol which only sets the quantum qubits $\tilde{q}$ to $|\tilde{0}\rangle\langle\tilde{0}|$. As the process $Set_{\tilde{0}}[\tilde{q}].\textbf{nil}$ never performs $fail!0$ or $hacked!0$, this indicates that the *insecurity degree* of BB84 is at most c^n, which decreases exponentially to 0 when n tends to infinity.

To show Eq.(1), take arbitrarily $\rho\in\mathcal{D}(\mathcal{H})$, and let $\mathcal{C}=\langle BB84'_{test}(n),\rho\rangle$ and $\mathcal{D}=\langle Set_{\tilde{0}}[\tilde{q}].\textbf{nil},\rho\rangle$. Basically, we only need to compute the total probability of $\mathcal{C}$ eventually performing $fail!0$ or $hacked!0$. The reason is, they are the only visible actions of $\mathcal{C}$ ($\mathcal{D}$ does not perform any visible action at all), and also the only actions which contribute to possible transition inconsistency of distributions obtained from $\mathcal{C}$. If the total probability of their appearance is upper bounded by c^n, then $\mathcal{C}$ and $\mathcal{D}$ are c^n-bisimilar.

For each qubit sent by Alice, Eve chooses the wrong basis with probability 1/2, and in this case if Bob measures this qubit according to the correct basis he will get an incorrect result with probability 1/2. Thus for each qubit that Bob guesses the correct basis, the

[4] Here we adopt a weak notion of security: by secure we mean the eavesdropper ends up with a false key string. A stronger and more practical notion of security should take into account the mutual information between the keys held by the legitimate parties and the eavesdropper. We leave the analysis of BB84 with respect to this notion of security for future work.

probability that Alice and Bob get different key bits is $1/4$. Furthermore, for each i-length raw key generated by the basic BB84, Alice and Bob will compare $i/2$ key bits during the eavesdropper-detection phase. The probability that they fail to detect the eavesdropper is then $(3/4)^{i/2}$. Note that only when the eavesdropper is not detected, the protocol proceeds. Hence the probability of observing *fail*!0 or *hacked*!0 is upper bounded by

$$\sum_{i=0}^{n} \sum_{\vec{x}\in\{0,1\}^i} \frac{\binom{n}{i}}{2^{n+i}} (3/4)^{i/2} = \frac{1}{2^n} \sum_{i=0}^{n} \binom{n}{i} (3/4)^{i/2} = c^n.$$

7 Conclusion and Future work

In this paper, we have proposed a novel notion of distribution-based bisimulation for quantum processes in qCCS. In contrast with previous bisimulations introduced in the literature, our definition is reasonably weaker in that it equates some intuitively bisimilar processes which are not bisimilar according to the previous definitions, thus is more useful in applications. We further defined a bisimulation distance to characterise the extent to which two processes are bisimilar. As an application, we applied the notions of distribution-based bisimulation and bisimulation distance to show that the quantum key distribution protocol BB84 is sound and secure against the intercept-resend attacker. To the best of our knowledge, this is the first time in the literature that the (asymptotic) security of BB84 has been analysed in the framework of a quantum process algebra.

There are still many questions remaining for further study. Firstly, as pointed out in Section 6, the notion of security we adopted for the analysis of BB84 is a rather weak one. In quantum information field, people normally use the mutual information between the states held by legitimate parties and the eavesdropper to quantify the leakage of secure information. To perform a security analysis of BB84 in terms of this stronger notion of security and against more complex model of attack beyond the intercept-resend one studied in the current paper is one of the future directions we are pursuing.

Secondly, bisimilarity checking is usually a very tedious and routine task which can barely be done by hand. This issue becomes more serious when the number of parties involved and the round of communications increase. To deal with this problem, making the process algebra approach more applicable for the analysis of general quantum cryptographic protocols, we are going to develop a software tool for automated bisimilarity checking. In the theoretical aspect, we will explore the possibility of extending symbolic bisimulation proposed in [12] to distribution-based case, to decrease the computational complexity of determining bisimilarity.

Finally, as shown in [9], distribution-based bisimulation is not a congruence in general, unless restricted to distributed schedulers. However, as argued by the authors of [9], non-distributed schedulers, which are responsible for the incongruence, are actually very unrealistic and do not appear in real-world applications. To show that our distribution-based bisimulation is a congruence for qCCS processes under distributed schedulers and to study the implication of distributed schedulers for quantum cryptographic protocols are also topics worthy of further consideration.

Acknowledgement. This work was partially supported by Australian Research Council (Grant No. DP130102764). Y. F. is also supported by the National Natural Science Foundation of China (Grant Nos. 61428208 and 61472412) and the CAS/SAFEA International Partnership Program for Creative Research Team.

References

1 A. Aldini and A. Di Pierro. Estimating the maximum information leakage. *International Journal of Information Security*, 7(3):219–242, 2008.

2 Ebrahim Ardeshir-Larijani, Simon J Gay, and Rajagopal Nagarajan. Equivalence checking of quantum protocols. In *TACAS'13*, pages 478–492. Springer, 2013.

3 Ebrahim Ardeshir-Larijani, Simon J Gay, and Rajagopal Nagarajan. Verification of concurrent quantum protocols by equivalence checking. In *TACAS'14*, pages 500–514. Springer, 2014.

4 C. H. Bennett and G. Brassard. Quantum cryptography: Public-key distribution and coin tossing. In *Proceedings of the IEEE International Conference on Computer, Systems and Signal Processing*, pages 175–179, 1984.

5 T. A. S. Davidson. *Formal Verification Techniques using Quantum Process Calculus*. PhD thesis, University of Warwick, 2011.

6 Y. Deng and Y. Feng. Open bisimulation for quantum processes. In *TCS'12: Proceedings of the 7th IFIP TC 1/WG 202 international conference on Theoretical Computer Science*. Springer-Verlag, September 2012. Full Version available at http://arxiv.org/abs/1201.0416.

7 Yuxin Deng, Rob van Glabbeek, Matthew Hennessy, and Carroll Morgan. Testing finitary probabilistic processes (extended abstract). In *CONCUR'09*, pages 274–288. Springer, 2009.

8 Laurent Doyen, Thomas A Henzinger, and Jean-Francois Raskin. Equivalence of labeled Markov chains. *International Journal of Foundations of Computer Science*, 19(03):549–563, 2008.

9 Christian Eisentraut, Jens Chr Godskesen, Holger Hermanns, Lei Song, and Lijun Zhang. Late Weak Bisimulation for Markov Automata. *http://arxiv.org/abs/1202.4116*, February 2012.

10 Y. Feng, R. Duan, Z. Ji, and M. Ying. Probabilistic bisimulations for quantum processes. *Information and Computation*, 205(11):1608–1639, November 2007.

11 Y. Feng, R. Duan, and M. Ying. Bisimulations for quantum processes. In Mooly Sagiv, editor, *POPL'11*, pages 523–534, 2011.

12 Yuan Feng, Yuxin Deng, and Mingsheng Ying. Symbolic bisimulation for quantum processes. *ACM Transactions on Computational Logic*, 15(2):14:1–14:32, May 2014.

13 S. J. Gay and R. Nagarajan. Communicating quantum processes. In J. Palsberg and M. Abadi, editors, *POPL'05*, pages 145–157, 2005.

14 Holger Hermanns, Jan Krcál, and Jan Kretínský. Probabilistic bisimulation: Naturally on distributions. In Paolo Baldan and Daniele Gorla, editors, *CONCUR'14*. Springer, 2014.

15 B. Jonsson, W. Yi, and K. G. Larsen. Probabilistic extensions of process algebras. In *Handbook of process algebra*, pages 685–710. North-Holland, Amsterdam, 2001.

16 P. Jorrand and M. Lalire. Toward a quantum process algebra. In P. Selinger, editor, *QPL'04*, page 111, 2004.

17 Takahiro Kubota, Yoshihiko Kakutani, Go Kato, Yasuhito Kawano, and Hideki Sakurada. Application of a process calculus to security proofs of quantum protocols. In *FCS'12*, pages 141–147, 2012.

18 M Kwiatkowska, G Norman, and D Parker. PRISM 2.0: a tool for probabilistic model checking. In *QEST'04*, pages 322–323, September 2004.

19 Marie Lalire. Relations among quantum processes: Bisimilarity and congruence. *Mathematical Structures in Computer Science*, 16(3):407–428, 2006.

20 Dominic Mayers. Unconditional security in quantum cryptography. *Journal of the ACM*, 48(3):351–406, 2001.

21 J Mitchell, A Ramanathan, A Scedrov, and V Teague. A Probabilistic Polynomial-time Calculus For Analysis of Cryptographic Protocols: (Preliminary Report). *Electronic Notes in Theoretical Computer Science*, 45:280–310, December 2000.

22 Rajagopal Nagarajan, Nikolaos Papanikolaou, Garry Bowen, and Simon Gay. An automated analysis of the security of quantum key distribution. In *SecCo'05*, 2005.

23 M. Nielsen and I. Chuang. *Quantum computation and quantum information*. Cambridge university press, 2000.

24 A. Di Pierro, C. Hankin, and H. Wiklicky. Measuring the confinement of probabilistic systems. *Theoretical Computer Science*, 340(1):3–56, 2005.

25 Ajith Ramanathan, John Mitchell, Andre Scedrov, and Vanessa Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. In *FOSSACS'04*, pages 468–483. Springer, Berlin, 2004.

26 Peter W Shor and John Preskill. Simple proof of security of the BB84 quantum key distribution protocol. *Physical Review Letters*, 85(2):441, 2000.

27 M. Ying, Y. Feng, R. Duan, and Z. Ji. An algebra of quantum processes. *ACM Transactions on Computational Logic*, 10(3):1–36, April 2009.

Unfolding-based Partial Order Reduction*

César Rodríguez[1], Marcelo Sousa[2], Subodh Sharma[3], and Daniel Kroening[4]

1 **Université Paris 13, Sorbonne Paris Cité, LIPN, CNRS, France**
2,4 **Department of Computer Science, University of Oxford, UK**
3 **Indian Institute of Technology Delhi, India**

Abstract

Partial order reduction (POR) and net unfoldings are two alternative methods to tackle state-space explosion caused by concurrency. In this paper, we propose the combination of both approaches in an effort to combine their strengths. We first define, for an abstract execution model, unfolding semantics parameterized over an arbitrary independence relation. Based on it, our main contribution is a novel stateless POR algorithm that explores at most one execution per Mazurkiewicz trace, and in general, can explore exponentially fewer, thus achieving a form of *super-optimality*. Furthermore, our unfolding-based POR copes with non-terminating executions and incorporates state caching. On benchmarks with busy-waits, among others, our experiments show a dramatic reduction in the number of executions when compared to a state-of-the-art DPOR.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Partial-order reduction, unfoldings, concurrency, model checking

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.456

1 Introduction

Efficient exploration of the state space of a concurrent system is a fundamental problem in automated verification. Concurrent actions often interleave in intractably many ways, quickly populating the state space with many equivalent but unequal states. Existing approaches to address this problem can essentially be classified as either partial-order reduction techniques (PORs) or unfolding methods.

Conceptually, POR methods [19, 7, 6, 8, 21, 20, 2, 1] exploit the fact that executing certain transitions can be postponed because their result is independent of the execution sequence taken in their stead. They execute a provably-sufficient subset of transitions enabled at every state, computed either statically [19, 7] or dynamically [6, 2]. The latter methods, referred as dynamic PORs (DPORs), are often *stateless* (i.e., they only store one execution in memory). By contrast, unfolding approaches [14, 5, 3, 10] model execution by partial orders, bound together by a conflict relation. They construct finite, complete prefixes by a saturation procedure, and cope with non-terminating executions using cutoff events [5, 3].

POR can employ highly sophisticated decision procedures to determine a sufficient subset of the transitions to fire, and in most cases [7, 6, 8, 21, 20, 2, 1] the *commutativity of transitions* is the enabling mechanism underlying the chosen method. Commutativity, or independence, is thus a mechanism and not necessarily an irreplaceable component of a

* This research was supported by ERC project 280053 (CPROVER).

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 456–469

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

POR [19, 9].[1] Conceptually, PORs that exploit commutativity establish an equivalence relation on the sequential executions of the system and explore at least one representative of each class, thus discarding equivalent executions. In this work we restrict our attention to exclusively PORs that exploit commutativity.

Despite impressive advances in the field, both unfoldings and PORs have shortcomings. We now give six of them. Current unfolding algorithms (1) need to solve an NP-complete problem when adding events to the unfolding [14], which seriously limits the performance of existing unfolders as the structure grows. They are also (2) inherently *stateful*, i.e., they cannot selectively discard visited events from memory, quickly running out of it. PORs, on the other hand, explore Mazurkiewicz traces [13], which (3) often outnumber the events in the corresponding unfolding by an exponential factor (e.g., Fig. 2 (d) gives an unfolding with $2n$ events and $\mathcal{O}(2^n)$ traces). Furthermore, DPORs often (4) explore the same states repeatedly [20], and combining them with stateful search, although achieved for non-optimal DPOR [20, 21], is difficult due to the dynamic nature of DPOR [21]. More on this in Example 1. The same holds when extending DPORs to (5) cope with non-terminating executions (note that a solution to (4) does not necessarily solve (5)). Lastly, (6) existing stateless PORs do not make full use of the available memory.

Either readily available solutions or promising directions to address these six problems can be found in, respectively, the opposite approach. PORs inexpensively add events to the current execution, contrary to unfoldings (1). They easily discard events from memory when backtracking, which addresses (2). On the other hand, while PORs explore Mazurkiewicz traces (*maximal configurations*), unfoldings explore events (*local configurations*), thus addressing (3). Explorations of repeated states and pruning of non-terminating executions is elegantly achieved in unfoldings by means of cutoff events. This solves (4) and (5).

Some of these solutions indeed seem, at present, incompatible with each other. We do not claim that the combination of POR and unfoldings immediately addresses the problems above. However, since both unfoldings and PORs share many fundamental similarities, tackling these problems in a unified framework is likely to shed light on them.

This paper lays out a DPOR algorithm on top of an unfolding structure. Our main result is a novel stateless, optimal DPOR that explores every Mazurkiewicz trace at most once, and often many fewer, owing to cutoff events. It also copes with non-terminating systems and exploits all available RAM with a *cache memory* of events, speeding up revisiting events. This provides a solution to (4), (5), (6), and a partial solution to (3). Our algorithm can alternatively be viewed as a stateless unfolding exploration, partially addressing (1) and (2).

Our result reveals DPORs as algorithms exploring an object that has richer structure than a plain directed graph. Specifically, unfoldings provide a solid notion of event *across multiple executions*, and a clear notion of conflict. Our algorithm indirectly maps important POR notions to concepts in unfolding theory.

▶ **Example 1.** We illustrate problems (3), (4), and (5), and explain how our DPOR deals with them. The following code is the skeleton of a producer-consumer program. Two concurrent producers write, resp., to `buf1` and `buf2`. The consumer accesses the buffers in sequence.

```
while (1):                    | while (1):
  lock(m1)                    |   lock(m2)
  if (buf1 < MAX): buf1++     |   if (buf2 < MAX): buf2++
  unlock(m1)                  |   unlock(m2)
```

[1] For instance, all PORs based on persistent sets [7] are based on commutativity.

```
while (1):
  lock(m1)
  if (buf1 > MIN): buf1--
  unlock(m1)
  // same for m2, buf2
```

Lock and unlock operations on both mutexes `m1` and `m2` create many Mazurkiewicz traces. However, most of them have isomorphic *suffixes*, e.g., producing two items in `buf1` and consuming one reaches the same state as only producing one. After the common state, both traces explore identical behaviours and only one needs to be explored. We use cutoff events, inherited from unfolding theory [5, 3], to dynamically stop the first trace and continue only with the second. This addresses (4) and (5), and partially deals with (3). Observe that cutoff events are a form of semantic pruning, in contrast to the syntactic pruning introduced by, e.g., bounding the depth of loops, a common technique for coping with non-terminating executions in DPOR. With cutoffs, the exploration can build unreachability *proofs*, while depth bounding renders DPOR incomplete, i.e., it limits DPOR to finding bugs.

Our first step is to formulate PORs and unfoldings in the same framework. PORs are often presented for abstract execution models, while unfoldings have mostly been considered for Petri nets, where the definition is entangled with the syntax of the net. We make a second contribution here. We define, for a general execution model, event structure semantics [16] parametric on a given independence relation.

Section 2 sets up basic notions and § 3 presents our parametric event-structure semantics. In § 4 we introduce our DPOR,§ 5 improves it with cutoff detection and discusses event caching. Experimental results are in § 6 and related work in § 7. We conclude in § 8. All lemmas cited along the paper and proofs of all stated results can be found in the extended version [17].

2 Execution Model and Partial Order Reductions

We set up notation and recall general ideas of POR. We consider an abstract model of (concurrent) computation. A *system* is a tuple $M := \langle \Sigma, T, \tilde{s} \rangle$ formed by a set Σ of *global states*, a set T of *transitions* and some *initial global state* $\tilde{s} \in \Sigma$. Each transition $t : \Sigma \to \Sigma$ in T is a *partial* function accounting for how the occurrence of t transforms the state of M.

A transition $t \in T$ is *enabled* at a state s if $t(s)$ is defined. Such t can *fire* at s, producing a new state $s' := t(s)$. We let $enabl(s)$ denote the set of transitions enabled at s. The *interleaving semantics* of M is the directed, edge-labelled graph $\mathcal{S}_M := \langle \Sigma, \to, \tilde{s} \rangle$ where Σ are the global states, $\tilde{s}$ is the initial state and $\to \subseteq \Sigma \times T \times \Sigma$ contains a triple $\langle s, t, s' \rangle$, denoted by $s \xrightarrow{t} s'$, iff t is enabled at s and $s' = t(s)$. Given two states $s, s' \in \Sigma$, and $\sigma := t_1.t_2 \dots t_n \in T^*$ (t_1 concatenated with t_2, ... until t_n), we denote by $s \xrightarrow{\sigma} s'$ the fact that there exist states $s_1, \dots, s_{n-1} \in \Sigma$ such that $s \xrightarrow{t_1} s_1$, ..., $s_{n-1} \xrightarrow{t_n} s'$.

A *run* (or *interleaving*, or *execution*) of M is any sequence $\sigma \in T^*$ such that $\tilde{s} \xrightarrow{\sigma} s$ for some $s \in \Sigma$. We denote by $state(\sigma)$ the state s that σ reaches, and by $runs(M)$ the set of runs of M, also referred to as the *interleaving space*. A state $s \in \Sigma$ is *reachable* if $s = state(\sigma)$ for some $\sigma \in runs(M)$; it is a *deadlock* if $enabl(s) = \varnothing$, and in that case σ is called *deadlocking*. We let $reach(M)$ denote the set of reachable states in M. For the rest of the paper, we fix a system $M := \langle \Sigma, T, \tilde{s} \rangle$ and assume that $reach(M)$ is finite.

The core idea behind POR[2] is that certain transitions can be seen as commutative

[2] To be completely correct we should say "*POR that exploits the independence of transitions*".

operators, i.e., changing their order of occurrence does not change the result. Given two transitions $t, t' \in T$ and one state $s \in \Sigma$, we say that t, t' *commute at* s iff

- if $t \in \mathit{enabl}(s)$ and $s \xrightarrow{t} s'$, then $t' \in \mathit{enabl}(s)$ iff $t' \in \mathit{enabl}(s')$; and
- if $t, t' \in \mathit{enabl}(s)$, then there is a state s' such that $s \xrightarrow{t.t'} s'$ and $s \xrightarrow{t'.t} s'$.

For instance, the lock operations on `m1` and `m2` (Example 1), commute on every state, as they update different variables. Commutativity of transitions at states identifies an equivalence relation on the set $\mathit{runs}(M)$. Two runs σ and σ' of the same length are *equivalent*, written $\sigma \equiv \sigma'$, if they are the same sequence modulo swapping commutative transitions. Thus equivalent runs reach the same state. POR methods explore a fragment of $\mathcal{S}_M$ that contains at least one run in the equivalence class of each run that reaches each deadlock state. This is achieved by means of a so-called *selective search* [7]. Since employing commutativity can be expensive, PORs often use *independence relations*, i.e., sound under-approximations of the commutativity relation. In this work, partially to simplify presentation, we use unconditional independence.

Formally, an *unconditional independence relation* on M is any symmetric and irreflexive relation $\diamond \subseteq T \times T$ such that if $t \diamond t'$, then t and t' commute at *every* state $s \in \mathit{reach}(M)$. If t, t' are not independent according to $\diamond$, then they are *dependent*, denoted by $t \circledast t'$.

Unconditional independence identifies an equivalence relation $\equiv_\diamond$ on the set $\mathit{runs}(M)$. Formally, $\equiv_\diamond$ is defined as the transitive closure of the relation $\equiv^1_\diamond$, which in turn is defined as $\sigma \equiv^1_\diamond \sigma'$ iff there is $\sigma_1, \sigma_2 \in T^*$ such that $\sigma = \sigma_1.t.t'.\sigma_2$, $\sigma' = \sigma_1.t'.t.\sigma_2$ and $t \diamond t'$. From the properties of $\diamond$, one can immediately see that $\equiv_\diamond$ refines $\equiv$, i.e., if $\sigma \equiv_\diamond \sigma'$, then $\sigma \equiv \sigma'$.

Given a run $\sigma \in \mathit{runs}(M)$, the equivalence class of $\equiv_\diamond$ to which σ belongs is called the *Mazurkiewicz trace* of σ [13], denoted by $\mathcal{T}_{\diamond,\sigma}$. Each trace $\mathcal{T}_{\diamond,\sigma}$ can equivalently be seen as a labelled partial order $\mathcal{D}_{\diamond,\sigma}$, traditionally called the *dependence graph* (see [13] for a formalization), satisfying that a run belongs to the trace iff it is a linearization of $\mathcal{D}_{\diamond,\sigma}$.

Sleep sets [7] are another method for state-space reduction. Unlike selective exploration, they prune successors by looking at the past of the exploration, not the future.

3 Parametric Partial Order Semantics

An unfolding is, conceptually, a tree-like structure of partial orders. In this section, given an independence relation $\diamond$ (our parameter) and a system M, we define an unfolding semantics $\mathcal{U}_{M,\diamond}$ with the following property: each constituent partial order of $\mathcal{U}_{M,\diamond}$ will correspond to one dependence graph $\mathcal{D}_{\diamond,\sigma}$, for some $\sigma \in \mathit{runs}(M)$. For the rest of this paper, let $\diamond$ be an arbitrary unconditional independence relation on M. We use prime event structures [16], a non-sequential, event-based model of concurrency, to define the unfolding $\mathcal{U}_{M,\diamond}$ of M.

▶ **Definition 2** (LES). Given a set A, an *A-labelled event structure* (A-LES, or LES in short) is a tuple $\mathcal{E} := \langle E, <, \#, h\rangle$ where E is a set of *events*, $< \subseteq E \times E$ is a strict partial order on E, called *causality relation*, $h\colon E \to A$ labels every event with an element of A, and $\# \subseteq E \times E$ is the symmetric, irreflexive *conflict relation*, satisfying

- for all $e \in E$, $\{e' \in E : e' < e\}$ is finite, and (1)
- for all $e, e', e'' \in E$, if $e \# e'$ and $e' < e''$, then $e \# e''$. (2)

The *causes* of an event $e \in E$ are the set $\lceil e \rceil := \{e' \in E : e' < e\}$ of events that need to happen before e for e to happen. A *configuration* of $\mathcal{E}$ is any finite set $C \subseteq E$ satisfying:

- (causally closed) for all $e \in C$ we have $\lceil e \rceil \subseteq C$; (3)
- (conflict free) for all $e, e' \in C$, it holds that $\neg e \# e'$. (4)

Intuitively, configurations represent partially-ordered executions. In particular, the *local configuration* of e is the $\subseteq$-minimal configuration that contains e, i.e. $\lceil e \rceil := \lceil e \rceil \cup \{e\}$.

We denote by $conf(\mathcal{E})$ the set of configurations of $\mathcal{E}$. Two events e, e' are in *immediate conflict*, $e \#^i e'$, iff $e \# e'$ and both $\lceil e \rceil \cup \lceil e' \rceil$ and $\lceil e \rceil \cup \lceil e' \rceil$ are configurations. Lastly, given two LESs $\mathcal{E} := \langle E, <, \#, h \rangle$ and $\mathcal{E}' := \langle E', <', \#', h' \rangle$, we say that $\mathcal{E}$ is a *prefix* of $\mathcal{E}'$, written $\mathcal{E} \trianglelefteq \mathcal{E}'$, when $E \subseteq E'$, $<$ and $\#$ are the projections of $<'$ and $\#'$ to E, and $E \supseteq \{e' \in E' : e' < e \wedge e \in E\}$.

Our semantics will unroll the system M into a LES $\mathcal{U}_{M,\diamond}$ whose events are labelled by transitions of M. Each configuration of $\mathcal{U}_{M,\diamond}$ will correspond to the dependence graph $\mathcal{D}_{\diamond,\sigma}$ of some $\sigma \in runs(M)$. For a LES $\langle E, <, \#, h \rangle$, we define the *interleavings* of C as $inter(C) := \{h(e_1), \ldots, h(e_n) : e_i, e_j \in C \wedge e_i < e_j \implies i < j\}$. Although for arbitrary LES $inter(C)$ may contain sequences not in $runs(M)$, the definition of $\mathcal{U}_{M,\diamond}$ will ensure that $inter(C) \subseteq runs(M)$. Additionally, since all sequences in $inter(C)$ belong to the same trace, all of them reach the same state. Abusing the notation, we define $state(C) := state(\sigma)$ if $\sigma \in inter(C)$. The definition is neither well-given nor unique for arbitrary LES, but will be so for the unfolding.

We now define $\mathcal{U}_{M,\diamond}$. Each event will be inductively identified by a canonical name of the form $e := \langle t, H \rangle$, where $t \in T$ is a transition of M and H a configuration of $\mathcal{U}_{M,\diamond}$. Intuitively, e represents the occurrence of t after the *history* (or the causes) $H := \lceil e \rceil$. The definition will be inductive. The base case inserts into the unfolding a special *bottom event* $\perp$ on which every event causally depends. The inductive case iteratively extends the unfolding with one event. We define the set $\mathcal{H}_{\mathcal{E},\diamond,t}$ of candidate *histories* for a transition t in an LES $\mathcal{E}$ as the set which contains exactly all configurations H of $\mathcal{E}$ such that

- transition t is enabled at $state(H)$, and
- either $H = \{\perp\}$ or all $<$-maximal events e in H satisfy that $h(e) \otimes t$,

where h is the labelling function in $\mathcal{E}$. Once an event e has been inserted into the unfolding, its associated transition $h(e)$ may be dependent with $h(e')$ for some e' already present and outside the history of e. Since the order of occurrence of e and e' matters, we need to prevent their occurrence within the same configuration, as configurations represent equivalent executions. We therefore introduce a conflict between e and e'. The set $\mathcal{K}_{\mathcal{E},\diamond,e}$ of *events conflicting* with $e := \langle t, H \rangle$ thus contains any event e' in $\mathcal{E}$ with $e' \notin [e]$ and $e \notin [e']$ and $t \otimes h(e')$.

Following common practice [4], the definition of $\mathcal{U}_{M,\diamond}$ proceeds in two steps. We first define (Def. 3) the collection of all prefixes of the unfolding. Then we show that there exists only one $\trianglelefteq$-maximal element in the collection, and define it to be *the* unfolding (Def. 4).

▶ **Definition 3** (Finite unfolding prefixes)**.** The set of *finite unfolding prefixes* of M under the independence relation $\diamond$ is the smallest set of LESs that satisfies the following conditions:

1. The LES having exactly one event $\perp$, empty causality and conflict relations, and $h(\perp) := \varepsilon$ is an unfolding prefix.
2. Let $\mathcal{E}$ be an unfolding prefix containing a history $H \in \mathcal{H}_{\mathcal{E},\diamond,t}$ for some transition $t \in T$. Then, the LES $\langle E, <, \#, h \rangle$ resulting from extending $\mathcal{E}$ with a new event $e := \langle t, H \rangle$ and satisfying the following constraints is also an unfolding prefix of M:
   - for all $e' \in H$, we have $e' < e$;
   - for all $e' \in \mathcal{K}_{\mathcal{E},\diamond,e}$, we have $e \# e'$; and $h(e) := t$.

Intuitively, each unfolding prefix contains the dependence graph (configuration) of one or more executions of M (of finite length). The unfolding starts from $\perp$, the "root" of the tree, and then iteratively adds events enabled by some configuration until saturation, i.e., when no more events can be added. Observe that the number of unfolding prefixes as per

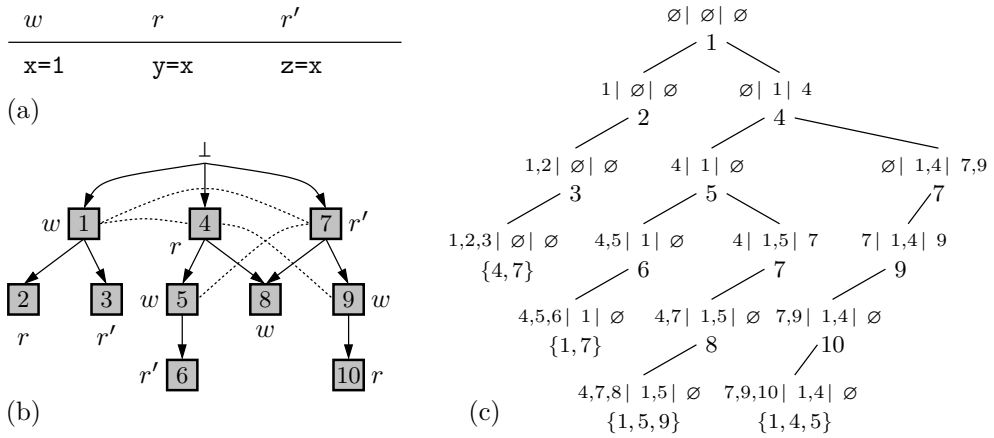


Figure 1 Running example. (a) A concurrent program; (b) its unfolding semantics. (c) The exploration performed by Alg. 1, where each node C| D| A represents one call to the function `Explore`(C, D, A). The set X underneath each leaf node is such that the value of variable U in Alg. 1 at the leaf is $U = C \cup D \cup X$. At ∅| ∅| ∅, the alternative taken is $\{4\}$, and at 4| 1| ∅ it is $\{7\}$.

Def. 3 will be finite iff all runs of M terminate. Due to lack of space, we give the definition of *infinite* unfolding prefixes in the extended version [17], as the main ideas of this section are well conveyed using only finite prefixes. In the sequel, by *unfolding prefix* we mean a finite or infinite one.

Our first task is checking that each unfolding prefix is indeed a LES [17, Lemma 14]. Next one shows that the configurations of every unfolding prefix correspond the Mazurkiewicz traces of the system, i.e., for any configuration C, $\mathit{inter}(C) = \mathcal{T}_{\diamond,\sigma}$ for some $\sigma \in \mathit{runs}(M)$ [17, Lemma 16]. This implies that the definition of $\mathit{inter}(C)$ and $\mathit{state}(C)$ is well-given when C belongs to an unfolding prefix. The second task is defining *the* unfolding $\mathcal{U}_{M,\diamond}$ of M. Here, we prove that the set of unfolding prefixes equipped with relation $\trianglelefteq$ forms a complete join-semilattice [17, Lemma 17]. This implies the existence of a unique $\trianglelefteq$-maximal element:

▶ **Definition 4** (Unfolding)**.** The *unfolding* $\mathcal{U}_{M,\diamond}$ of M under the independence relation $\diamond$ is the *unique* $\trianglelefteq$-maximal element in the set of unfolding prefixes of M under $\diamond$.

Finally we verify that the definition is well given and that the unfolding is *complete*, i.e., every run of the system is represented by a unique configuration of the unfolding.

▶ **Theorem 5.** *The unfolding $\mathcal{U}_{M,\diamond}$ exists and is unique. Furthermore, for any non-empty run σ of M, there exists a unique configuration C of $\mathcal{U}_{M,\diamond}$ such that $\sigma \in \mathit{inter}(C)$.*

▶ **Example 6** (Programs)**.** Figure 1 (a) gives a concurrent program, where process w writes a global variable and processes r and r' read it. We can associate various semantics to it. Under an empty independence relation, the unfolding would be the computation tree, where executions would be totally ordered. Considering (the unique transition of) r and r' independent, and w dependent on them, we get the unfolding given in Fig. 1 (b).

Events are numbered from 1 to 10, and labelled with a transition. Arrows represent causality between events and dotted lines immediate conflict. The Mazurkiewicz trace of each deadlocking execution is represented by a unique $\subseteq$-maximal configuration, e.g., the run $w.r.r'$ yields configuration $\{1, 2, 3\}$, where the two possible interleavings reach the same

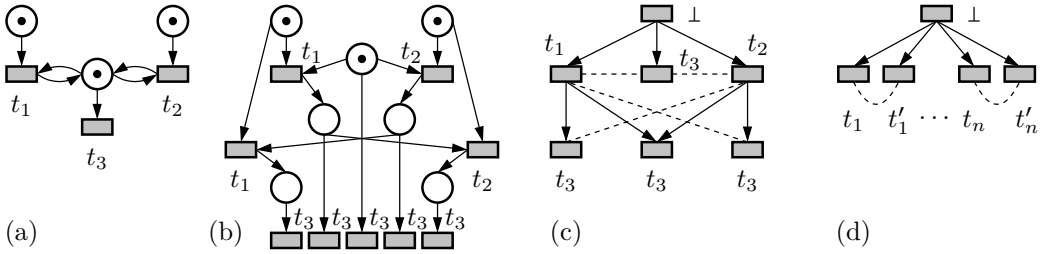


Figure 2 (a) A Petri net; (b) its classic unfolding; (c) our parametric semantics.

state. For instance, the canonic name of event 1 is $\langle w, \{\bot\}\rangle$ and of event 2 it is $\langle r, \{\bot, 1\}\rangle$. Let $\mathcal{P}$ be the unfolding prefix that contains events $\{\bot, 1, 2\}$. Definition 3 can extend it with three possible events: 3, 4, and 7. Consider transition r'. Three configurations of $\mathcal{P}$ enable r': $\{\bot\}, \{\bot, 1\}$ and $\{\bot, 1, 2\}$. But since $\neg(h(2) \otimes r')$, only the first two will be in $\mathcal{H}_{\mathcal{P},\diamond,r'}$, resulting in events $3 := \langle r', \{\bot, 1\}\rangle$ and $7 := \langle r', \{\bot\}\rangle$. Also, $\mathcal{K}_{\mathcal{P},\diamond,7}$ is $\{1\}$, as $w \otimes r'$. The four maximal configurations are $\{1, 2, 3\}$, $\{4, 5, 6\}$, $\{4, 7, 8\}$ and $\{7, 9, 10\}$, resp. reaching the states $\langle x, y, z\rangle = \langle 1, 1, 1\rangle$, $\langle 1, 0, 1\rangle$, $\langle 1, 0, 0\rangle$ and $\langle 1, 1, 0\rangle$, assuming that variables start at 0.

▶ **Example 7** (Comparison to Petri Net Unfoldings)**.** In contrast to our parametric semantics, classical unfoldings of Petri nets [5] use a fixed independence relation, specifically the complement of the following one (valid only for safe nets): given two transitions t and t',

$$t \otimes_n t' \text{ iff } (t^\bullet \cap {}^\bullet t' \neq \varnothing) \text{ or } (t'^\bullet \cap {}^\bullet t \neq \varnothing) \text{ or } ({}^\bullet t' \cap {}^\bullet t \neq \varnothing),$$

where ${}^\bullet t$ and $t^\bullet$ are respectively the *preset* and *postset* of t. Classic Petri net unfoldings (of safe nets) are therefore a specific instantiation of our semantics. A well known challenge for classic unfoldings are transitions that "*read*" places, e.g., t_1 and t_2 in Fig. 2 (a). Since $t_1 \otimes_n t_2$, the classic unfolding, Fig. 2 (b), sequentializes all their occurrences. A solution for this issue is the so-called *place replication (PR) unfolding* [15], or alternatively *contextual unfoldings* (which anyway internally are asymptotically the same size as the PR-unfolding).

This problem vanishes with our parametric unfolding. It suffices to use a dependency relation $\otimes_n' \subset \otimes_n$ that makes transitions that "*read*" common places independent. The result is that our unfolding, Fig. 2 (c), can be of the same size as the PR-unfolding, i.e., exponentially more compact than the classic unfolding. For instance, when Fig. 2 (a) is generalized to n *reading* transitions, the classic unfolding would have $\mathcal{O}(n!)$ copies of t_3, while ours would have $\mathcal{O}(2^n)$. The point here is that our semantics naturally accommodates a more suitable notion of independence without resorting to specific ad-hoc tricks.

Furthermore, although this work is restricted to *unconditional* independence, we conjecture that an adequately restricted *conditional* dependence would suffice, e.g., the one of [12]. Gains achieved in such setting would be difficult with classic unfoldings.

4 Stateless Unfolding Exploration Algorithm

We present a DPOR algorithm to explore an arbitrary event structure (e.g., the one of § 3) instead of sequential executions. Our algorithm explores one configuration at a time and organizes the exploration into a binary tree. Figure 1 (c) gives an example. The algorithm is optimal [2], in the sense that no configuration is ever visited twice in the tree.

Algorithm 1: An unfolding-based POR exploration algorithm.

```
1  Initially, set U := {⊥}, set G := ∅, and call Explore({⊥}, ∅, ∅).
2  Procedure Explore(C, D, A)
3      Extend(C)
4      if en(C) = ∅ return
5      if A = ∅
6          Choose e from en(C)
7      else
8          Choose e from A ∩ en(C)
9      Explore(C ∪ {e}, D, A ∖ {e})
10     if ∃J ∈ Alt(C, D ∪ {e})
11         Explore(C, D ∪ {e}, J ∖ C)
12     Remove(e, C, D)
13 Procedure Extend(C)
14     Add ex(C) to U
15 Procedure Remove(e, C, D)
16     Move {e} ∖ Q_{C,D,U} from U to G
17     foreach ê ∈ #^i_U(e)
18         Move [ê] ∖ Q_{C,D,U} from U to G
```

For the rest of the paper, let $\mathcal{U}_{\diamond,M} := \langle E, <, \#, h\rangle$ be the unfolding of M under $\diamond$, which we abbreviate as $\mathcal{U}$. For this section we assume that $\mathcal{U}$ is finite, i.e., that all computations of M terminate. This is only to ease presentation, and we relax this assumption in § 5.2.

We give some new definitions. Let C be a configuration of $\mathcal{U}$. The *extensions* of C, written $ex(C)$, are all those events outside C whose causes are included in C. Formally, $ex(C) := \{e \in E : e \notin C \wedge \lceil e \rceil \subseteq C\}$. We let $en(C)$ denote the set of events *enabled* by C, i.e., those corresponding to the transitions enabled at $state(C)$, formally defined as $en(C) := \{e \in ex(C) : C \cup \{e\} \in conf(\mathcal{U})\}$. All those events in $ex(C)$ that are not in $en(C)$ are the *conflicting extensions*, $cex(C) := \{e \in ex(C) : \exists e' \in C,\ e \#^i e'\}$. Clearly, sets $en(C)$ and $cex(C)$ partition the set $ex(C)$. Lastly, we define $\#^i(e) := \{e' \in E : e \#^i e'\}$, and $\#^i_U(e) := \#^i(e) \cap U$. The difference between both is that $\#^i(e)$ contains events from *anywhere* in the unfolding structure, while $\#^i_U(e)$ can only *see* events in U.

The algorithm is given as Alg. 1. The main procedure `Explore(`C, D, A`)` is given the configuration that is to be explored as parameter C. The parameter D (for *disabled*) is the set of set of events that have already been explored and prevents that `Explore()` repeats work. It can be seen as a *sleep set* [7]. The set A (for *add*) is occasionally used to guide the direction of the exploration.

Additionally, a global set U stores all events presently known to the algorithm. Whenever some event can safely be discarded from memory, `Remove` will move it from U to G (for *garbage*). Once in G, it can be discarded at any time, or be preserved in G in order to save work when it is re-inserted in U. Set G is thus our *cache memory* of events.

The key intuition for Alg. 1 is as follows. A call to `Explore(`C, D, A`)` visits all maximal configurations of $\mathcal{U}$ that contain C and do not contain D; and the first one explored will contain $C \cup A$. Figure 1 (c) gives one execution; tree nodes are of the form C| D| A .

The algorithm first updates U with all extensions of C (procedure `Extend`). If C is a maximal configuration, then there is nothing to do, and it backtracks. If not, it chooses an event in U enabled at C, using the function `en(`C`)` $:= en(C) \cap U$. If A is empty, any enabled event can be taken. If not, A needs to be explored and e must come from the intersection. Next it makes a recursive call (left subtree), where it explores *all* configurations containing all events in $C \cup \{e\}$ and no event from D. Since `Explore(`C, D, A`)` had to visit all maximal configurations containing C, it remains to visit those containing C but not e, but only if

there exists at least one! Thus, we determine whether $\mathcal{U}$ has a maximal configuration that contains C, does not contain D and does not contain e. Function `Alt` will return a set of events that witness the existence of such configuration (iff one exists). If one exists, we make a second recursive call (right subtree). Formally, we call such witness an *alternative*:

▶ **Definition 8** (Alternatives). Given a set of events $U \subseteq E$, a configuration $C \subseteq U$, and a set of events $D \subseteq U$, an *alternative* to D after C is any configuration $J \subseteq U$ satisfying that

- $C \cup J$ is a configuration (5)
- for all events $e \in D$, there is some $e' \in C \cup J$ such that $e' \in \#^i_U(e)$. (6)

Function `Alt`(X, Y) returns all alternatives (in U) to Y after X. Notice that it is called as `Alt`$(C, D \cup \{e\})$ from Alg. 1. Any returned alternative J witnesses the existence of a maximal configuration C' (constructed by arbitrarily extending $C \cup J$) where $C' \cap (D \cup \{e\}) = \varnothing$.

Although `Alt` reasons about maximal configurations of $\mathcal{U}$, thus potentially about events that have not yet been seen, it can only look at events in U. Thus, the set U needs to be large enough to contain enough *conflicting events* to satisfy (6). Perhaps surprisingly, it suffices to store only events seen (during the past exploration) in immediate conflict with C and D. Consequently, when the algorithm calls `Remove`, to clean from U events that are no longer necessary (i.e., necessary to find alternatives in the future), it needs to preserve at least those conflicting events. Specifically, `Remove` will preserve in U the following events:

$$Q_{C,D,U} := C \cup D \cup \bigcup_{e \in C \cup D, e' \in \#^i_U(e)} [e'].$$

That is, events in C, in D and events in conflict with those. An alternative definition that makes $Q_{C,D,U}$ smaller would mean that `Remove` discards more events, which could prevent a future call to `Alt` from discovering a maximal configuration that needs to be explored.

We focus now on the correctness of Alg. 1. Every call to `Explore`(C, D, A) explores a tree, where the recursive calls at lines 9 and 11 respectively explore the left and right subtrees (proof in [17, Corollary 25]). Tree nodes are tuples $\langle C, D, A\rangle$ corresponding to the arguments of calls to `Explore`, cf. Fig. 1. We refer to this object as the *call tree*. For every node, both C and $C \cup A$ are configurations, and $D \subseteq ex(C)$, cf. [17, Lemma 18]. As the algorithm goes down in the tree it monotonically increases the size of either C or D. Since $\mathcal{U}$ is finite, this implies that the algorithm terminates:

▶ **Theorem 9** (Termination). *Regardless of its input, Alg. 1 always stops.*

Next we assert that Alg. 1 never visits twice the same configuration, which is why it is called an *optimal* POR [2]. We show that for every node in the call tree, the set of configurations in the left and right subtrees are disjoint [17, Lemma 24]. This implies:

▶ **Theorem 10** (Optimality). *Let $\tilde{C}$ be a maximal configuration of $\mathcal{U}$. Then* `Explore`$(\cdot,\cdot,\cdot)$ *is called* at most once *with its first parameter being equal to $\tilde{C}$.*

Parameter A of `Explore` plays a central role in making Alg. 1 optimal. It is necessary to ensure that, once the algorithm decides to explore some alternative J, such an alternative is visited first. Not doing so makes it possible to extend C in such a way that no maximal configuration can ever avoid including events in D. Such a configuration, referred as a *sleep-set blocked* execution in [2], has already been explored before.

Finally, we ensure that Alg. 1 visits every maximal configuration of $\mathcal{U}$. This essentially reduces to showing that it makes the second recursive call, line 11, whenever there exists some unexplored maximal configuration not containing $D \cup \{e\}$. The difficulty of proving

this [17, Lemma 27] aises from the fact that Alg. 1 *only* sees events in U. Owing to space constraints, we omit an additional result on the memory consumption, see [17, Appendix B.5.].

▶ **Theorem 11** (Completeness). *Let $\tilde{C}$ be a maximal configuration of $\mathcal{U}$. Then* `Explore`$(\cdot,\cdot,\cdot)$ *is called* at least once *with its first parameter being equal to $\tilde{C}$.*

5 Improvements

5.1 State Caching

Stateless model checking algorithms explore only one configuration of $\mathcal{U}$ at a time, thus potentially under-using remaining available memory. A desirable property for an algorithm is the capacity to exploit all available memory without imposing the liability of actually requiring it. The algorithm in § 4 satisfies this property. The set G, storing events discarded from U, can be cleaned at discretion, e.g., when the memory is approaching full utilisation. Events cached in G are exploited in two different ways.

First, whenever an event in G shall be included again in U, we do not need to reconstruct it in memory (causality, conflicts, etc.). This might happen frequently. Second, using the result of the next section, cached events help prune the number of maximal configurations to visit. This means that our POR potentially visits *fewer* final states than the number of configurations of $\mathcal{U}$, thus conforming to the requirements of a *super-optimal DPOR*. The larger G is, the fewer configurations will be explored.

5.2 Non-Acyclic State Spaces

In this section we remove the assumption that $\mathcal{U}_{M,\diamond}$ is finite. We employ the notion of cutoff events [14]. While cutoffs are a standard tool for unfolding pruning, their application to our framework brings unexpected problems.

The core question here is preventing Alg. 1 from getting stuck in the exploration of an infinite configuration. We need to create the illusion that maximal configurations are finite. We achieve this by substituting procedure `Extend` in Alg. 1 with another procedure `Extend'` that operates as `Extend` except that it only adds to U an event from $e \in ex(C)$ if the predicate $\mathsf{cutoff}(e,U,G)$ evaluates to false. We define $\mathsf{cutoff}(e,U,G)$ to hold iff there exists some event $e' \in U \cup G$ such that

$$state([e]) = state([e']) \quad \text{and} \quad |[e']| < |[e]|. \tag{7}$$

We refer to e' as the *corresponding* event of e, when it exists. This definition declares e cutoff as function of U and G. This has important consequences. An event e could be declared cutoff while exploring one maximal configuration and non-cutoff while exploring the next, as the corresponding event might have disappeared from $U \cup G$. This is in stark contrast to the classic unfolding construction, where events are declared cutoffs *once and for all*. The main implication is that the standard argument [14, 5, 3] invented by McMillan for proving completeness fails. We resort to a completely different argument for proving completeness of our algorithm (see [17, Appendix C.1.]), which we are forced to skip due to lack of space.

We focus now on the correction of Alg. 1 using `Extend'` instead of `Extend`. A *causal cutoff* is any event e for which there is some $e' \in [e]$ satisfying (7). It is well known that causal cutoffs define a finite prefix of $\mathcal{U}$ as per the classic saturation definition [3]. Also, $\mathsf{cutoff}(e,U,G)$ always holds for causal cutoffs, regardless of the contents of U and G. This

means that the modified algorithm can only explore configurations from a finite prefix. It thus necessarily terminates. As for optimality, it is unaffected by the use of cutoffs, existing proofs for Alg. 1 still work. Finally, for completeness we prove the following result, stating that local reachability (e.g., fireability of transitions of M) is preserved:

▶ **Theorem 12** (Completeness). *For any reachable state $s \in reach(M)$, Alg. 1 updated with the cutoff mechanism described above explores one configuration C such that for some $C' \subseteq C$ it holds that $state(C') = s$.*

Lastly, we note that this cutoff approach imposes no liability on what events shall be kept in the prefix, set G can be cleaned at discretion. Also, redefining (7) to use adequate orders [5] is straightforward (see [17], where our proofs actually assume adequate orders).

6 Experiments

As a proof of concept, we implemented our algorithm in a new explicit-state model checker baptized Poet (Partial Order Exploration Tool).[3] Written in Haskell, a lazy functional language, it analyzes programs from a restricted fragment of the C language and supports POSIX threads. The analyzer accepts deterministic programs, implements a variant of Alg. 1 where the computation of the alternatives is memoized, and supports cutoffs events with the criteria defined in § 5.

We ran Poet on a number of multi-threaded C programs. Most of them are adapted from benchmarks of the Software Verification Competition [18]; others are used in related works [8, 20, 2]. We investigate the characteristics of average program unfoldings (depth, width, etc.) as well as the frequency and impact of cutoffs on the exploration. We also compare Poet with Nidhugg [1], a state-of-the-art stateless model checking for multi-threaded C programs that implements Source-DPOR [2], an efficient but non-optimal DPOR. All experiments were run on an Intel Xeon CPU with 2.4 GHz and 4 GB memory. Tables 1 and 2 give our experimental data for programs with acyclic and non-acyclic state spaces, respectively.

For programs with acyclic state spaces (Table 1), Poet with and without cutoffs seems to perform the same exploration when the unfolding has no cutoffs, as expected. Furthermore, the number of explored executions also coincides with Nidhugg when the latter reports 0 sleep-set blocked executions (cf., § 4), providing experimental evidence of Poet's optimality.

The unfoldings of most programs in Table 1 do not contain cutoffs. All these programs are deterministic, and many of them highly sequential (Stf, Spin08, Fib), features known to make cutoffs unlikely. Ccnf(n) are concurrent programs composed of $n-1$ threads where thread i and $i+1$ race on writing one variable, and are independent of all remaining threads. Their unfoldings resemble Fig. 2 (d), with $2^{(n-1)/2}$ traces but only $\mathcal{O}(n)$ events. Saturation-based unfolding methods would win here over both Nidhugg and Poet.

In the ssb benchmarks, Nidhugg encounters sleep-set blocked executions, thus performing sub-optimal exploration. By contrast, Poet finds many cutoff events and achieves a *super-optimal* exploration, exploring fewer traces than both Poet without cutoffs and Nidhugg. The data shows that this *super-optimality* results in substantial savings in runtime.

For non-acyclic state spaces (Table 2), unfoldings are infinite. We thus compare Poet with cutoffs and Nidhugg with a loop bound. Hence, while Nidhugg performs bounded model checking, Poet does complete verification. The benchmarks include classical mutual

[3] Source code and benchmarks available from: `http://www.cs.ox.ac.uk/people/marcelo.sousa/poet/`.

Table 1 Programs with acyclic state space. Columns are: $|P|$: nr. of threads; $|I|$: nr. of explored traces; $|B|$: nr. of sleep-set blocked executions; $t(s)$: running time; $|E|$: nr. of events in $\mathcal{U}$; $|E_{\text{cut}}|$: nr. of cutoff events; $|\Omega|$: nr. of maximal configurations; $\langle|U_\Omega|\rangle$: avg. nr. of events in U when exploring a maximal configuration. A $*$ marks programs containing bugs. `<7K` reads as "*fewer than 7000*".

Benchmark		Nidhugg			Poet (without cutoffs)				Poet (with cutoffs)				
Name	$\|P\|$	$\|I\|$	$\|B\|$	$t(s)$	$\|E\|$	$\|\Omega\|$	$\langle\|U_\Omega\|\rangle$	$t(s)$	$\|E\|$	$\|E_{\text{cut}}\|$	$\|\Omega\|$	$\langle\|U_\Omega\|\rangle$	$t(s)$
Stf	3	6	0	0.06	121	6	79	0.04	121	0	6	79	0.06
Stf$*$	3	–	–	0.05	–	–	–	0.02	–	–	–	–	0.03
Spin08	3	84	0	0.08	2974	84	1506	2.04	2974	0	84	1506	2.93
Fib	3	8953	0	3.36	<185K	8953	92878	305	<185K	0	8953	92878	704
Fib$*$	3	–	–	0.74	–	–	–	81.0	–	–	–	–	133
Ccnf(9)	9	16	0	0.05	49	16	46	0.07	49	0	16	46	0.06
Ccnf(17)	17	256	0	0.15	97	256	94	5.76	97	0	256	94	6.09
Ccnf(19)	19	512	0	0.28	109	512	106	22.5	109	0	512	106	22.0
Ssb	5	4	2	0.05	48	4	38	0.03	46	1	4	37	0.03
Ssb(1)	5	22	14	0.06	245	23	143	0.11	237	4	23	140	0.11
Ssb(3)	5	169	67	0.12	2798	172	1410	3.51	1179	48	90	618	0.90
Ssb(4)	5	336	103	0.15	<7K	340	3333	20.3	2179	74	142	1139	2.07
Ssb(8)	5	2014	327	0.85	<67K	2022	32782	4118	<12K	240	470	6267	32.1

Table 2 Programs with non-terminating executions. Column b is the loop bound. The value is chosen based on experiments described in [1].

Benchmark			Nidhugg			Poet (with cutoffs)				
Name	$\|P\|$	b	$\|I\|$	$\|B\|$	$t(s)$	$\|E\|$	$\|E_{\text{cut}}\|$	$\|\Omega\|$	$\langle\|U_\Omega\|\rangle$	$t(s)$
Szymanski	3	–	103	0	0.07	1121	313	159	591	0.36
Dekker	3	10	199	0	0.11	217	14	21	116	0.07
Lamport	3	10	32	0	0.06	375	28	30	208	0.12
Peterson	3	10	266	0	0.11	175	15	20	100	0.05
Pgsql	3	10	20	0	0.06	51	8	4	40	0.03
Rwlock	5	10	2174	14	0.83	<7317	531	770	3727	12.29
Rwlock(2)$*$	5	2	–	–	7.88	–	–	–	–	0.40
Prodcons	4	5	756756	0	332.62	3111	568	386	1622	5.00
Prodcons(2)	4	5	63504	0	38.49	640	25	15	374	1.61

exclusion protocols (Szymanski, Sekker, Lamport and Peterson), where Nidhugg is able to leverage an important static optimization that replaces each spin loop by a load and assume statement [1]. Hence, the number of traces and maximal configurations is not comparable. Yet Poet, which could also profit from this static optimization, achieves a significantly better reduction thanks to cutoffs alone. Cutoffs dynamically prune redundant unfolding branches and arguably constitute a more robust approach than the load and assume syntactic substitution. The substantial reduction in number of explored traces, several orders of magnitude in some cases, translates in clear runtime improvements. Finally, in our experiments, both tools were able to successfully discover assertion violations in stf$*$, fib$*$ and rwlock(2)$*$.

In our experiments, Poet's average maximal memory consumption (measured in events) is roughly half of the size of the unfolding. We also notice that most of these unfoldings are quite narrow and deep ($|E_{\text{cut}}| \div |E|$ is low) when compared with standard benchmarks for Petri nets. This suggests that they could be amenable for saturation-based unfolding verification, possibly pointing the opportunity of applying these methods in software verification.

7 Related Work

This work focuses on explicit-state POR, as opposed to symbolic POR techniques exploited inside SAT solvers, e.g., [11, 8]. Early POR statically computed the necessary transitions to fire at every state [19, 7]. Flanagan and Godefroid [6] first proposed to compute persistent sets dynamically (DPOR). However, even when combined with sleep sets [7], DPOR was still unable to explore exactly one interleaving per Mazurkiewicz trace. Abdulla et al. [2, 1] recently proposed the first solution to this, using a data structure called wakeup trees. Their DPOR is thus optimal (ODPOR) in this sense.

Unlike us, ODPOR operates on an interleaved execution model. Wakeup trees store chains of dependencies that assist the algorithm in reversing races throughly. Technically, each branch roughly correspond to one of our alternatives. According to [2], constructing and managing wakeup trees is expensive. This seems to be related with the fact that wakeup trees store canonical linearizations of configurations, and need to canonize executions before inserting them into the tree to avoid duplicates. Such checks become simple linear-time verifications when seen as partial-orders. Our alternatives are computed dynamically and exploit these partial orders, although we do not have enough experimental data to compare with wakeup trees. Finally, our algorithm is able to visit up to exponentially fewer Mazurkiewicz traces (due to cutoff events), copes with non-terminating executions, and profits from state caching. The work in [2] has none of these features.

Combining DPOR with stateful search is challenging [21]. Given a state s, DPOR relies on a complete exploration from s to determine the necessary transitions to fire from s, but such exploration could be pruned if a state is revisited, leading to unsoundness. Combining both methods requires addressing this difficulty, and two works did it [21, 20], but for non-optimal DPOR. By contrast, incorporating cutoff events into Alg. 1 was straightforward.

Classic, saturation-based unfolding algorithms are also related [14, 5, 3, 10]. They are inherently stateful, cannot discard events from memory, but explore events instead of configurations, thus may do exponentially less work. They can furthermore guarantee that the number of explored events will be at most the number of reachable states, which at present seems a difficult goal for PORs. On the other hand, finding the events to extend the unfolding is computationally harder. In [10], Kähkönen and Heljanko use unfoldings for concolic testing of concurrent programs. Unlike ours, their unfolding is not a semantics of the program, but rather a means for discovering all concurrent program paths.

While one goal of this paper is establishing an (optimal) POR exploiting the same commutativity as some non-sequential semantics, a longer-term goal is building formal connections between the latter and PORs. Hansen and Wang [9] presented a characterization of (a class of) stubborn sets [19] in terms of configuration structures, another non-sequential semantics more general than event structures. We shall clarify that while we restrict ourselves to commutativity-based PORs, they attempt a characterization of stubborn sets, which do not necessarily rely on commutativity.

8 Conclusions

In the context of commutativity-exploiting POR, we introduced an optimal DPOR that leverages on cutoff events to prune the number of explored Mazurkiewicz traces, copes with non-terminating executions, and uses state caching to speed up revisiting events. The algorithm provides a new view to DPORs as algorithms exploring an object with richer structure. In future work, we plan exploit this richer structure to further reduce the number of explored traces for both PORs and saturation-based unfoldings.

References

1 Parosh Abdulla, Stavros Aronis, Mohamed Faouzi Atig, Bengt Jonsson, Carl Leonardsson, and Konstantinos Sagonas. Stateless Model Checking for TSO and PSO. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, number 9035 in LNCS, pages 353–367. Springer, 2015.

2 Parosh Abdulla, Stavros Aronis, Bengt Jonsson, and Konstantinos Sagonas. Optimal dynamic partial order reduction. In *Principles of Programming Languages (POPL)*, pages 373–384. ACM, 2014.

3 Blai Bonet, Patrik Haslum, Victor Khomenko, Sylvie Thiébaux, and Walter Vogler. Recent advances in unfolding technique. *Theoretical Comp. Science*, 551:84–101, September 2014.

4 Javier Esparza and Keijo Heljanko. *Unfoldings – A Partial-Order Approach to Model Checking*. EATCS Monographs in Theoretical Computer Science. Springer, 2008.

5 Javier Esparza, Stefan Römer, and Walter Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods in System Design*, 20:285–310, 2002.

6 Cormac Flanagan and Patrice Godefroid. Dynamic partial-order reduction for model checking software. In *Principles of Programming Languages (POPL)*, pages 110–121. ACM, 2005.

7 Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem*, volume 1032 of *LNCS*. Springer, 1996.

8 Guy Gueta, Cormac Flanagan, Eran Yahav, and Mooly Sagiv. Cartesian partial-order reduction. In *Model Checking Software (SPIN)*, volume 4595 of *LNCS*, pages 95–112. Springer, 2007.

9 Henri Hansen and Xu Wang. On the origin of events: branching cells as stubborn sets. In *Proc. International Conference on Application and Theory of Petri Nets and Concurrency (ICATPN)*, volume 6709 of *LNCS*, pages 248–267. Springer, 2011.

10 Kari Kähkönen and Keijo Heljanko. Testing multithreaded programs with contextual unfoldings and dynamic symbolic execution. In *Application of Concurrency to System Design (ACSD)*, pages 142–151. IEEE, 2014.

11 Vineet Kahlon, Chao Wang, and Aarti Gupta. Monotonic partial order reduction: An optimal symbolic partial order reduction technique. In *Computer Aided Verification (CAV)*, volume 5643 of *LNCS*, pages 398–413. Springer, 2009.

12 Shmuel Katz and Doron Peled. Defining conditional independence using collapses. *Theoretical Computer Science*, 101(2):337–359, 1992.

13 Antoni Mazurkiewicz. Trace theory. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *LNCS*, pages 278–324. Springer, 1987.

14 K. L. McMillan. Using unfoldings to avoid the state explosion problem in the verification of async. circuits. In *Proc. CAV'92*, volume 663 of *LNCS*, pages 164–177. Springer, 1993.

15 Ugo Montanari and F. Rossi. Contextual nets. *Acta Informatica*, 32(6):545–596, 1995.

16 Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13(1):85–108, 1981.

17 César Rodríguez, Marcelo Sousa, Subodh Sharma, and Daniel Kroening. Unfolding-based partial order reduction. *CoRR*, abs/1507.00980, 2015.

18 `http://sv-comp.sosy-lab.org/2015/`.

19 Antti Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets 1990*, number 483 in LNCS, pages 491–515. Springer, 1991.

20 Yu Yang, Xiaofang Chen, Ganesh Gopalakrishnan, and Robert M. Kirby. Efficient stateful dynamic partial order reduction. In *Model Checking Software (SPIN)*, volume 5156 of *LNCS*, pages 288–305. Springer, 2008.

21 Xiaodong Yi, Ji Wang, and Xuejun Yang. Stateful dynamic partial-order reduction. In *Formal Methods and Sw. Eng.*, number 4260 in LNCS, pages 149–167. Springer, 2006.

Verification of Population Protocols

Javier Esparza[1], Pierre Ganty[2], Jérôme Leroux[3], and Rupak Majumdar[4]

1 **TUM, Germany**
2 **IMDEA Software Institute, Spain**
3 **LaBRI, CNRS & Université Bordeaux, France**
4 **MPI-SWS, Germany**

Abstract

Population protocols (Angluin et al., PODC, 2004) are a formal model of sensor networks consisting of identical mobile devices. Two devices can interact and thereby change their states. Computations are infinite sequences of interactions satisfying a strong fairness constraint.

A population protocol is well-specified if for every initial configuration C of devices, and every computation starting at C, all devices eventually agree on a consensus value depending only on C. If a protocol is well-specified, then it is said to compute the predicate that assigns to each initial configuration its consensus value.

While the predicates computable by well-specified protocols have been extensively studied, the two basic verification problems remain open: is a given protocol well-specified? Does a protocol compute a given predicate? We prove that both problems are decidable. Our results also prove decidability of a natural question about home spaces of Petri nets.

1998 ACM Subject Classification C.2.2 Network Protocols, D.2.4 Software/Program Verification, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases population protocols, Petri nets, parametrized verification

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.470

1 Introduction

Population protocols [2] are a model of distributed computation by anonymous, interacting finite-state agents. In each step, a fixed number of agents are chosen nondeterministically, and the agents interact and update their states according to a joint transition function. A population protocol is said to compute a predicate on the initial states of the agents if, in all fair executions, all agents eventually converge to the correct value of the predicate. An execution is fair if it is finite and cannot be extended, or it is infinite and every configuration of agent states that is reachable at infinitely many positions along the execution is also reached infinitely often along that execution.

The original motivation for population protocols was to model distributed computation in passively mobile sensors [2], but the model captures the essence of distributed computation in diverse areas such as trust propagation [7] and chemical reactions [15].

Much of the work on population protocols has concentrated on characterizing what predicates on the input values can be computed by *well-specified* protocols. A protocol is well-specified if, on every input, every fair execution eventually converges to configurations in which every agent agrees on a consensus value that depends only on the input. Angluin et al. [2] gave explicit well-specified protocols to compute every predicate definable in Presburger arithmetic. Later, Angluin et al. [4] showed that well-specified population protocols compute exactly the Presburger-definable predicates.

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 470–482

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

Since it is easy to erroneously design protocols that are not well-specified, one can ask the natural verification question: given a population protocol, is it well-specified? In this paper, we show that the well-specification problem for population protocols is decidable. We also study the correctness problem: given a protocol and a Presburger specification, does the protocol compute the specification? Our techniques show decidability of the correctness problem as well.

The semantics of a population protocol is an infinite family of finite-state transition systems, one for each possible input. Whether the protocol reaches consensus for a given input can be decided by inspecting only one of these transition systems. However, the well-specification problem asks if consensus is reached for *all* inputs, and so it is not obviously decidable; indeed, similar questions are undecidable for many parameterized systems [5]. Moreover, the set of configurations where all agents agree on a value is not upward-closed; thus, coverability-like techniques are not immediately applicable.

Our main result is a characterization of well-specification using Presburger-definable predicates. We show that for every well-specified protocol, one can find a *witness* consisting of four Presburger-definable predicates $(\mathcal{S}_0, \mathcal{S}_1, \mathcal{B}_0, \mathcal{B}_1)$ and a bounded regular language W such that:

- each predicate is inductive (closed under taking a step of the protocol),
- each initial state is either in $\mathcal{S}_0$ or in $\mathcal{S}_1$, but not in both,
- for $i \in \{0, 1\}$, all configurations of $\mathcal{B}_i$ agree on the consensus value i; moreover, $\mathcal{B}_i$ is reachable from each configuration in $\mathcal{S}_i$ using a string from W.

Using the decidability of Presburger arithmetic, we show that each condition above is decidable. Our proof of correctness uses recent results from the theory of Petri nets. We use the existence of Presburger-definable inductive sets that separate unreachable markings [11] to identify $\mathcal{S}_0$ and $\mathcal{S}_1$. We use the Presburger-definability of the mutual reachability relation [12] to identify $\mathcal{B}_0$ and $\mathcal{B}_1$. Finally, we use the theory of accelerations [14] to identify W. Along the way, we obtain an alternative proof of the theorem that well-specified protocols compute only Presburger-definable predicates.

Ultimately, our decision procedure consists of running two semi-decision procedures in parallel and does not provide a complexity upper bound. For lower bounds, we show that reachability for Petri nets can be reduced in polynomial-time to the complement of the well-specification problem.

While we focus on population protocols, our techniques also lead to new results for the theory of Petri nets. The *home space* problem asks, given a Petri net and two sets $\mathcal{I}$ and $\mathcal{H}$ of markings, if every marking reachable from $\mathcal{I}$ can also reach $\mathcal{H}$. De Frutos and Johnen [8] showed that the home space problem is decidable if $\mathcal{I}$ is a single marking and $\mathcal{H}$ is a linear set. They left the case in which $\mathcal{H}$ is a Presburger-definable set open. We make the first partial progress on this problem. Our results show that the home space problem is decidable for Presburger-definable sets $\mathcal{I}$ and $\mathcal{H}$, provided the set of markings reachable from any marking in $\mathcal{I}$ is finite.

The paper is organized as follows. Section 2 introduces population protocols. Section 3 formally defines witnesses of well-specification, shows decidability of the conditions to be met by a witness, and proves that existence of a witness implies well-specification. The proof of the converse (well-specification implies existence of a witness) is more involved. Section 4 introduces the results of Petri net theory needed for the proof, and Section 5 the proof itself. Section 6 reduces Petri net reachability to the complement of the well-specification problem. Finally, Section 7 proves the result about home spaces in Petri nets.

2 Population Protocols

A *population* on a finite set E is a mapping $P\colon E \to \mathbb{N}$ such that $P(e) > 0$ for some $e \in E$. Intuitively, $P(e)$ denotes the number of individuals of type $e \in E$ in the population. The set of all populations on E is denoted by $\mathrm{Pop}(E)$. Operations on populations, like addition or maximum, are implicitly defined component wise. Given $e \in E$, we denote by $\mathbf{e}$ the population consisting of one individual of type e, that is, the population satisfying $\mathbf{e}(e) = 1$ and $\mathbf{e}(e') = 0$ for every $e' \neq e$. The *support* of a population $P \in \mathbb{N}^E$, denoted by $\mathrm{Sup}(P)$, is the subset of E given by $\{e \in E \mid P(e) > 0\}$. A set of populations $\mathcal{C} \subseteq \mathrm{Pop}(E)$ is said to be *Presburger* if it can be denoted by a formula in *Presburger arithmetic*, i.e., in the first-order theory of addition $FO(\mathbb{N}, +)$.

▶ **Example 1.** Let $E = \{a, b\}$. The set of populations $\{P \in \mathrm{Pop}(E) \mid P(a) \geq P(b)\}$ is Presburger, since it is denoted by the Presburger formula $F(X_a, X_b) = \exists Y\colon X_a = Y + X_b$. The set $\{P \in \mathrm{Pop}(E) \mid P(a) = P(b)^2\}$ is not Presburger.

2.1 Protocol Scheme

A *protocol scheme* $\mathcal{A} = (Q, \Delta)$ consists of a finite non-empty set Q of states and a set $\Delta \subseteq Q^4$. If $(q_1, q_2, q_1', q_2') \in \Delta$, we write $(q_1, q_2) \mapsto (q_1', q_2')$ and call it a *transition*. The populations of $\mathrm{Pop}(Q)$ are called *configurations*. Intuitively, a configuration C describes a collection of identical finite-state *agents* with Q as set of states, containing $C(q)$ agents in state q for every $q \in Q$. Pairs of agents interact using transitions from Δ.[1] Formally, given two configurations C and C' and a transition $\delta = (q_1, q_2) \mapsto (q_1', q_2')$, we write $C \xrightarrow{\delta} C'$ if

$$C \geq (\mathbf{q}_1 + \mathbf{q}_2) \text{ holds, and } C' = C - (\mathbf{q}_1 + \mathbf{q}_2) + (\mathbf{q}_1' + \mathbf{q}_2') \ .$$

We write $C \xrightarrow{w} C'$ for a word $w = \delta_1 \dots \delta_k$ of transitions if there exists a sequence $C_0, \dots, C_k$ of configurations satisfying $C = C_0 \xrightarrow{\delta_1} C_1 \cdots \xrightarrow{\delta_k} C_k = C'$. In this case, we say that C' is *reachable from* C. We also write $C \to C'$ if $C \xrightarrow{\delta} C'$ for some transition $\delta \in \Delta$. We have:

▶ **Lemma 2.** *For every configuration C, the set of configurations reachable from C is finite.*

Proof. Follows immediately from the fact that an interaction does not create or destroy agents, just changes their current states. Since Q is finite, there are only finitely many configurations C' satisfying $\sum_{q \in Q} C(q) = \sum_{q \in Q} C'(q)$. ◀

Observe that $(\mathrm{Pop}(Q), \to)$ defines a directed graph with infinitely many vertices and edges. Consider the partition $\{\mathrm{Pop}(Q)_i\}_{i \geq 1}$ of $\mathrm{Pop}(Q)$, where $\mathrm{Pop}(Q)_i = \{C \in \mathrm{Pop}(Q) \mid \sum_{q \in Q} C(q) = i\}$. (Note that i starts at 1 because every population contains at least one agent.) Since interactions do not create or destroy agents, the set $\{\to_i\}_{i \geq 1}$, where $\to_i = \to \cap\, \mathrm{Pop}(Q)_i^2$, is also a partition of $\to$. Therefore $(\mathrm{Pop}(Q), \to)$ consists of the infinitely many disjoint and finite subgraphs $\{(\mathrm{Pop}(Q)_i, \to_i)\}_{i \geq 1}$.

An *execution* of $\mathcal{A}$ is a finite or infinite sequence of configurations $C_0, C_1, \dots$ such that $C_i \to C_{i+1}$ for each $i \geq 0$. An execution is *fair* if it is finite and cannot be extended, or it is infinite and for every step $C \to C'$, if C occurs infinitely often along the execution, then C' also occurs infinitely often. It follows from Lemma 2 that every execution reaches a strongly

[1] While protocol schemes model pairwise interactions only, one can model k-way interactions for a fixed $k > 2$ by adding additional states.

connected component (SCC) of $(\mathrm{Pop}(Q), \to)$ and never leaves it. We deduce the following lemma, where a bottom SCC of $(\mathrm{Pop}(Q), \to)$ is an SCC such that every edge of $\to$ whose source is in the SCC also belongs the SCC. (In particular, a single vertex with no outgoing transition forms a bottom SCC.)

▶ **Lemma 3.** *Every fair execution eventually reaches a bottom SCC of* $(\mathrm{Pop}(Q), \to)$.

Proof. If the execution is finite, then, since it cannot be extended, its last configuration is a bottom SCC with one single vertex and no outgoing transitions. If the execution is infinite, then the fairness condition forces it to eventually leave every non-bottom SCC it enters. ◀

2.2 Computation by Population Protocols

We define what it means for a protocol scheme to compute a predicate $\Pi\colon \mathrm{Pop}(\Sigma) \to \{0,1\}$, where Σ is a non-empty, finite set of *inputs*.

An *initial mapping* of a protocol scheme $\mathcal{A} = (Q, \Delta)$ is a function $I\colon \mathrm{Pop}(\Sigma) \to \mathrm{Pop}(Q)$ that maps each input population X to a configuration of $\mathcal{A}$. The set of *initial configurations* is $\mathcal{I} = \{I(X) \mid X \in \mathrm{Pop}(\Sigma)\}$. An initial mapping I is *Presburger* if the predicate $C = I(X)$, where $C \in \mathrm{Pop}(Q)$ and $X \in \mathrm{Pop}(\Sigma)$, is definable in Presburger arithmetic. An initial mapping I is *simple* if there exists a sequence $(q_\sigma)_{\sigma\in\Sigma}$ of states of Q satisfying

$$I(X) = \sum_{\sigma\in\Sigma} X(\sigma)\,\mathbf{q}_\sigma$$

for every input population X on Σ.

An *output mapping* of a protocol scheme $\mathcal{A} = (Q, \Delta)$ is a function $O\colon \mathrm{Pop}(Q) \to \{0, \bot, 1\}$ that associates to each configuration C of $\mathcal{A}$ an output value in $\{0, \bot, 1\}$. A population C on Q such that $O(C) = b$ for some $b \in \{0, \bot, 1\}$ is called a *b-population*. An output mapping O is *Presburger* if the predicate $O(C) = b$ where $C \in \mathrm{Pop}(Q)$ and $b \in \{0, 1\}$ is definable in Presburger arithmetic. An output mapping O is *simple* if there exists a partition (Q_0, Q_1) of Q such that

$$O(C) \;=\; \begin{cases} 0 & \text{if } \mathrm{Sup}(C) \subseteq Q_0 \\ 1 & \text{if } \mathrm{Sup}(C) \subseteq Q_1 \\ \bot & \text{otherwise} \end{cases}$$

for every configuration C. Notice that O is well-defined because $\mathrm{Sup}(C) \neq \emptyset$. An execution $C_0, C_1, \ldots$ *stabilizes to* b for a given $b \in \{0, \bot, 1\}$ if there exists $n \in \mathbb{N}$ such that $O(C_m) = b$ for every $m \geq n$ (if the execution is finite, then this means for every m between n and the length of the execution). So, intuitively, an execution stabilizes to b if from some moment on all agents stay within the subset of states with output b. Notice that there may be many different executions from a given configuration C_0, each of which may stabilize to 0, 1, or $\bot$, or not stabilize at all.

Most papers only consider population protocols with simple initial and output mappings. We study the more general class of Presburger initial and output mappings. In our general setting, a *population protocol* is a triple $(\mathcal{A}, \mathtt{I}, \mathtt{O})$, where $\mathcal{A}$ is a protocol scheme, $\mathtt{I}(X, C)$ is a formula in Presburger arithmetic denoting a Presburger initial mapping $C = I(X)$, and $\mathtt{O}(C, b)$ is a formula in Presburger arithmetic denoting a Presburger output mapping $O(C) = b$. This definition encompasses population protocols with leader [3]. In these protocols the initial configuration contains one agent, called the *leader*, occupying a distinguished initial

state q_l not initially occupied by any other agent. This corresponds to the initial mapping $I(X) = \mathbf{q}_l + \sum_{\sigma\in\Sigma} X(\sigma)\,\mathbf{q}_\sigma$ which is obviously Presburger.

A population protocol $(\mathcal{A}, \mathtt{I}, \mathtt{O})$ is *well-specified* if for every input population $X \in \mathrm{Pop}(\Sigma)$, every fair execution of $\mathcal{A}$ starting at $I(X)$ stabilizes to the same value, and *ill-specified* otherwise. A population protocol $(\mathcal{A}, \mathtt{I}, \mathtt{O})$ *computes* a predicate Π if every fair execution of $\mathcal{A}$ starting at $I(X)$ stabilizes to $\Pi(X)$ for every $X \in \mathrm{Pop}(\Sigma)$.

The *well-specification problem* asks if a given protocol $(\mathcal{A}, \mathtt{I}, \mathtt{O})$ is well-specified. The *correctness problem* asks if a given population protocol $(\mathcal{A}, \mathtt{I}, \mathtt{O})$ computes a given Presburger predicate Π. Note that the correctness problem does not assume $(\mathcal{A}, \mathtt{I}, \mathtt{O})$ to be well-specified. Consequently, if $(\mathcal{A}, \mathtt{I}, \mathtt{O})$ does not compute Π then either the population protocol is ill-specified; otherwise it stabilizes to b for some input $X \in \mathrm{Pop}(\Sigma)$ such that $\Pi(X) = 1 - b$.

3 A Decidable Criterion for Well-Specification

In this paper, the well-specification problem is shown to be decidable thanks to a decidable criterion based on Presburger arithmetic. This criterion is defined as follows. Let $\mathcal{A} = (Q, \Delta)$ be a protocol scheme. A set $\mathcal{C}$ of configurations of $\mathcal{A}$ is said to be *inductive* if $C \in \mathcal{C}$ and $C \to C'$ implies $C' \in \mathcal{C}$. Given a language $W \subseteq \Delta^*$ and a set $\mathcal{C}$ of configurations, we denote by $pre_{\mathcal{A}}(\mathcal{C}, W)$ the set of configurations C such that $C \xrightarrow{w} C'$ for some word $w \in W$ and some configuration $C' \in \mathcal{C}$.

▶ **Definition 4.** Let $\mathcal{A} = (Q, \Delta)$ be a protocol scheme. A *witness of well-specification* of the population protocol $(\mathcal{A}, \mathtt{I}, \mathtt{O})$ is a tuple $(\mathtt{S}_0, \mathtt{S}_1, \mathtt{B}_0, \mathtt{B}_1, w_1, \ldots, w_k)$, where $\mathtt{S}_0, \mathtt{S}_1, \mathtt{B}_0, \mathtt{B}_1$ are predicates in Presburger arithmetic denoting Presburger sets of configurations $\mathcal{S}_0, \mathcal{S}_1, \mathcal{B}_0, \mathcal{B}_1$, and $w_1, \ldots, w_k$ are words in Δ^* denoting the language $W = w_1^* \ldots w_k^*$, such that:
(1) $\mathcal{S}_0, \mathcal{S}_1, \mathcal{B}_0, \mathcal{B}_1$ are inductive.
(2) The pair $(\mathcal{I}_0, \mathcal{I}_1)$, where $\mathcal{I}_0 = \mathcal{S}_0 \cap \mathcal{I}$ and $\mathcal{I}_1 = \mathcal{S}_1 \cap \mathcal{I}$, is a partition of $\mathcal{I}$.
(3) $\mathcal{B}_0$ is a set of 0-populations and $\mathcal{S}_0 \subseteq pre_{\mathcal{A}}(\mathcal{B}_0, W)$.
(4) $\mathcal{B}_1$ is a set of 1-populations and $\mathcal{S}_1 \subseteq pre_{\mathcal{A}}(\mathcal{B}_1, W)$.

▶ **Lemma 5.** *The set of witnesses of well-specification is recursive.*

Proof. Let $(\mathcal{A}, \mathtt{I}, \mathtt{O})$ and $(\mathtt{S}_0, \mathtt{S}_1, \mathtt{B}_0, \mathtt{B}_1, w_1, \ldots, w_k)$ be as in Definition 4. We show that conditions (1)–(4) can be effectively expressed in Presburger arithmetic. For (1), a set $\mathcal{M}$ of configurations denoted by a predicate $\mathtt{M}(C)$ in Presburger arithmetic is inductive iff the following Presburger formula is valid:

$$\forall C, C' \colon \mathtt{M}(C) \wedge C \to C' \Rightarrow \mathtt{M}(C') \ .$$

So the inductiveness of $\mathcal{S}_0, \mathcal{S}_1, \mathcal{B}_0, \mathcal{B}_1$ is expressible. For (2), $(\mathcal{I}_0, \mathcal{I}_1)$ is a partition of $\mathcal{I}$ iff

$$\forall C \colon (\exists X \colon \mathtt{I}(X, C)) \Leftrightarrow \big((\mathtt{I}_0(C) \wedge \neg \mathtt{I}_1(C)) \vee (\neg \mathtt{I}_0(C) \wedge \mathtt{I}_1(C)\big)$$

is valid, where $\mathtt{I}_b(C) = (\exists X \colon \mathtt{I}(X, C)) \wedge \mathtt{S}_b(C)$. For (3-4), $\mathcal{B}_b$ is a set of b-populations iff

$$\forall C \colon \mathtt{B}_b(C) \Rightarrow \mathtt{O}(C, b)$$

is valid. It remains to express $\mathcal{S}_b \subseteq pre_{\mathcal{A}}(\mathcal{B}_b, W)$. Observe that for every word $w \in \Delta^*$, the relation $\xrightarrow{w^*}$ defined by $C \xrightarrow{w^*} C'$ if $C \xrightarrow{w^n} C'$ for some $n \in \mathbb{N}$ is effectively definable in Presburger arithmetic. (For $w = \delta$, where $\delta = (q_1, q_2) \mapsto (q_1', q_2')$, this follows easily from $C' = C - (\mathbf{q}_1 + \mathbf{q}_2) + (\mathbf{q}_1' + \mathbf{q}_2')$. For the general case, see [14].) So the inclusion holds iff

$$\forall C_0 \colon \big(\mathtt{S}_b(C_0) \Rightarrow \exists C_1, \ldots, C_k \colon C_0 \xrightarrow{w_1^*} C_1 \cdots \xrightarrow{w_k^*} C_k \ \wedge \mathtt{B}_b(C_k)\big)$$

is valid. ◀

3.1 The Criterion is Sound

We show that every population protocol satisfying the criterion is well-specified.

▶ **Lemma 6.** *Every population protocol* $(\mathcal{A}, \mathtt{I}, \mathtt{O})$ *admitting a witness* $(\mathtt{S}_0, \mathtt{S}_1, \mathtt{B}_0, \mathtt{B}_1, w_1, \ldots, w_k)$ *of well-specification is well-specified. Moreover, in this case the population protocol computes the predicate* $\Pi : \mathrm{Pop}(\Sigma) \to \{0,1\}$ *defined by:*

$$\Pi(X) = \begin{cases} 0 & \text{if } \exists C\colon \mathtt{I}(X,C) \wedge \mathtt{S}_0(C) \\ 1 & \text{if } \exists C\colon \mathtt{I}(X,C) \wedge \mathtt{S}_1(C) \end{cases} .$$

Proof. Let $\mathcal{S}_0, \mathcal{S}_1, \mathcal{B}_0, \mathcal{B}_1$ be the Presburger sets of configurations denoted by $\mathtt{S}_0, \mathtt{S}_1, \mathtt{B}_0, \mathtt{B}_1$, respectively. Let $W = w_1^* \ldots w_k^*$. Since $\mathcal{I}_0$ and $\mathcal{I}_1$ form a partition of $\mathcal{I}$, it suffices to prove that every fair execution starting at $\mathcal{I}_b$ stabilizes to b. Let $C \in \mathcal{I}_b$ and let $C_0, C_1, \ldots$ be a fair execution starting at C. Lemma 3 shows that the execution ends up in a bottom SCC. Hence, there exists $n \in \mathbb{N}$ such that C_n is in a bottom SCC. As $\mathcal{S}_b$ is inductive, it follows that C_n is in this set. Moreover, as $\mathcal{S}_b \subseteq \mathit{pre}_{\mathcal{A}}(\mathcal{B}_b, W)$, there exists a word $w \in W$ and a configuration $C' \in \mathcal{B}_b$ such that $C_n \xrightarrow{w} C'$. Since C_n is in a bottom SCC, there exists a word $w' \in \Delta^*$ such that $C' \xrightarrow{w'} C_n$. Now, let $m \geq n$. Since C_m is reachable from C_n, it follows that C_m is reachable from C'. As $C' \in \mathcal{B}_b$ and $\mathcal{B}_b$ is inductive, it follows that $C_m \in \mathcal{B}_b$. As $\mathcal{B}_b$ is a set of b-populations, it follows that $O(C_m) = b$; thus, the execution stabilizes to b. ◀

In the rest of the paper we prove the converse of Lemma 6: every well-specified protocol admits a witness of well-specification. But before, we close the section with an example.

3.2 Example

Let $\Sigma = \{\sigma\}$, and consider the predicate $\Pi : \mathrm{Pop}(\Sigma) \to \{0,1\}$, where $\Pi(X)$ is the parity of $X(\sigma)$. In other words, $\Pi(X) = 0$ if $X(\sigma)$ is even, and $\Pi(X) = 1$ otherwise. This predicate is computed by a simple well-specified population protocol. The protocol scheme $\mathcal{A} = (Q, \Delta)$ has $Q = \{A_0, A_1, P_0, P_1\}$ as set of states. We call agents in A_0 and A_1 *active*, and those in P_0 and P_1 *passive*. Further, we say that agents in A_b and P_b carry the value b. The set Δ of transitions is $\{\delta_{x,y}, \delta_x \mid x, y \in \{0,1\}\}$. Transitions $\delta_{x,y}$ allow two active agents to add their numbers modulo 2 and deactivate one of them:

$$\delta_{x,y} \;=\; (A_x, A_y) \mapsto (A_{x+y}, P_{x+y}) \ .$$

Transitions δ_x allow an active agent to change the value of a passive agent:

$$\delta_x \;=\; (A_x, P_{1-x}) \mapsto (A_x, P_x) \ .$$

The simple population protocol computing Π is given by $(\mathcal{A}, \mathtt{I}, \mathtt{O})$, where the simple input mapping is defined by

$$I(X) \;=\; X(\sigma)\mathbf{A}_0$$

and the simple output mapping by $Q_0 = \{A_0, P_0\}$ and $Q_1 = \{A_1, P_1\}$.

Let us provide a witness of well-specification explaining why the protocol computes Π. We choose $\mathtt{B}_0(C) = \big(C(A_0) = 1 \wedge C(A_1) = 0 \wedge C(P_1) = 0\big)$ and $\mathtt{B}_1(C) = \big(C(A_1) = 1 \wedge C(A_0) = 0 \wedge C(P_0) = 0\big)$. Notice that the set of configurations $\mathcal{B}_b$ denoted by $\mathtt{B}_b$ is inductive for every $b \in \{0,1\}$. In fact, since a configuration $C \in \mathcal{B}_b$ only has one active agent, and all agents

carry the same value b, no transition in Δ is enabled at C. Further, we define $\mathtt{S}_0(C)$ as "$C(A_1)$ is even" and $\mathtt{S}_1(C)$ as "$C(A_1)$ is odd". Inspection of the transitions in Δ immediately shows that the sets $\mathcal{S}_0$ and $\mathcal{S}_1$ denoted by these two Presburger predicates are inductive. Notice that $\mathcal{I} \cap \mathcal{S}_0$ and $\mathcal{I} \cap \mathcal{S}_1$ is a partition of $\mathcal{I}$.

It remains to define the language W. Let us first describe a strategy to reach $\mathcal{B}_0 \cup \mathcal{B}_1$ from any configuration C. We first execute the transition $\delta_{0,0}$ as long as possible, until there is at most one active agent carrying a 0. Then we execute $\delta_{1,1}$ as long as possible, until there is at most one active agent carrying a 1. Then we execute $\delta_{1,0}$ if possible, reaching a configuration with exactly one active agent carrying a value b. Finally, we execute δ_0 as long as possible, followed by δ_1 as long as possible, leading to a configuration in which every passive agent also carries the value b. The language W models this strategy:

$$W = \delta_{0,0}^* \; \delta_{1,1}^* \; \delta_{1,0}^* \; \delta_0^* \; \delta_1^* \;\; .$$

4 Petri Net Theory for the Population Protocols Aficionados

The computation of a population protocol can be simulated by an associated Petri net. This allows us to apply results on Petri nets to population protocols.

A Petri net $N = (P, T, F)$ consists of a finite set P of *places*, a finite set T of *transitions*, and a *flow function* $F\colon (P \times T) \cup (T \times P) \to \mathbb{N}$. A *marking* is a mapping from P to $\mathbb{N}$, i.e. a mapping in $\mathbb{N}^P$. A transition $t \in T$ is *enabled at* marking M, written $M\,[t\rangle$, if $F(p,t) \leq M(p)$ for each place $p \in P$. A transition t that is enabled at M can *fire*, yielding a marking M' such that $M'(p) = M(p) - F(p,t) + F(t,p)$ for each $p \in P$. We write this fact as $M\,[t\rangle\,M'$. We extend enabledness and firing inductively to words of transitions as follows. Let $w = t_1 \dots t_k$ be a finite word of transitions $t_j \in T$. We define $M\,[w\rangle\,M'$ if, and only if, there exists a sequence $M_0, \dots, M_k$ of markings such that $M = M_0\,[t_1\rangle\,M_1 \cdots [t_k\rangle\,M_k = M'$. In that case, we say that M' is *reachable from* M.

4.1 From Population Protocols To Petri Nets

Given a protocol scheme $\mathcal{A} = (Q, \Delta)$, we define the Petri net $N(\mathcal{A}) = (Q, \Delta, F)$, whose places and transitions are the states and transitions of the protocol, respectively, and where F is defined for every transition $\delta = (q_1, q_2) \mapsto (q_1', q_2')$ in Δ and every state $q \in Q$ by $F(q,\delta) = \mathbf{q}_1(q) + \mathbf{q}_2(q)$ and $F(\delta, q) = \mathbf{q}_1'(q) + \mathbf{q}_2'(q)$. Note that a configuration of the protocol scheme $\mathcal{A}$ is a marking of the Petri net $N(\mathcal{A})$. Further, whenever $C \xrightarrow{\delta} C'$ for configurations C and C', we have $C\,[\delta\rangle\,C'$ in the Petri net, and vice versa.

The correspondence between $\mathcal{A}$ and $N(\mathcal{A})$ allows us to transfer results from Petri nets to population protocols. Next, we briefly recall the results we need.

4.2 Acceleration Technique

Given a Petri net $N = (P, T, F)$, a set $\mathcal{M}$ of markings, and a language $W \subseteq T^*$, we introduce the sets:

$$post_N(\mathcal{M}, W) = \{M' \in \mathbb{N}^P \mid \exists M \in \mathcal{M}\; \exists w \in W\colon M\,[w\rangle\,M'\}$$
$$pre_N(\mathcal{M}, W) = \{M \in \mathbb{N}^P \mid \exists M' \in \mathcal{M}\; \exists w \in W\colon M\,[w\rangle\,M'\} \;\; .$$

When $W = T^*$ these sets are denoted by $post_N^*(\mathcal{M})$ and $pre_N^*(\mathcal{M})$, respectively.

The theory of acceleration (see for instance [14]) will provide a simple way for extracting the language W introduced in Definition 4. A language $W \subseteq T^*$ is said to be *bounded* [10] if

there exists a sequence $w_1, \ldots, w_k$ of words in T^* such that $W \subseteq w_1^* \ldots w_k^*$. The following result will be useful for extracting the language W introduced in Definition 4.

▶ **Theorem 7** ([14, Corollary XI.3]). *For every Petri net $N = (P, T, F)$ and for every Presburger sets of markings $\mathcal{S}$ and $\mathcal{B}$ such that $\mathcal{S} \subseteq pre_N^*(\mathcal{B})$, there exists a bounded language $W \subseteq T^*$ such that $\mathcal{S} \subseteq pre_N(\mathcal{B}, W)$.*

4.3 Separators For Reachability Problems

Recently, the reachability problem for Petri nets was proved to be decidable using a very simple algorithm based on Presburger inductive sets of markings. Let us recall that a set $\mathcal{M}$ of markings is *inductive* for a Petri net $N = (P, T, F)$, if $post_N(\mathcal{M}, T) \subseteq \mathcal{M}$. The following result will provide the sets $\mathcal{S}_0$ and $\mathcal{S}_1$ introduced in Definition 4.

▶ **Theorem 8** ([13, Lemma 9.1]). *For every Petri net N and for every Presburger set of markings $\mathcal{M}$ and $\mathcal{M}'$ such that $post_N^*(\mathcal{M}) \cap \mathcal{M}' = \emptyset$, there exists a Presburger inductive set of markings $\mathcal{S}$ for N such that $\mathcal{M} \subseteq \mathcal{S}$ and $\mathcal{S} \cap \mathcal{M}' = \emptyset$.*

4.4 Mutual Reachability Relations

The *mutual reachability relation* of a Petri net N is the binary relation over the markings that contains the pair (M, M') if M' is reachable from M and M is reachable from M'. Intuitively, M and M' coincide and otherwise they are in the same SCC for the reachability graph. The following theorem will be useful for extracting the sets $\mathcal{B}_0$ and $\mathcal{B}_1$ introduced in Definition 4.

▶ **Theorem 9** ([12]). *For every Petri net N, the mutual reachability relation is effectively definable in Presburger arithmetic.*

4.5 Decomposable sets

In this section, we introduce a new result for Petri nets. This result will be used for characterizing the sets $\mathcal{I}_0$ and $\mathcal{I}_1$ introduced in Definition 4. The proof of this result is based on the geometrical characterization of the reachability sets of Petri nets based on almost semi-linear sets and decomposable sets (see [12] for definitions). It uses the technical result that if the union of two disjoint decomposable sets $\mathbf{X}, \mathbf{Y}$ is Presburger definable, then both $\mathbf{X}$ and $\mathbf{Y}$ are Presburger definable as well. We defer the details to the full version of the paper.

▶ **Theorem 10.** *For every Petri net N, and for every Presburger sets of markings $\mathcal{B}_0, \mathcal{B}_1$ and $\mathcal{I}$ such that $\mathcal{I}_0 = \mathcal{I} \cap pre_N^*(\mathcal{B}_0)$ and $\mathcal{I}_1 = \mathcal{I} \cap pre_N^*(\mathcal{B}_1)$ is a partition of $\mathcal{I}$, it follows that $\mathcal{I}_0$ and $\mathcal{I}_1$ are Presburger.*

5 The Criterion is Complete

We use the previous results to prove that every well-specified protocol admits a witness.

5.1 Characterization of Bottom Strongly Connected Components

Given a protocol scheme $\mathcal{A} = (Q, \Delta)$, a bottom SCC of the graph $(\mathrm{Pop}(Q), \to)$ of $\mathcal{A}$ is said to be *b-bottom* ($b \in \{0, 1\}$) if all its configurations, which are called bottom configurations, are b-populations. When this holds, the configurations of the SCC are called *b-bottom configurations*. We denote the sets of bottom configurations and b-bottom configurations by $\mathcal{B}$ and $\mathcal{B}_b$, respectively.

▶ **Proposition 11.** *Given a protocol scheme, the sets $\mathcal{B}$, $\mathcal{B}_0$ and $\mathcal{B}_1$ are effectively Presburger.*

Proof. We show that the predicate $\mathtt{B}(C)$ associated to the set of bottom configurations is definable in Presburger arithmetic. Let us introduce the predicate $\mathtt{MR}(C, C')$ associated to the mutual reachability relation. Theorem 9 shows that $\mathtt{MR}(C, C')$ is effectively Presburger. Now, we just observe that C is a bottom configuration iff for every configuration C' such that C and C' are mutually reachable and for every C'' such that $C' \to C''$, we have C and C'' are also mutually reachable:

$$\mathtt{B}(C) = \forall C' \, \forall C'' \colon (\mathtt{MR}(C, C') \wedge C' \to C'') \Rightarrow \mathtt{MR}(C, C'') \ .$$

We claim that $\mathcal{B}_b$ is a Presburger set of configurations. To prove this, we just notice that $\mathcal{B}_b$ is denoted by the following formula:

$$\mathtt{B}_b(C) = \mathtt{B}(C) \wedge \forall C' \colon \mathtt{MR}(C, C') \Rightarrow \mathtt{O}(C', b) \ . \qquad \blacktriangleleft$$

5.2 The final piece

In the rest of this section, we show that a population protocol is well-specified if, and only if, it admits a witness of well-specification. We deduce from this characterization that the well-specification problem, and the correctness problem are decidable.

▶ **Theorem 12.** *A population protocol is well-specified iff it admits a witness of well-specification.*

Proof. Lemma 6 shows that a population protocol that admits a witness of well-specification is well-specified. Conversely, let us consider a population protocol $(\mathcal{A}, \mathtt{I}, \mathtt{O})$ that is well-specified. We define $\mathcal{B}_0$ and $\mathcal{B}_1$ as the 0-bottom configurations and 1-bottom configurations, respectively. Proposition 11 shows that these sets are Presburger. Notice these two sets are also inductive.

Let us show that $(\mathcal{I}_0, \mathcal{I}_1)$ defined by $\mathcal{I}_b = \mathcal{I} \cap \mathit{pre}^*_{\mathcal{A}}(\mathcal{B}_b)$ is a partition of $\mathcal{I}$. Since the population protocol is well specified, it follows that $\mathcal{I}_0 \cap \mathcal{I}_1 = \emptyset$. Now let C be an initial configuration in $\mathcal{I}$. Notice that there exists at least one fair execution $C_0, C_1, \ldots$ with $C_0 = C$ that stabilizes to b. Lemma 3 shows that the execution ends up in a bottom SCC. It follows that there exists $n \in \mathbb{N}$ such that C_n is a bottom configuration. Thanks to the fairness of the execution, all the configurations of the strongly connected component of C_n are b-populations. Thus $C_n \in \mathcal{B}_b$. We have proved that $C \in \mathcal{I}_b$. Thus $(\mathcal{I}_0, \mathcal{I}_1)$ is a partition of $\mathcal{I}$. Following Section 4.1, define $N(\mathcal{A})$ as the Petri net associated with $\mathcal{A}$. From Theorem 10, we derive that $\mathcal{I}_0$ and $\mathcal{I}_1$ are Presburger.

Since the population protocol is well-specified, it follows that $\mathit{post}^*_{\mathcal{A}}(\mathcal{I}_0) \cap (\mathcal{B} \backslash \mathcal{B}_0)$ is empty. Hence $\mathit{post}^*_{N(\mathcal{A})}(\mathcal{I}_0) \cap (\mathcal{B} \backslash \mathcal{B}_0)$ is also empty and Theorem 8 shows that there exists a Presburger inductive set of markings and, by extension, configurations $\mathcal{S}_0$ such that $\mathcal{I}_0 \subseteq \mathcal{S}_0$ and such that $\mathcal{S}_0 \cap (\mathcal{B} \backslash \mathcal{B}_0)$ is empty. Let us prove that $\mathcal{S}_0 \subseteq \mathit{pre}^*_{\mathcal{A}}(\mathcal{B}_0)$. Let C be a configuration in $\mathcal{S}_0$ and let us consider a fair execution $C_0, C_1, \ldots$ starting from $C_0 = C$. Lemma 3 shows that the execution ends up in a bottom SCC. It follows that there exists $n \in \mathbb{N}$ such that C_n is a bottom configuration. As $\mathcal{S}_0$ is inductive, it holds that $C_n \in \mathcal{S}_0$. Moreover, as $\mathcal{S}_0 \cap (\mathcal{B} \backslash \mathcal{B}_0)$ is empty, we derive $C_n \in \mathcal{B}_0$. We have proved that $\mathcal{S}_0 \subseteq \mathit{pre}^*_{\mathcal{A}}(\mathcal{B}_0)$. Theorem 7 shows that there exists a bounded language $W_0 \subseteq \Delta^*$ such that $\mathcal{S}_0 \subseteq \mathit{pre}_{N(\mathcal{A})}(\mathcal{B}_0, W_0)$, hence the same holds for $\mathcal{A}$. Symmetrically, there exists a Presburger inductive set of configurations $\mathcal{S}_1$ such that $\mathcal{I}_1 \subseteq \mathcal{S}_1$ and a bounded language $W_1 \subseteq \Delta^*$ such that $\mathcal{S}_1 \subseteq \mathit{pre}_{\mathcal{A}}(\mathcal{B}_1, W_1)$. Since W_0 and W_1 are bounded languages, it follows that $W = W_0 \cup W_1$ is also a bounded language.

Hence, there exists a sequence of words $w_1, \ldots, w_k$ in Δ^* such that $W \subseteq w_1^* \ldots, w_k^*$. We have proved that $(\mathtt{S}_0, \mathtt{S}_1, \mathtt{B}_0, \mathtt{B}_1, w_1, \ldots, w_k)$ is a witness of well-specification. ◀

That well-specified population protocols can compute Presburger predicates was shown by Angluin et al. [2] using a direct construction. Showing that well-specified population protocols can not compute anything else than Presburger predicates was harder, and first proved by Angluin, Aspnes and Eisenstat [4]. Our constructions provide an alternate proof.

▶ **Corollary 13.** *The well-specification problem and the correctness problem are decidable. Moreover, well-specified population protocols compute Presburger predicates, and we can effectively compute formulas in Presburger arithmetic denoting the predicates computed by well-specified population protocols.*

Proof. Notice that if a population protocol is ill-specified there exists a witness of this property given by an initial input population X in $\mathrm{Pop}(\Sigma)$ and a configuration C satisfying $\mathtt{I}(X, C)$ such that not all the bottom configurations reachable from C are b-populations for some $b \in \{0, 1\}$.

In particular, enumerating the finite graphs $(\mathrm{Pop}(Q)_i, \to_i)$, and checking, for each, whether it contains a witness of ill-specification shows that the problem of deciding if a population protocol is ill-specified is recursively enumerable.

By Theorem 12, when a population protocol is well-specified, the algorithm that enumerates all the tuples $(\mathtt{S}_0, \mathtt{S}_1, \mathtt{B}_0, \mathtt{B}_1)$ of predicates in Presburger arithmetic, and all the finite sequences $w_1, \ldots, w_k$ of words in Δ^* and checks using Lemma 5 that we have a witness of well-specification, will eventually terminate with such a witness. It follows that the well-specification problem is recursively enumerable. Moreover, in that case, from the computed witness, we derive a predicate in Presburger arithmetic denoting the computed predicate Π using Lemma 6. Together with the recursive enumerability of ill-specification above, it follows that the problem is decidable. ◀

Clement et al. [6] proved the decidability of the well-specification problem when the number of agents is fixed. Corollary 13 shows decidability of the same problem but for an arbitrary number of agents.

6 Lower Bounds

Finally, we show hardness for the well-specification problem by showing a polynomial-time reduction from Petri net reachability to its complement.

▶ **Theorem 14.** *The reachability problem for Petri nets is polynomially reducible to the complement of the well-specification problem and the complement of the correctness problem for population protocols (even with simple output mappings).*

Proof. We proceed by means of a sequence of polynomial time reductions so as to reduce the reachability problem for Petri nets to the problem of reaching, in a Petri Net $N = (P, T, F)$ with initial marking M_0, a marking M with no tokens in $z \in P$, i.e. $M(z) = 0$. Furthermore, the reduction is such that:

(a) $M_0(z) > 0$,
(b) N is deadlock-free, and
(c) every transition of N has at least one output place, and at most two input and two output places.
(d) N contains no two transitions with the same set of input and output places

The details of the reductions are standard and omitted.

Then we construct a population protocol $(\mathcal{A}, \mathtt{I}, \mathtt{O})$ with semi-linear initial mapping. We first describe the protocol scheme $\mathcal{A} = (Q, \Delta)$. The set Q of states of the protocol contains

- a state q_p for every place $p \in P$;
- a state q_t for every transition $t \in T$; and
- two states *Source* and *Sink*.

Following (d), we write $t = (P_1, P_2)$ to denote that transition t has P_1 as set of input places and P_2 as set of output places. The set Δ of transitions contains

(1) for every Petri net transition $t = (\{p_1, p_2\}, \{p_3, p_4\})$, two protocol transitions $(q_{p_1}, q_{p_2}) \mapsto (q_t, \mathit{Sink})$ and $(q_t, \mathit{Source}) \mapsto (q_{p_3}, q_{p_4})$;

(2) for every Petri net transition $t = (\{p_1, p_2\}, \{p_3\})$, two protocol transitions $(q_{p_1}, q_{p_2}) \mapsto (q_t, \mathit{Sink})$ and $(q_t, \mathit{Source}) \mapsto (q_{p_3}, \mathit{Sink})$;

(3) for every Petri net transition $t = (\{p_1\}, \{p_2, p_3\})$, one protocol transition $(q_{p_1}, \mathit{Source}) \mapsto (q_{p_2}, q_{p_3})$; and

(4) for every Petri net transition $t = (\{p_1\}, \{p_2\})$, one protocol transition $(q_{p_1}, \mathit{Source}) \mapsto (q_{p_2}, \mathit{Sink})$;

(5) a transition $(q_p, q_z) \mapsto (\mathit{Sink}, q_z)$ for each place $p \neq z$.

This completes the description of $\mathcal{A}$.

The output mapping O, which is simple, is given by the partition Q_0, Q_1 of Q such that $Q_0 = \{z, \mathit{Sink}\}$. The initial mapping $I\colon \mathrm{Pop}(\Sigma) \to \mathrm{Pop}(Q)$ is defined as follows. The set Σ is a singleton $\{\sigma\}$, and I assigns to the number n – a population of $\mathrm{Pop}(\{\sigma\})$ – the configuration that puts

- n agents in *Source*;
- $M_0(p)$ agents in q_p for every place p; and
- 0 agents elsewhere.

Observe that I is a semi-linear mapping.

The transitions of (1)–(4) simulate the firing of t (in the case of (1) and (2), firing t is simulated by the occurrence, one after the other, of two protocol transitions). In all cases, simulating the firing of t requires one agent to leave the *Source* state. On the other hand, no agents ever enter *Source*. Hence each execution of $(\mathcal{A}, \mathtt{I}, \mathtt{O})$ contains only finitely many occurrences of transitions of (1)–(4). Further, since every transition of (5) moves an agent to *Sink*, and no agents ever leave *Sink*, the transitions of (5) also occur only finitely often. Therefore all executions of $(\mathcal{A}, \mathtt{I}, \mathtt{O})$ are finite.

Assume that some reachable marking M of N satisfies $M(z) = 0$. Let $\tau \in T^*$ be such that $M_0 \,[\tau\rangle\, M$, and let k be the length of τ. Since $M_0(z) > 0$, we have $k > 0$. We claim that $\mathcal{A}$ has a fair (finite) execution from $I(k\,\boldsymbol{\sigma})$ that does not stabilize. Consider the execution that starts by simulating τ through transitions (1)–(4). At the end of this simulation the protocol reaches a configuration C such that $C(\mathit{Source}) = C(q_z) = 0$ and $C(\mathit{Sink}) > 0$. Observe that C cannot be extended because $C(\mathit{Source}) = 0$ disables all transitions (1)–(4) and $C(q_z) = 0$ disables all transitions (5). Further, since every transition has at least one output place, the configuration satisfies $C(q_p) > 0$ for some $p \neq z$. Since $\mathit{Sink} \in Q_0$ and $\{q_p \mid p \in P\} \subseteq Q_1$, we have that $O(C) = \bot$, hence that $(\mathcal{A}, \mathtt{I}, \mathtt{O})$ is ill-specified.

Assume now that every reachable marking M of N satisfies $M(z) > 0$. Let $C_0 C_1 \ldots$ be an arbitrary fair execution of $(\mathcal{A}, \mathtt{I}, \mathtt{O})$. As shown above, there is a configuration C_j such that from that moment on C_j disable all transitions (1)–(4). In particular since N is deadlock-free, we necessarily have $C_j(\mathit{Source}) = 0$. Because some transitions of (5) might be enabled at C_j, we extend the execution by firing them as many times as possible. This can occur only finitely many times and yield a configuration C_ℓ which cannot be extended further – all

transitions of (1)–(5) are disabled – and in which all agents are in state q_z or *Sink*. We thus find that $\mathrm{Sup}(C_\ell) \subseteq Q_0$, hence that $O(C_\ell) = 0$ and finally that $C_0 \dots C_\ell$ is a fair execution that converges to 0. Since we picked an arbitrary fair execution we conclude that every fair execution stabilizes to 0, and therefore $(\mathcal{A}, \mathtt{I}, \mathtt{O})$ is well-specified.

The same reduction shows hardness for the complement of the correctness problem for the predicate *false*. ◀

7 Home Spaces

As a byproduct of our main result, we present a new theorem on home spaces of Petri nets. Let N be a Petri net, and let $\mathcal{I}, \mathcal{H}$ be two sets of markings of N. We say that $\mathcal{H}$ is a *home space* of N with respect to $\mathcal{I}$ if $post^*_N(\mathcal{I}) \subseteq pre^*_N(\mathcal{H})$, that is, if $\mathcal{H}$ can be reached from any marking reachable from $\mathcal{I}$. The *home space problem* for a given triple $(N, \mathcal{I}, \mathcal{H})$ asks whether $\mathcal{H}$ is a home space of N with respect to $\mathcal{I}$.

De Frutos and Johnen [8] have proved that the home space problem is decidable when $\mathcal{I}$ is a singleton and $\mathcal{H}$ is a *linear set*, that is, a set of the form $\{M_0 + n_1 M_1 + \cdots + n_k M_k \mid n_1, \dots, n_k \in \mathbb{N}\}$ for a given *root marking* M_0 and a given finite set $\{M_1, \dots, M_k\}$ of *periods*. They also extend the result to finite unions of linear sets *having the same periods*. While every such set is a Presburger set, the converse does not hold, and De Frutos and Johnen [8] explicitly leave the case of arbitrary Presburger sets $\mathcal{H}$ open.

We prove decidability of the home space problem for triples $(N, \mathcal{I}, \mathcal{H})$ where $\mathcal{I}$ and $\mathcal{H}$ are arbitrary Presburger sets, and the net N satisfies the following condition: for every marking $M_0 \in \mathcal{I}$, the set $post^*_N(\{M_0\})$ is finite. Observe that this condition is met by Petri nets modelling parameterized systems, as in the many-process systems of German and Sistla [9, 1]. Indeed, in these systems each token of $M_0 \in \mathcal{I}$ models a finite-state process, and, since the systems have no dynamic process creation, the number of tokens does not change while the net evolves. So, while our result does not close the open problem left by De Frutos and Johnen, it provides a partial answer, and the first new result in the area since 1989.

If $post^*_N(\{M_0\})$ is finite for every $M_0 \in \mathcal{I}$, then each reachable marking can reach a bottom SCC. This is the fact we exploit. Notice that this fact no longer holds for arbitrary Petri nets. For instance, it is easy to exhibit a Petri net whose reachability graph is an infinite line, and so has no bottom SCC.

▶ **Lemma 15.** *Let N be a net, and let $\mathcal{B}$ be the set of bottom markings of N, i.e., the set of markings M that are reachable from any marking reachable from M. Let $\mathcal{I}$ be a set of markings of N such that $post^*_N(\{M_0\})$ is finite for every $M_0 \in \mathcal{I}$. A set $\mathcal{H}$ is a home space of N with respect to $\mathcal{I}$ iff $\mathcal{B} \setminus post^*_N(\mathcal{B} \cap \mathcal{H})$ is not reachable from $\mathcal{I}$.*

Proof. ($\Rightarrow$): Assume that some marking $M \in \mathcal{B} \setminus post^*_N(\mathcal{B} \cap \mathcal{H})$ is reachable from some marking of $\mathcal{I}$. We claim that $post^*_N(\{M\}) \cap \mathcal{H} = \emptyset$, which implies that $\mathcal{H}$ is not a home space. Let $M' \in post^*_N(\{M\})$. By the definition of $\mathcal{B}$, the markings M and M' are mutually reachable, and so $M \in post^*_N(\{M'\})$. If $M' \in \mathcal{H}$, then $M' \in \mathcal{B} \cap \mathcal{H}$, and so $M \in post^*_N(\mathcal{B} \cap \mathcal{H})$, contradicting the hypothesis. So $M' \notin \mathcal{H}$, and we are done.

($\Leftarrow$): Assume $\mathcal{H}$ is not a home space. Then there exists a marking $M \in post^*_N(\mathcal{I})$ such that $post^*_N(\{M\}) \cap \mathcal{H} = \emptyset$. Let $M_0 \in \mathcal{I}$ be a marking such that $M \in post^*_N(\{M_0\})$. By hypothesis $post^*_N(\{M_0\})$ is finite, and so, since M is reachable from M_0, some marking $M' \in post^*_N(\mathcal{I}) \cap \mathcal{B}$ is reachable from M. We prove that $M' \in \mathcal{B} \setminus post^*_N(\mathcal{B} \cap \mathcal{H})$. Since $M' \in \mathcal{B}$, it suffices to prove $M' \notin post^*_N(\mathcal{B} \cap \mathcal{H})$. Assume M' is reachable from some $M'' \in \mathcal{B} \cap \mathcal{H}$, hence $M'' \neq M'$. By the definition of $\mathcal{B}$, the markings M' and M'' are mutually reachable, and so M'' is also reachable from M. But, since $M'' \in \mathcal{H}$, then some marking of $\mathcal{H}$ is reachable from M, contradicting the definition of M. ◀

▶ **Theorem 16.** *The home space problem is decidable for triples* $(N, \mathcal{I}, \mathcal{H})$ *where*
1. $\mathcal{I}$ *and* $\mathcal{H}$ *are arbitrary Presburger sets of markings, and*
2. $post^*_N(\{M_0\})$ *is finite for every* $M_0 \in \mathcal{I}$.

Proof. By Lemma 15, it suffices to decide whether $\mathcal{B} \setminus post^*_N(\mathcal{B} \cap \mathcal{H})$ is reachable from $\mathcal{I}$. We show that $\mathcal{B} \setminus post^*_N(\mathcal{B} \cap \mathcal{H})$ is an effectively Presburger set, and then apply the decidability of the reachability problem for Presburger sets of markings (i.e., given two Presburger sets $\mathcal{P}_1, \mathcal{P}_2$, decide if some marking of $\mathcal{P}_2$ is reachable from some marking of $\mathcal{P}_1$.)

By Theorem 9 and Proposition 11, the set $\mathcal{B}$ of bottom markings of N is effectively Presburger. So, since Presburger sets are effectively closed under boolean operations, it suffices to show that $post^*_N(\mathcal{B} \cap \mathcal{H})$ is effectively Presburger. Observe first that, since $\mathcal{H}$ is effectively Presburger, so is $\mathcal{B} \cap \mathcal{H}$. By the definition of $\mathcal{B}$, if $M' \in post^*_N(M)$ for some $M \in \mathcal{B} \cap \mathcal{H}$, then $M \in post^*_N(M')$. So $M \in post^*_N(\mathcal{B} \cap \mathcal{H})$ iff there is a marking $M' \in \mathcal{B} \cap \mathcal{H}$ such that M and M' are mutually reachable. Since the mutual reachability relation of N is effectively Presburger, $post^*_N(\mathcal{B} \cap \mathcal{H})$ is effectively Presburger. ◀

References

1 Benjamin Aminof, Tomer Kotek, Sasha Rubin, Francesco Spegni, and Helmut Veith. Parameterized model checking of rendezvous systems. In *CONCUR'14*, volume 8704 of *LNCS*, pages 109–124. Springer, 2014.

2 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *PODC'04*, pages 290–299. ACM, 2004.

3 Dana Angluin, James Aspnes, and David Eisenstat. Fast computation by population protocols with a leader. In *DISC'06*, volume 4167 of *LNCS*, pages 61–75. Springer, 2006.

4 Dana Angluin, James Aspnes, and David Eisenstat. Stably computable predicates are semilinear. In *PODC'06*, pages 292–299. ACM, 2006.

5 Krzysztof R. Apt and Dexter C. Kozen. Limits for automatic verification of finite-state concurrent systems. *Information Processing Letters*, 22(6):307–309, 1986.

6 J. Clement, C. Delporte-Gallet, H. Fauconnier, and M. Sighireanu. Guidelines for the verification of population protocols. In *ICDCS'11*, pages 215–224, 2011.

7 Z. Diamadi and Michael J. Fischer. A simple game for the study of trust in distributed systems. *Wuhan University Journal of Natural Sciences*, 6(1–2):72–82, 2001.

8 David Frutos-Escrig and C. Johnen. Decidability of home space property. Technical Report 503, LRI, Université de Paris-Sud. Centre d'Orsay., 1989.

9 Steven M. German and A. Prasad Sistla. Reasoning about systems with many processes. *Journal of ACM*, 39(3):675–735, 1992.

10 Seymour Ginsburg. *The Mathematical Theory of Context-Free Languages.* McGraw-Hill, Inc., New York, NY, USA, 1966.

11 Jérôme Leroux. The general vector addition system reachability problem by presburger inductive invariants. In *LICS'09*, pages 4–13. IEEE Computer Society, 2009.

12 Jérôme Leroux. Vector addition system reversible reachability problem. In *CONCUR'11*, volume 6901 of *LNCS*, pages 327–341. Springer, 2011.

13 Jérôme Leroux. Vector addition systems reachability problem (a simpler solution). In *Turing-100: The Alan Turing Centenary Conference*, volume 10 of *EPiC Series*, pages 214–228. EasyChair, 2012.

14 Jérôme Leroux. Presburger vector addition systems. In *LICS'13*, pages 23–32. IEEE Computer Society, 2013.

15 Saket Navlakha and Ziv Bar-Joseph. Distributed information processing in biological and computational systems. *Commun. ACM*, 58(1):94–102, December 2014.

Rely/Guarantee Reasoning for Asynchronous Programs

Ivan Gavran[1], Filip Niksic[1], Aditya Kanade[2], Rupak Majumdar[1], and Viktor Vafeiadis[1]

1 **Max Planck Institute for Software Systems (MPI-SWS), Germany**
2 **Indian Institute of Science, Bangalore, India**

Abstract

Asynchronous programming has become ubiquitous in smartphone and web application development, as well as in the development of server-side and system applications. Many of the uses of asynchrony can be modeled by extending programming languages with *asynchronous procedure calls* – procedures not executed immediately, but stored and selected for execution at a later point by a non-deterministic scheduler. Asynchronous calls induce a flow of control that is difficult to reason about, which in turn makes formal verification of asynchronous programs challenging. In response, we take a *rely/guarantee* approach: Each asynchronous procedure is verified separately with respect to its rely and guarantee predicates; the correctness of the whole program then follows from the natural conditions the rely/guarantee predicates have to satisfy. In this way, the verification of asynchronous programs is modularly decomposed into the more usual verification of sequential programs with synchronous calls. For the sequential program verification we use Hoare-style deductive reasoning, which we demonstrate on several simplified examples. These examples were inspired from programs written in C using the popular Libevent library; they are manually annotated and verified within the state-of-the-art Frama-C platform.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases asynchronous programs, rely/guarantee reasoning

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.483

1 Introduction

Asynchronous programming is a technique to efficiently manage multiple concurrent interactions with the environment. Application development environments for smartphone applications provide asynchronous APIs; client-side web programming with Javascript and AJAX, high-performance systems software (e.g., nginx, Chromium, Tor), as well as embedded systems, all make extensive use of asynchronous calls. By breaking long-running tasks into individual procedures and posting callbacks that are triggered when background processing completes, asynchronous programs enable resource-efficient, low-latency management of concurrent requests.

In its usual implementation, the underlying programming system exposes an *asynchronous* procedure call construct (either in the language or using a library), which allows the programmer to post a procedure for execution in the future when a certain event occurs. An event scheduler manages asynchronously posted procedures. When the corresponding event occurs, the scheduler picks the associated procedure and runs it to completion. These procedures are sequential code, possibly with recursion, and can post further asynchronous procedures.

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 483–496

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

Unfortunately, while asynchronous programs can be very efficient, the manual management of resources and asynchronous procedures can make programming in this model quite difficult. The natural control flow of a task is obscured and the programmer must ensure correct behavior for all possible orderings of external events. Specifically, the global state of the program can change between the time an asynchronous procedure is posted and the time the scheduler picks and runs it.

In recent years, a number of automatic static analyses for asynchronous programs have been proposed. The main theoretical result is the equivalence between an abstract model of asynchronous programs with Boolean variables and Petri nets, thus showing that safety and liveness verification problems are decidable for this model [14, 9, 6]. In practice, this equivalence has been the basis for several automatic tools [11, 3]. Unfortunately, existing tools still fall short of verifying "real" asynchronous programs. First, existing tools often ignore important features such as passing data as arguments to asynchronous calls or heap data structures in order to find a Boolean abstraction. Second, existing tools perform a *global* coverability analysis of the Petri net equivalent to the abstracted program. Despite the use of sophisticated heuristics, global coverability analysis tools scale poorly, especially when there are many Boolean variables [4].

In this paper, we provide a modular proof system for asynchronous programs based on rely/guarantee reasoning [10, 1, 5]. For each asynchronous procedure, we use a ("local") precondition and a postcondition, similar to modular proofs for sequential recursive programs. In addition, we use a *rely* and a *guarantee*. Intuitively, the rely is the assumption about the global state that the procedure makes about all other procedures that may happen in parallel with it. The guarantee is what the procedure ensures about the global state. In addition to predicates over global state, our rules also use predicates *posted* and *pending* that track if a task was posted asynchronously in the current call stack, or if it is pending, respectively. With these additional predicates, our modular proof rules are extremely simple:

- running each task from its precondition establishes its guarantee and postcondition;
- the rely of each task must preserve its precondition;
- if a procedure posts task h and does not cancel it, it establishes the precondition of h at the end of its execution; and finally,
- the guarantee of each task that may run between the time h is posted and h is executed establishes the rely of h.

We prove soundness of these rules, based on an invariant that ensures that if a procedure is pending, then its precondition remains valid at every schedule point.

It is possible to simulate asynchronous programs using multi-threaded programs and vice versa [12]. Thus, in principle, rely/guarantee reasoning for multi-threaded programs [10, 5, 8, 7] – extended with rules for dynamic thread creation – could be used to reason about asynchronous programs. However, by focusing on the specific concurrency model, we can deal with programming features such as recursive tasks, as well as more advanced asynchronous programming features such as deletion of tasks. To support these features, the reduction to multi-threaded programs would add additional data structures to the program, losing the structure of the program. Thus, "compiling" to threads, while theoretically possible, is not likely to preserve the local, and often simple, reason why a program is correct.

We have implemented our proof system on top of the Frama-C framework and show modular proofs of partial correctness on two asynchronous programs written in C using the Libevent library. The programs are simple but realistic examples of common asynchronous idioms. We show that one can verify these idioms by constructing "small" modular proofs, using generic rely and guarantee predicates that can be automatically derived from preconditions.

```
1  struct client_state { ... };
2
3  async main() {
4    // prepare a socket for
5    // incoming connections
6    int socket = prepare_socket();
7    post accept(socket);
8  }
9
10 //@ requires \valid(s);
11 async read(struct client_state *s) {
12   if (/* s->fd ready */) {
13     // receive a chunk and store a
14     // rot13'd version into s->buffer
15     post write(s);
16     post read(s);
17   }
18   else { // connection closed
19     delete write(s);
20     free(s);
21   }
22 }
```

```
23 async accept(int socket) {
24   if (/* socket ready */) {
25     struct client_state *s = malloc(...);
26     s->fd = accept_connection(socket);
27     // initialize s->buffer
28     post read(s);
29   }
30   post accept(socket);
31 }
32
33 //@ requires \valid(s);
34 async write(struct client_state *s) {
35   if (/* s->fd ready */) {
36     // send a chunk
37     if (/* there's more to send */)
38       post write(s);
39   }
40   else { // connection closed
41     delete read(s);
42     free(s);
43   }
44 }
```

Figure 1 Snippet of the ROT13 program. In this and the subsequent figures, parts of the code are omitted and replaced by comments for brevity.

Moreover, reasoning about asynchronous programs can be effectively reduced to modular reasoning about sequential programs, for which sophisticated verification environments already exist.

2 Main Idea

Asynchronous Programs. Figure 1 shows a version of the ROT13 server from the Libevent manual [13]. The server receives input strings from a client, and sends back the strings obfuscated using ROT13. The execution starts in the procedure `main`, which prepares a non-blocking socket for incoming connections, and passes it to the procedure `accept` via an asynchronous call. The asynchronous call, denoted by the keyword **`post`**, schedules `accept` for later execution. In general, a procedure marked with the keyword **`async`** can be *posted* with some arguments. The arguments determine an instance of the procedure; the instance is stored in a set of pending instances. After some pending instance finishes executing, a scheduler non-deterministically selects the next one and executes it completely. In case of the ROT13 server, after `main` is done, the scheduler selects the single pending instance of `accept`. `accept` checks whether a client connection is waiting to be accepted; if so, it accepts the connection and allocates memory consisting of a socket and a buffer for communication with the client. The allocated memory, addressed by the pointer `s`, is then asynchronously passed to the procedure `read`. Finally, regardless of whether the connection has been accepted or not, `accept` re-posts itself to reschedule itself for processing any upcoming connections.

While the client connection is open, the corresponding memory allocated by `accept` is handled by a reader-writer pair: the reader (`read`) receives an input string and stores an obfuscated version of it into the buffer. It then posts the writer (`write`), which sends the content of the buffer back to the client. An interesting thing happens when the client disconnects, which can happen during the execution of either the reader or the writer. When one of the procedures notices that the connection has been closed, it releases the allocated memory. However, the procedure does not know whether an instance of its counterpart is still pending; if it is, it would try to access the deallocated memory. To make sure this does not happen, before releasing the memory, the procedure deletes (keyword **`delete`**) the potentially pending instance of its counterpart.

The example shows that control structures for asynchronous programs can be complex: tasks may post other tasks for later processing, arguments can be passed on to asynchronously posted tasks, and an unbounded number of tasks can be pending at a time.

Safety Verification. We would like to verify that every memory access in this program is safe; that is, we want to verify that both `read` and `write` can safely dereference the pointer `s`. We assume this property is expressed by the predicate valid(`s`). We write valid(`s`) as a precondition for `read` and `write` in lines 10 and 33.

Let us focus only on `read`. Its precondition clearly holds at each call site: it holds at line 28 since the memory addressed by `s` has just been freshly allocated (for simplicity, we assume malloc succeeds), and it holds at line 16 assuming `read`'s precondition holds. However, between the point `read` is posted and the point it is executed, two different things might invalidate its precondition. First, the caller may still have code to execute after the call. Second, there may be pending instances of `accept`, `read`, and `write` concurrent with `read(s)` that get executed before `read(s)` and deallocate the memory addressed by `s`.

To deal with the code of `read`'s callers, also referred to as `read`'s *parents*, we introduce predicates $\text{posted}_r(s)$ and $\text{pending}_r(s)$. (We also introduce a pair of predicates for every other asynchronous procedure, namely `main`, `accept`, and `write`.) Predicate $\text{posted}_r(s)$ holds if and only if `read(`s`)` has been posted during the execution of the current asynchronous procedure (and not deleted). Predicate $\text{pending}_r(s)$ holds if and only if `read(`s`)` is in the set of pending instances. Note that if an asynchronous procedure posts and afterwards deletes `read(`s`)`, neither $\text{posted}_r(s)$ nor $\text{pending}_r(s)$ will hold. Using the introduced predicates, `read`'s parents can now express the following parent-child postcondition:

$$\forall s.\ \text{posted}_r(s) \implies \text{valid}(s)\,. \tag{PC}$$

Informally, this postcondition says that every instance of `read` that has been posted during the execution of the procedure, and that has not been deleted afterwards, has been posted with the argument s that is valid, i.e., that can be safely dereferenced.

Rely/Guarantee. To deal with the procedures whose instances are concurrent with `read`, also referred to as `read`'s *concurrent siblings*, we employ rely/guarantee reasoning. We introduce a *rely condition* for `read`:

$$\forall s.\ \text{pending}'_r(s) \wedge \text{pending}_r(s) \wedge \text{valid}'(s) \implies \text{valid}(s)\,, \tag{R}$$

where the primed versions of the predicates denote their truth at the beginning of execution of a procedure. Informally, the rely condition says that if `read(`s`)` was pending with a valid s when a concurrent sibling started executing, and `read(`s`)` is still pending at the end of that execution, then s is still valid. In other words, `read` *relies* on the assumption that its precondition is preserved by its concurrent siblings. Any of `read`'s concurrent siblings, namely `accept`, `read`, and `write`, must *guarantee* `read`'s rely condition. This is achieved by ensuring the concurrent siblings' postconditions imply the rely condition.

As shown formally in the next section, the rely/guarantee conditions ensure the following global invariant:

$$\forall s.\ \text{pending}_r(s) \implies \text{valid}(s)\,. \tag{I}$$

This invariant holds at the beginning of execution of every asynchronous procedure. Consequently, `read`'s precondition holds at the moment `read` is executed.

The benefit of the described approach is that it abstracts away reasoning about the non-deterministic scheduler and the order in which it dispatches pending instances. We only need to verify that each asynchronous procedure satisfies its postcondition. This can be achieved using a sequential verification tool (e.g., Frama-C in our case).

A natural question to ask is why we have two predicates – posted_r and pending_r – when it seems that pending_r alone should be sufficient? If the global invariant (I) is what we are after, why not just make it a precondition and a postcondition of every asynchronous procedure? While this is sufficient, in order to prove (I) as a postcondition of an asynchronous procedure, one would need to do a case split, and separately consider `read`'s instances posted during the execution of the procedure, and instances that had been pending before the procedure started executing. In the first case, the procedure *knows* why `read`'s precondition holds, while in the second case it *assumes* that `read`'s precondition holds. By having the special predicate posted_r, we can make this case split explicit: the two separate cases correspond to the parent-child condition (PC) and the rely condition (R). The asynchronous procedures assume only their original preconditions, making the overall reasoning more local.

3 Technical Details

We formalize the rely/guarantee proof rules on SLAP, a *s*imple *l*anguage with *a*synchronous *p*rocedure calls. The main result of the paper is Theorem 1, which says that in order to verify a program with asynchronous procedure calls, it suffices to modularly verify each procedure of a sequential program.

3.1 SLAP: Syntax and Semantics

Syntax. A SLAP program consists of a set of program variables Var, a set of procedure names H, a subset $AH \subseteq H$ of *asynchronous* procedures including a special procedure *main*, and a mapping Π from procedure names in H to commands from a set $Cmds$ of commands (defined below).

We distinguish between global variables, denoted by $GVar$, and local variables, denoted by $LVar$. Local variables also include parameters of procedures. We also introduce a set of *logical variables*, disjoint from the program variables, which are used for constructing quantified formulas. We write x, y, z for single variables, and $\vec{x}, \vec{y}, \vec{z}$ for vectors of variables. We usually use $x, \vec{x}$ to denote global variables, and $y, z, \vec{y}, \vec{z}$ to denote local variables. We use the same letters for logical variables; this should not cause confusion.

We use a disjoint, *primed*, copy of the set of variables Var. Primed variables are used to denote the state of the program at the beginning of execution of a procedure, while the unprimed variables denote the current state. Logical variables do not have primed counterparts, although we often abuse notation and write them primed.

Variables are used to construct *expressions*. We leave the exact syntax of expressions unspecified. We just distinguish between Boolean expressions (usually denoted by B), and all expressions (usually denoted by E).

The set of *commands*, denoted $Cmds$, is generated by the grammar:

$$\begin{aligned} C \quad ::= \quad & x := E \mid \mathbf{assume}(B) \mid \mathbf{assert}(B) \mid g(E_1, \ldots, E_k) \\ & \mid \mathbf{post}\ h(E_1, \ldots, E_k) \mid \mathbf{delete}\ h(E_1, \ldots, E_k) \mid \mathbf{enter}\ h \mid \mathbf{exit}\ h \\ & \mid C_1; C_2 \mid C_1 + C_2 \mid C^* \end{aligned}$$

The atomic commands are assignments ($x := E$), assumptions ($\mathbf{assume}(B)$), assertions ($\mathbf{assert}(B)$), and synchronous calls ($g(\ldots)$ for $g \in H \setminus AH$), as in a sequential imperative

$$
\begin{array}{ll}
\sigma', \sigma, o', o, p', p \models \text{posted}'_h(E_1, \ldots, E_k) & \text{iff } (h, \bot[y_1 \mapsto [\![E_1]\!]_{\sigma',\sigma}, \ldots, y_k \mapsto [\![E_k]\!]_{\sigma',\sigma}]) \in o' \\
\sigma', \sigma, o', o, p', p \models \text{posted}_h(E_1, \ldots, E_k) & \text{iff } (h, \bot[y_1 \mapsto [\![E_1]\!]_{\sigma',\sigma}, \ldots, y_k \mapsto [\![E_k]\!]_{\sigma',\sigma}]) \in o \\
\sigma', \sigma, o', o, p', p \models \text{pending}'_h(E_1, \ldots, E_k) & \text{iff } (h, \bot[y_1 \mapsto [\![E_1]\!]_{\sigma',\sigma}, \ldots, y_k \mapsto [\![E_k]\!]_{\sigma',\sigma}]) \in p' \\
\sigma', \sigma, o', o, p', p \models \text{pending}_h(E_1, \ldots, E_k) & \text{iff } (h, \bot[y_1 \mapsto [\![E_1]\!]_{\sigma',\sigma}, \ldots, y_k \mapsto [\![E_k]\!]_{\sigma',\sigma}]) \in p \\
\sigma', \sigma, o', o, p', p \models B & \text{iff } [\![B]\!]_{\sigma',\sigma} = \text{true}
\end{array}
$$

Figure 2 Semantics of atomic formulas.

language, together with the additional commands for asynchronous calls (**post** $h(\ldots)$ for $h \in AH$), deletions of pending instances (**delete** $h(\ldots)$), and special commands **enter** h and **exit** h marking the entrance to and exit from a procedure. Starting with the atomic commands, complex commands are built using sequential composition (;), non-deterministic choice (+), and iteration (*).

Most of the commands in the language have their expected semantics. The command **post** $h(E_1, \ldots, E_k)$ posts an asynchronous call of procedure $h \in AH$ with arguments $E_1, \ldots, E_k$ for future execution, and **delete** $h(E_1, \ldots, E_k)$ deletes the pending occurrence of the asynchronously posted procedure h with arguments $E_1, \ldots, E_k$ if it exists. The **enter** h and **exit** h commands are there for a technical reason: they mark the entry and exit of procedure h. We assume that the command $\Pi(h)$ of each procedure h starts with **enter** h, ends with **exit** h, and that those two commands do not appear anywhere in between.

Formulas are generated as first order formulas whose atomic predicates are Boolean expressions as well as the predicates posted'_h, posted_h, $\text{pending}'_h$, and pending_h, for each asynchronous procedure $h \in AH$. Intuitively, posted_h is used for reasoning about the accumulated asynchronous calls to h made during the execution of a single asynchronous procedure, and pending_h is used for reasoning about the pending asynchronous calls to h, not necessarily made during the execution of a single asynchronous procedure. Like program variables, these predicates have the corresponding primed versions, used for reasoning about the state at the beginning of execution of a procedure. For every formula F we write F' for the formula obtained by replacing all unprimed occurrences of program variables, as well as the predicates posted_h and pending_g, by their primed counterparts.

Semantics. Assuming there is a set of values *Val*, a function $\sigma\colon Var \to Val$ is called a *valuation*. We use notation $\sigma_G := \sigma|_{GVar}$ for the restriction of σ to global variables, and $\sigma_L := \sigma|_{LVar}$ for the restriction of σ to local variables. We call the restrictions *global valuation* and *local valuation*, respectively. We use notation $\sigma[x_1 \mapsto v_1, \ldots, x_k \mapsto v_k]$ to denote a valuation that differs from σ only in variables $x_1, \ldots, x_k$, which are mapped to values $v_1, \ldots, v_k$. We assume there is a special value $\bot \in Val$ denoting a non-initialized value. We also use $\bot$ to denote a constant valuation that maps every variable to $\bot$.

Given valuations σ' and σ, we denote the value of an expression E by $[\![E]\!]_{\sigma',\sigma}$. Here, σ' is used for evaluating the primed variables (the values at the beginning of execution of the current procedure), and σ is used for evaluating the unprimed variables (the current values).

Next, we define a *configuration* $\Phi = (s, \sigma_G, o, p)$ of a SLAP program, where s is a stack that keeps track of synchronous calls, σ_G is a valuation that describes the global state, o is a set of instances asynchronously posted within the current asynchronous procedure, and p is a set of pending instances. Stack s holds *stack frames* – tuples of the form $(C, \sigma', \sigma_L, o', p')$, where C is the command that needs to be executed in the current stack frame, σ' is the

$$\text{[Enter]}\ \frac{}{((\textbf{enter}\ h; C, \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p) \xrightarrow{\Pi}_s ((C, \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p)}$$

$$\text{[Exit]}\ \frac{}{((\textbf{exit}\ h, \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p) \xrightarrow{\Pi}_s (s, \sigma_G, o, p)}$$

$$\text{[Assume]}\ \frac{\sigma', \sigma, o', o, p', p \models F}{((\textbf{assume}(F); C, \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p) \xrightarrow{\Pi}_s ((C, \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p)}$$

$$\text{[Assert OK]}\ \frac{\sigma', \sigma, o', o, p', p \models F}{((\textbf{assert}(F); C, \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p) \xrightarrow{\Pi}_s ((C, \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p)}$$

$$\text{[Assert Wrong]}\ \frac{\sigma', \sigma, o', o, p', p \not\models F}{((\textbf{assert}(F); C, \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p) \xrightarrow{\Pi}_s \textbf{wrong}}$$

$$\text{[Assign]}\ \frac{\rho = \sigma[x \mapsto [\![E]\!]_{\sigma', \sigma}]}{((x := E; C, \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p) \xrightarrow{\Pi}_s ((C, \sigma', \rho_L, o', p') :: s, \rho_G, o, p)}$$

$$\text{[Choice]}\ \frac{i \in \{1, 2\}}{((C_1 + C_2; C, \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p) \xrightarrow{\Pi}_s ((C_i; C, \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p)}$$

$$\text{[Loop Skip]}\ \frac{}{((C^*; C', \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p) \xrightarrow{\Pi}_s ((C', \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p)}$$

$$\text{[Loop Step]}\ \frac{}{((C^*; C', \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p) \xrightarrow{\Pi}_s ((C; C^*; C', \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p)}$$

$$\text{[Sync Call]}\ \frac{h \in H \setminus AH \qquad \rho = \bot[y_1 \mapsto [\![E_1]\!]_{\sigma', \sigma}, \ldots, y_k \mapsto [\![E_k]\!]_{\sigma', \sigma}]}{((h(E_1, \ldots, E_k); C, \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p) \xrightarrow{\Pi}_s ((\Pi(h), \sigma_G \cup \rho_L, \rho_L, o, p) :: (C; \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p)}$$

$$\text{[Async Call]}\ \frac{h \in AH \qquad \rho = \bot[y_1 \mapsto [\![E_1]\!]_{\sigma', \sigma}, \ldots, y_k \mapsto [\![E_k]\!]_{\sigma', \sigma}] \qquad q = o \cup \{(h, \rho_L)\} \qquad r = p \cup \{(h, \rho_L)\}}{((\textbf{post}\ h(E_1, \ldots, E_k); C, \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p) \xrightarrow{\Pi}_s ((C, \sigma', \sigma_L, o', p') :: s, \sigma_G, q, r)}$$

$$\text{[Async Delete]}\ \frac{h \in AH \qquad \rho = \bot[y_1 \mapsto [\![E_1]\!]_{\sigma', \sigma}, \ldots, y_k \mapsto [\![E_k]\!]_{\sigma', \sigma}] \qquad q = o \setminus \{(h, \rho_L)\} \qquad r = p \setminus \{(h, \rho_L)\}}{((\textbf{delete}\ h(E_1, \ldots, E_k); C, \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p) \xrightarrow{\Pi}_s ((C, \sigma', \sigma_L, o', p') :: s, \sigma_G, q, r)}$$

Figure 3 Semantics of SLAP – sequential part.

valuation at the beginning of execution in the current stack frame, σ_L is the valuation that describes the current local state, and o' and p' are sets of posted and pending instances at the beginning of execution in the current stack frame. Sets o, p, o', and p' hold pairs of the form (h, σ_L), where h is the posted or pending procedure, and σ_L is a valuation that describes the values passed to h. We use notation $t :: ts$ to denote a stack consisting of a head t and a tail ts, and $\emptyset$ to denote both an empty stack and an empty set. Apart from configurations of the form (s, σ_G, o, p), which are part of the correct program execution, there is also a special configuration **wrong**.

At this point we have introduced all the concepts and terminology needed to give semantics to SLAP programs. First, the semantics of formulas is given in terms of valuations σ', σ, and sets o', o, p', p. The semantics of atomic formulas is shown in Figure 2, and the semantics of complex formulas is defined inductively. We write $\sigma', \sigma, o', o, p', p \models F$ if F holds with respect to $\sigma', \sigma, o', o, p', p$. We also write $\Phi \models F$ if $\Phi = ((C, \sigma', \sigma_L, o', p') :: s, \sigma_G, o, p)$ and $\sigma', \sigma, o', o, p', p \models F$. If $\Phi = (\emptyset, \sigma_G, o, p)$, the truth of F containing local or primed variables, or the predicates posted'_h and $\text{pending}'_h$ is undefined. Finally, **wrong** $\models F$ for any F.

Next, we define the sequential semantics of a SLAP program $\Pi\colon H \to \mathit{Cmds}$ as a transition system over configurations. The rules that define the sequential transition relation

$$\frac{\Phi \xrightarrow{\Pi}_s \Phi'}{\Phi \xrightarrow{\Pi}_a \Phi'}\ [\textsc{Extend}] \qquad \frac{h \in AH \qquad (h, \sigma_L) \in p \qquad r = p \setminus \{(h, \sigma_L)\}}{(\emptyset, \sigma_G, o, p) \xrightarrow{\Pi}_a ((\Pi(h), \sigma, \sigma_L, \emptyset, r) :: \emptyset, \sigma_G, \emptyset, r)}\ [\textsc{Dispatch}]$$

Figure 4 Semantics of SLAP – asynchronous part.

$$\forall \vec{x}', \vec{x}, \vec{y}', \vec{y}.\ P'_h(\vec{x}', \vec{y}') \wedge Q_h(\vec{x}', \vec{y}', \vec{x}, \vec{y}) \implies G_h(\vec{x}', \vec{x}) \tag{1}$$

$$\forall \vec{x}', \vec{x}, \vec{y}, \vec{z}', \vec{z}.\ \text{posted}_h(\vec{y}) \wedge P'_g(\vec{x}', \vec{z}') \wedge Q_g(\vec{x}', \vec{z}', \vec{x}, \vec{z}) \implies P_h(\vec{x}, \vec{y}), \tag{2}$$
where $g \in \text{parents}(h)$

$$\forall \vec{x}', \vec{x}, \vec{y}.\ \text{pending}'_h(\vec{y}) \wedge \text{pending}_h(\vec{y}) \wedge P'_h(\vec{x}', \vec{y}) \wedge R_h(\vec{x}', \vec{x}) \implies P_h(\vec{x}, \vec{y}) \tag{3}$$

$$\forall \vec{x}', \vec{x}.\ G_g(\vec{x}', \vec{x}) \implies R_h(\vec{x}', \vec{x}), \tag{4}$$
where $g \in \text{siblings}(h)$

Figure 5 Rely/guarantee conditions. Variables $\vec{x}', \vec{x}$ represent global variables, and variables $\vec{y}', \vec{y}, \vec{z}', \vec{z}$ represent parameters.

$\xrightarrow{\Pi}_s$ are given in Figure 3. The asynchronous semantics extends the sequential semantics by integrating the behavior of the non-deterministic scheduler. The rules that define the asynchronous transition relation $\xrightarrow{\Pi}_a$ are given in Figure 4.

Note that we are modeling the pool of pending procedure instances using a set. Therefore, posting an instance that is already pending has no effect. An alternative would be to use a multiset and count the number of pending instances. We chose the first option for three reasons. First, it corresponds to the semantics of Libevent's function `event_add()`. Second, it simplifies the semantics of deleting procedure instances. And third, one can always simulate the counting semantics by extending the procedure with an extra parameter that acts as a counter.

3.2 Rely/Guarantee Decomposition

In order to reason about an asynchronous program $\Pi\colon H \to \mathit{Cmds}$ modularly, for each of its asynchronous procedures $h \in AH$ we require a specification in terms of formulas P_h, R_h, G_h, and Q_h. Formulas P_h and Q_h are h's precondition and postcondition in the standard sense: P_h is a formula over Var that is supposed to hold at the beginning of h's execution, while Q_h is a formula over $\mathit{Var}' \cup \mathit{Var}$ that is supposed to hold at the end of h's execution. Predicates R_h and G_h are formulas over $\mathit{GVar}' \cup \mathit{GVar}$, and they represent the procedure's rely and guarantee conditions. Intuitively, R_h tells what h *relies on* about the change of the global state, while G_h tells what h *guarantees about* the change of the global state. We require that the predicates posted'_g, posted_g, $\text{pending}'_g$, and pending_g appear in the specification only in negative positions. Furthermore, in P_h we allow only the unprimed predicates, and we require $P_{main} \equiv \text{true}$.

We will say that the specification $(P_h, R_h, G_h, Q_h)_{h \in AH}$ is a *rely/guarantee decomposition* of Π if the four conditions in Figure 5 are satisfied. Condition (1) requires procedure h to establish its guarantee. Condition (2) requires that each parent of h, i.e. each asynchronous procedure that posts h, establishes h's precondition. Condition (3) is the *stability condition*: it requires the rely predicate R_h to be strong enough to preserve preconditions of procedure

h's pending instances. Finally, condition (4) requires that h's rely predicate is guaranteed by each of h's concurrent siblings, i.e. asynchronous procedures that may be executed between the point when h is posted and the point when h itself is executed. Together, conditions (1)–(4) imply the following lifecycle of an asynchronous procedure instance: once posted by its parent, its precondition is established. Before it is executed, its precondition is preserved by its concurrent siblings. When the procedure instance is finally executed, its precondition holds.

Given a rely/guarantee decomposition $(P_h, R_h, G_h, Q_h)_{h \in AH}$ of a program $\Pi\colon H \to \mathit{Cmds}$, we define a transformation of commands $\tau\colon \mathit{Cmds} \to \mathit{Cmds}$ that inserts assumptions and assertions of preconditions and postconditions at the right places:

$$\begin{aligned}
\tau(\textbf{enter } h) &:= \textbf{enter } h; \textbf{assume}(P_h), && \text{for } h \in AH,\\
\tau(\textbf{exit } h) &:= \textbf{assert}(Q_h); \textbf{exit } h, && \text{for } h \in AH,\\
\tau(C_1; C_2) &:= \tau(C_1); \tau(C_2)\\
\tau(C_1 + C_2) &:= \tau(C_1) + \tau(C_2)\\
\tau(C^*) &:= \tau(C)^*\\
\tau(C) &:= \tau(C), && \text{otherwise.}
\end{aligned}$$

The definition of τ is naturally lifted to configurations (s, σ_G, o, p) and **wrong**: in case of (s, σ_G, o, p), τ transforms all commands that await execution on the stack s, while $\tau(\textbf{wrong}) = \textbf{wrong}$.

Given a program $\Pi\colon H \to \mathit{Cmds}$, we will say that Π is *sequentially correct* with respect to a rely/guarantee decomposition $(P_h, R_h, G_h, Q_h)_{h \in AH}$ if for every valuation $\sigma\colon \mathit{Var} \to \mathit{Val}$, every set of pending instances p, and every asynchronous procedure $h \in AH$ we have

$$((\tau(\Pi(h)), \sigma, \sigma_L, \emptyset, p) :: \emptyset, \sigma_G, \emptyset, p) \xrightarrow{\tau \circ \Pi}\!\!\!\!\!\!\nearrow\,{}^{*}_{s}\ \textbf{wrong}\,.$$

We will say that Π is *correct* if we have

$$(\emptyset, \bot_G, \emptyset, \{(\mathit{main}, \bot_L)\}) \xrightarrow{\Pi}\!\!\!\!\!\!\nearrow\,{}^{*}_{a}\ \textbf{wrong}\,.$$

With these definitions, the soundness of rely/guarantee reasoning is stated in the following theorem.

▶ **Theorem 1.** *Let $\Pi\colon H \to \mathit{Cmds}$ be an asynchronous program. If Π is sequentially correct with respect to a rely/guarantee decomposition $(P_h, R_h, G_h, Q_h)_{h \in AH}$, then it is correct.*

The proof of Theorem 1 is based on four technical results.

▶ **Lemma 2.** *Let $\Pi\colon H \to \mathit{Cmds}$ be an asynchronous program, and let Φ_0, Φ be configurations of Π such that $\Phi_0 = (\emptyset, \bot_G, \emptyset, \{(\mathit{main}, \bot_L)\})$, $\Phi = (s, \sigma_G, o, p)$ and $\Phi_0 \xrightarrow{\Pi}{}^{*}_{a} \Phi$.*

1. *For every stack frame $(C, \sigma', \sigma_L, o', p') \in s$, $p \subseteq o \cup p'$.*
2. *If s is non-empty, then for every $h \in AH$,*

$$\Phi \models \forall \vec{y}.\ \mathrm{pending}_h(\vec{y}) \implies (\mathrm{posted}_h(\vec{y}) \vee \mathrm{pending}'_h(\vec{y}))\,.$$

Proof. The first statement is proved by induction on the length of the trace. A straightforward check shows that every rule preserves the invariant $p \subseteq o \cup p'$. The second statement is a direct corollary of the first statement. ◀

The next lemma states that preconditions of pending instances hold at each dispatch point. Thus, it formalizes the discussion in Section 2, where the invariant (I) is found to hold at each dispatch point.

▶ **Lemma 3.** *Let* $\Pi\colon H \to \mathit{Cmds}$ *be an asynchronous program with a rely/guarantee decomposition* $(P_h, R_h, G_h, Q_h)_{h\in AH}$*. If* $(\emptyset, \bot_G, \emptyset, \{(\mathit{main}, \bot_L)\}) \xrightarrow{\tau\circ\Pi}{}^*_a (\emptyset, \sigma_G, o, p)$*, then for every* $g \in AH$ *we have* $\sigma_G, \sigma_G, \emptyset, o, p, p \models \forall \vec{y}.\ \mathrm{pending}_g(\vec{y}) \implies P_g(\vec{x}, \vec{y})$*.*

Proof. By induction on the number of applications of the rule [Dispatch], using the rely/guarantee conditions (1)–(4) and the invariant from Lemma 2(2). ◀

▶ **Corollary 4.** *Let* $\Pi\colon H \to \mathit{Cmds}$ *be an asynchronous program with a rely/guarantee decomposition* $(P_h, R_h, G_h, Q_h)_{h\in AH}$*, and let* $(\emptyset, \bot_G, \emptyset, \{(\mathit{main}, \bot_L)\}) \xrightarrow{\tau\circ\Pi}{}^+_a \tau(\Phi)$*, with the last step being a dispatch of procedure* $h \in AH$*. Then,* $\tau(\Phi) \models P_h(\vec{x}, \vec{y})$*.*

Proof. From Lemma 3, we know that the state just before the dispatch satisfies $P_h(\vec{x}, \vec{y})$ because h is pending in that state. Our conclusion, therefore, follows because $P_h(\vec{x}, \vec{y})$ can contain predicates posted_g and $\mathrm{pending}_g$ only in negative positions, and the rule [Dispatch] makes the sets of posted and pending instances smaller. ◀

▶ **Lemma 5.** *Let* $\Pi\colon H \to \mathit{Cmds}$ *be an asynchronous program with a rely/guarantee decomposition* $(P_h, R_h, G_h, Q_h)_{h\in AH}$*, and let* $\Phi_0, \Phi'_0, \ldots, \Phi_k, \Phi'_k$ *be configurations of* Π *such that* $\Phi_0 = (\emptyset, \bot_G, \emptyset, \{(\mathit{main}, \bot_L)\})$ *and*

$$\Phi_0 \xrightarrow{\Pi}{}^*_s \Phi'_0 \xrightarrow{\Pi}_a \Phi_1 \xrightarrow{\Pi}{}^*_s \Phi'_1 \xrightarrow{\Pi}_a \cdots \xrightarrow{\Pi}_a \Phi_k \xrightarrow{\Pi}{}^*_s \Phi'_k\,,$$

with all of the asynchronous steps being taken according to the rule [Dispatch]*. Either:*

1. $\forall i \in 0..k.\ \tau(\Phi_i) \xrightarrow{\tau\circ\Pi}{}^*_s \tau(\Phi'_i)$*, or*
2. $\exists i \in 0..k.\ \tau(\Phi_i) \xrightarrow{\tau\circ\Pi}{}^*_s$ **wrong**

Proof. By induction on the length of the trace. A straightforward inspection of the rules in Figure 3 shows that each step of the original trace can be simulated by one or two steps of the transformed trace. The only non-trivial point is showing that the inserted assume statements always hold, which follows from Corollary 4. ◀

Proof of Theorem 1. By contraposition and application of Lemma 5. ◀

Notice that the rely/guarantee decomposition uses two relations: parents and siblings. Formally, $g \in \mathrm{parents}(h)$ if there is a reachable configuration obtained by executing **exit** g in which h is in the set of posted instances. Similarly, $g \in \mathrm{siblings}(h)$ if there is a reachable configuration in which both g and h are in the set of pending instances. While these relations are hard to compute precisely, Theorem 1 holds when we use any over-approximation of these relations. A trivial over-approximation of both relations is the set AH of all asynchronous procedures. In Section 4, we discuss a better approximation obtained through simple static analysis.

4 Rely/Guarantee in Practice

4.1 Implementation for Libevent

We focused on C programs that use the Libevent library[1]. Libevent is an event notification library whose main purpose is to unify OS-specific mechanisms for handling events that occur on file descriptors. From this it also extends to handling signals and timeout events. The

[1] `http://libevent.org/`

$$R_h(\vec{x}', \vec{x}) \equiv \forall \vec{y}.\ \text{pending}'_h(\vec{y}) \wedge \text{pending}_h(\vec{y}) \wedge P'_h(\vec{x}', \vec{y}) \implies P_h(\vec{x}, \vec{y})$$

$$G_h(\vec{x}', \vec{x}) \equiv \bigwedge_{g \in \text{siblings}(h)} R_g(\vec{x}', \vec{x})$$

$$Q_h(\vec{x}', \vec{z}', \vec{x}, \vec{z}) \equiv G_h(\vec{x}', \vec{x}) \wedge \bigwedge_{g \in \text{children}(h)} \forall \vec{y}.\ \text{posted}_g(\vec{y}) \implies P_g(\vec{x}, \vec{y})$$

Figure 6 Generic rely/guarantee predicates.

library is used in asynchronous applications such as the *Chromium* web browser, *Memchached*, *SNTP*, and *Tor*.

We abstract away the details of the events by assuming their handlers are dispatched non-deterministically, instead of when the events actually occur. Thus, registering an event handler for a specific event corresponds to calling the handler asynchronously in our model.

Even with this abstraction, Libevent remains too complex for reasoning about directly. Therefore, we hide it behind a much simpler interface that corresponds to SLAP: For each asynchronous procedure h, we provide two (synchronous) functions called `post_`h and `delete_`h, with the same parameters as h. As the prefixes suggest, these functions are used for posting and deleting h's instances. With these functions, the C code we are analyzing directly resembles the code of the ROT13 server in Figure 1; the difference is that instead of the keywords **`post`** and **`delete`**, we use the functions with the corresponding prefixes.

We implemented the rely/guarantee rules on top of the Frama-C verification platform [2]. We use ACSL, Frama-C's specification language, which is expressive enough to encode the predicates posted and pending, with their state being maintained using ghost code. The specification is a fairly straightforward encoding of the semantics of SLAP. After the transformation that inserts appropriate preconditions and postconditions (along with the necessary ghost code), we use Frama-C's WP (*weakest-precondition*) plugin to generate verification conditions that are discharged by Z3.

In order to over-estimate the sets of parents and concurrent siblings, we manually perform a simple static analysis. In this analysis, we ignore deletes, and only look at posts. For each procedure h we perform a 0–1–ω abstraction of the number of asynchronous procedures' instances posted at each location. Specifically, at h's exit point this gives us an abstracted number of instances of each procedure posted by h. From this information, we can directly construct sets of children, or equivalently parents. Furthermore, if h posts f and g, then $f \in \text{siblings}(g)$ and $g \in \text{siblings}(f)$. Also, if h posts more than one instance of g, then $g \in \text{siblings}(g)$. We use these two facts to bootstrap the following recursion that computes siblings: if $f \in \text{siblings}(g)$, then $f' \in \text{siblings}(g)$ and $f \in \text{siblings}(g')$ for every $f' \in \text{children}(f)$ and $g' \in \text{children}(g)$.

An archive with the verified programs can be found at the URL: `http://www.mpi-sws.org/~fniksic/concur2015/rely-guarantee.tar.gz`.

4.2 Generic Rely/Guarantee Predicates

Instead of asking the programmer to manually specify rely/guarantee predicates and post-conditions, and then checking that they satisfy the rely/guarantee conditions (1)–(4), we can use generic predicates shown in Figure 6. These predicates trivially satisfy conditions (1)–(4); in fact, they are the weakest predicates to do so.

Note that the generic predicates, while convenient, might not be sufficient for verifying correctness of all programs. The reason is that the proof obligations for the sequential

```
1  struct device {
2    int owner;
3    // ...
4  } dev;
5
6  async main() {
7    dev.owner = 0;
8    int socket = prepare_socket();
9    post listen(socket);
10 }
11
12 /*@ requires id > 0;
13   @ requires global_invariant_write;
14   @*/
15 async new_client(int id, int fd) {
16   if (dev.owner > 0)
17     post new_client(id, fd);
18   else {
19     dev.owner = id;
20     post write(id, fd);
21   }
22 }
```

```
23 async listen(int socket) {
24   if (/* socket ready */) {
25     int id = new_client_id();
26     int fd = accept_connection(socket);
27     post new_client(id, fd);
28   }
29   post listen(socket);
30 }
31
32 /*@ requires
33   @   id > 0 ∧
34   @   dev.owner = id ∧
35   @   ∀ int id1, int fd1;
36   @     pending_write(id1, fd1)
37   @       ⇒ id = id1 ∧ fd = fd1
38   @*/
39 async write(int id, int fd) {
40   if (transfer(fd, dev))
41     post write(id, fd);
42   else // write complete
43     dev.owner = 0;
44 }
```

Figure 7 Snippet of the Race program.

programs obtained by applying the transformation τ may not be provable. The generic predicates are sufficient for the ROT13 server from Section 2, where the preconditions are: $P_{\texttt{main}} \equiv P_{\texttt{accept}} \equiv \text{true}$, and $P_{\texttt{read}} \equiv P_{\texttt{write}} \equiv \text{valid}(\texttt{s})$. Plugging these preconditions into the generic predicates from Figure 6 gives rise to proof obligations that can be discharged by Z3. However, the generic predicates are not sufficient for the program we discuss next.

Consider the Race program [9] in Figure 7. Initially, procedure `main` sets up a global resource called `dev` to which multiple clients will transfer data by setting the `dev.owner` flag to zero. `main` then prepares a socket and posts the procedure `listen` to listen to client connections. `listen` checks whether the socket is ready; if so, it accepts the connection to get a file descriptor `fd`, and generates a unique positive `id` for the client. It then passes `id` and `fd` to the procedure `new_client`. `new_client` checks whether the device is currently owned by some client (`dev.owner>0`); if so, it re-posts itself. If the device is free (`dev.owner==0`), `new_client` takes the ownership for the client identified by `id` and posts the procedure `write` that performs the transfer of the client's data. `write` operates in multiple steps, re-posting itself until the transfer is done. At the end, it releases the device by setting `dev.owner` back to zero.

In this example, since multiple clients are trying to non-atomically write to a single shared resource, the important property is mutual exclusion: there should always be at most one pending instance `write(id, fd)`, and if there is one, `dev.owner` should be set to `id`. We encode this property as a precondition $P_{\texttt{write}}$ to `write` in lines 32–37.

To ensure mutual exclusion, procedure `new_client` assumes `id>0` as its precondition in line 12. However, this precondition is not sufficiently strong for `new_client` to establish `write`'s generic rely predicate. This is due to `write`'s precondition including assumptions about other of `write`'s pending instances. However, `new_client` can establish `write`'s rely if it additionally assumes `write`'s global invariant $\forall id, fd.\ \text{pending}_{\texttt{write}}(id, fd) \implies P_{\texttt{write}}(id, fd)$ (line 13).

In order to justify `new_client`'s additional assumption, and show that there is no hidden circular reasoning, we show that we can weaken the rely/guarantee conditions (1) and (2). Indeed, Lemma 3 still holds if we replace conditions (1) and (2) with the following weaker

versions:

$$\forall \vec{x}', \vec{x}, \vec{y}', \vec{y}. \bigwedge_{f \in AH} \left(\forall \vec{z}.\ \text{pending}'_f(\vec{z}) \implies P'_f(\vec{x}', \vec{z})\right) \tag{1'}$$
$$\wedge\, P'_h(\vec{x}', \vec{y}') \wedge Q_h(\vec{x}', \vec{y}', \vec{x}, \vec{y}) \implies G_h(\vec{x}', \vec{x})$$

$$\forall \vec{x}', \vec{x}, \vec{y}, \vec{z}', \vec{z}. \bigwedge_{f \in AH} \left(\forall \vec{u}.\ \text{pending}'_f(\vec{u}) \implies P'_f(\vec{x}', \vec{u})\right) \tag{2'}$$
$$\wedge\, \text{posted}_h(\vec{y}) \wedge P'_g(\vec{x}', \vec{z}') \wedge Q_g(\vec{x}', \vec{z}', \vec{x}, \vec{z}) \implies P_h(\vec{x}, \vec{y})\,,$$

where $g \in \text{parents}(h)$

The generic postconditions Q_h can now be weakened as follows.

$$Q_h(\vec{x}', \vec{z}', \vec{x}, \vec{z}) \equiv \bigwedge_{f \in AH} \left(\forall \vec{y}.\ \text{pending}'_f(\vec{y}) \implies P'_f(\vec{x}', \vec{y})\right)$$
$$\implies \left(G_h(\vec{x}', \vec{x}) \wedge \bigwedge_{g \in \text{children}(h)} \forall \vec{y}.\ \text{posted}_g(\vec{y}) \implies P_g(\vec{x}, \vec{y})\right)$$

This allows asynchronous procedures to freely assume any of the global invariants ensured by Lemma 3 if it helps them to establish their guarantees. Even though at first glance such additional assumptions might seem vacuous, the Race example shows this is not the case.

4.3 Limitations

In practice, Frama-C and the WP plugin have limitations that are orthogonal to the rely/guarantee approach. One limitation is WP's lack of support for dynamic memory allocation. In fact, in order to verify the ROT13 server, we were not able to use Frama-C's built-in predicate `\valid`. Instead, we had to specify our own validity predicate and use corresponding dedicated malloc and free functions. Generalizing such an approach to more complicated programs is infeasible, as our custom memory model does not integrate well with the built-in one. A related limitation is restricted reasoning about inductive data structures. While Frama-C's specification language ACSL supports inductive predicates, the WP plugin does not fully support them. Moreover, reasoning about even the simplest inductive data structures such as linked lists may require separation predicates that are beyond the expressive power of ACSL. Our rely/guarantee rules work "modulo" a sequential verifier, so better handling of these limitations will allow reasoning about more complex asynchronous programs.

Acknowledgements. This research was sponsored in part by the EC FP7 FET project ADVENT (308830) and an ERC Synergy Award ("ImPACT").

References

1 M. Abadi and L. Lamport. Conjoining specifications. *ACM Transactions on Programming Languages and Systems*, 17(3):507–534, 1995.

2 P. Cuoq, F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, and B. Yakobowski. Frama-C – A software analysis perspective. In *SEFM 2012*, pages 233–247, 2012.

3 E. D'Osualdo, J. Kochems, and C.-H. L. Ong. Automatic verification of Erlang-style concurrency. In *Proceedings of the 20th Static Analysis Symposium*, SAS'13. Springer-Verlag, 2013.

4 J. Esparza, R. Ledesma-Garza, R. Majumdar, P. Meyer, and F. Niksic. An SMT-based approach to coverability analysis. In *CAV 2014*, volume 8559 of *Lecture Notes in Computer Science*, pages 603–619. Springer, 2014.

5 C. Flanagan, S.N. Freund, and S. Qadeer. Thread-modular verification for shared-memory programs. In *ESOP 2002*, pages 262–277, 2002.

6 P. Ganty and R. Majumdar. Algorithmic verification of asynchronous programs. *ACM Trans. Program. Lang. Syst.*, 34(1):6, 2012.

7 A. Gupta, C. Popeea, and A. Rybalchenko. Predicate abstraction and refinement for verifying multi-threaded programs. In *POPL 11*, pages 331–344. ACM, 2011.

8 T.A. Henzinger, R. Jhala, and R. Majumdar. Race checking by context inference. In *PLDI 2004*, pages 1–13. ACM, 2004.

9 R. Jhala and R. Majumdar. Interprocedural analysis of asynchronous programs. In *POPL 2007*, pages 339–350. ACM, 2007.

10 C.B. Jones. Tentative steps toward a development method for interfering programs. *ACM Transactions on Programming Languages and Systems*, 5(4):596–619, 1983.

11 Alexander Kaiser, Daniel Kroening, and Thomas Wahl. Efficient coverability analysis by proof minimization. In *Proceedings of the 23rd International Conference on Concurrency Theory*, CONCUR'12, pages 500–515. Springer-Verlag, 2012.

12 H.C. Lauer and R.M. Needham. On the duality of operating system structures. *SIGOPS Oper. Syst. Rev.*, 13(2):3–19, 1979.

13 N. Mathewson. Fast portable non-blocking network programming with Libevent. `http://www.wangafu.net/~nickm/libevent-book/`. Accessed: 2015-03-12.

14 K. Sen and M. Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *CAV'06*, volume 4144 of *LNCS*, pages 300–314. Springer, 2006.

Partial Order Reduction for Security Protocols*

David Baelde, Stéphanie Delaune, and Lucca Hirschi

LSV, ENS Cachan & CNRS, France
{baelde,delaune,hirschi}@lsv.ens-cachan.fr

Abstract

Security protocols are concurrent processes that communicate using cryptography with the aim of achieving various security properties. Recent work on their formal verification has brought procedures and tools for deciding trace equivalence properties (*e.g.,* anonymity, unlinkability, vote secrecy) for a bounded number of sessions. However, these procedures are based on a naive symbolic exploration of all traces of the considered processes which, unsurprisingly, greatly limits the scalability and practical impact of the verification tools.

In this paper, we mitigate this difficulty by developing partial order reduction techniques for the verification of security protocols. We provide reduced transition systems that optimally eliminate redundant traces, and which are adequate for model-checking trace equivalence properties of protocols by means of symbolic execution. We have implemented our reductions in the tool Apte, and demonstrated that it achieves the expected speedup on various protocols.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases cryptographic protocols, verification, process algebra, trace equivalence

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2015.497

1 Introduction

Security protocols are concurrent processes that use various cryptographic primitives in order to achieve security properties such as secrecy, authentication, anonymity, unlinkability, *etc.* They involve a high level of concurrency and are difficult to analyse by hand. Actually, many protocols have been shown to be flawed several years after their publication (and deployment). This has led to a flurry of research on formal verification of protocols.

A successful way of representing protocols is to use variants of the π-calculus, whose labelled transition systems naturally express how a protocol may interact with a (potentially malicious) environment whose knowledge increases as more messages are exchanged over the network. Some security properties (*e.g.,* secrecy, authentication) are then described as reachability properties, while others (*e.g.,* unlinkability, anonymity) are expressed as trace equivalence properties. In order to decide such properties, a reasonable assumption is to bound the number of protocol sessions, thereby limiting the length of execution traces. Even under this assumption, infinitely many traces remain, since each input may be fed infinitely many different messages. However, symbolic execution and dedicated constraint solving procedures have been devised to provide decision procedures for reachability [18, 12] and, more recently, equivalence properties [23, 9]. Unfortunately, the resulting tools, especially those for checking equivalence (*e.g.,* Apte [8], Spec [22]), have a very limited practical impact because they scale very badly. This is not surprising since they treat concurrency in a very naive way, exploring all possible symbolic interleavings of concurrent actions.

* This work has been partially supported by the project JCJC VIP ANR-11-JS02-006, and the Inria large scale initiative CAPPRIS.

26th International Conference on Concurrency Theory (CONCUR 2015).
Editors: Luca Aceto and David de Frutos Escrig; pp. 497–510

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

Contributions. We develop partial order reduction (POR) techniques for trace equivalence checking of security protocols. Our main challenge is to do it in a way that is compatible with symbolic execution: we should provide a reduction that is effective when messages remain unknown, but leverages information about messages when it is inferred by the constraint solver. We achieve this by refining interleaving semantics in two steps, gradually eliminating redundant traces. The first refinement, called *compression*, uses the notion of polarity [2] to impose a simple strategy on traces. It does not rely on data analysis at all and can easily be used as a replacement for the usual semantics in verification algorithms. The second one, called *reduction*, takes data into account and achieves optimality in eliminating redundant traces. In practice, the reduction step can be implemented in an approximated fashion, through an extension of constraint resolution procedures. We have done so in the tool Apte, showing that our theoretical results do translate to significant practical optimisations.

Related work. The theory of partial order reduction is well developed in the context of reactive systems verification (*e.g.,* [21, 6, 16]). However, as pointed out by E. Clarke *et al.* in [11], POR techniques from traditional model-checking cannot be directly applied in the context of security protocol verification. Indeed, the application to security requires one to keep track of the knowledge of the attacker, and to refer to this knowledge in a meaningful way (in particular to know which messages can be forged at some point to feed some input). Furthermore, security protocol analysis does not rely on the internal reduction of a protocol, but has to consider arbitrary execution contexts (representing interactions with arbitrary, active attackers). Thus, any input may depend on any output, since the attacker has the liberty of constructing arbitrary messages from past outputs. This results in a dependency relation which is *a priori* very large, rendering traditional POR arguments suboptimal, and calling for domain-specific techniques.

In order to achieve our goal of improving existing tools, our techniques are designed to integrate nicely with symbolic execution. This is necessary to precisely deal with infinite, structured data, without considering an *a priori* fixed and finite set of messages, as is the case in several earlier works, *e.g.,* [11, 13]. In this task, we get some inspiration from Mödersheim *et al.* [20]. While their reduction is very limited, it brings some key insight on how POR may be combined with symbolic execution in the context of security protocols verification. All of the papers mentioned above only consider reachability properties, while we develop an approach which is adequate for model-checking trace equivalence properties. In earlier work [4] we have combined the idea of [20] with more powerful partial order reduction, in a way that is compatible with trace equivalence checking. This settled the general ideas behind the present paper, but only covered the very restrictive class of *simple processes* (parallel processes communicate on distinct channels, and replication and nested parallel composition are not allowed). Actually, we made heavy use of specific properties of those simple processes to define our reductions and prove them correct. The present work also brings a solid implementation in the tool Apte [8].

Outline. We consider in Section 2 a rich process algebra for representing security protocols. It supports arbitrary cryptographic primitives, and even includes a replication operator suitable for modelling unbounded numbers of sessions. Thus, we are not restricted to a particular fragment for which a decision procedure exists, but show the full scope of our theoretical results. We give in Section 3 an *annotated* semantics that will facilitate the following technical developments. We then define our *compressed* semantics in Section 4 and the *reduced* semantics in Section 5. In both sections, we first restrict the transition

system, then show that the restriction is adequate for checking trace equivalence under some action-determinism condition. We finally discuss how these results can be lifted to the symbolic setting in Section 6. Specifically, we describe how we have implemented our techniques in Apte, and we present experimental results showing that the optimisations are fully effective in practice.

Due to lack of space, the reader is referred to the companion technical report [5] for the missing proofs and additional details. In particular, a comparison of this work with the extensive literature about POR can be found in [5, §7].

2 Model for security protocols

In this section we introduce our process algebra, which is a variant of the applied π-calculus [1] that has been designed with the aim of modelling cryptographic protocols. Processes can exchange complex messages, represented by terms quotiented by some equational theory.

One of the key difficulties in the applied π-calculus is that it models the knowledge of the environment, seen as an attacker who listens to network communication and may also inject messages. One has to make a distinction between the content of a message (sent by the environment) and the way the message has been created (from knowledge available to the environment). While the distinction between messages and recipes came from security applications, it may be of much broader interest, as it gives a precise, intentional content to labelled transitions that we exploit to analyse data dependencies.

We study a process algebra that may seem quite restrictive: we forbid internal communication and private channels. However, this is reasonable when studying security protocols faced with the usual omnipotent attacker. In such a setting, we end up considering the worst-case scenario where any communication has to be made via the environment.

2.1 Syntax

We assume a number of disjoint and infinite sets: a set $\mathcal{C}$ of *channels*, whose elements are denoted by a, b, c; a set $\mathcal{N}$ of *private names* or *nonces*, denoted by n or k; a set $\mathcal{X}$ of *variables*, denoted by x, y, z as usual; and a set $\mathcal{W}$ of *handles*, denoted by w and used for referring to previously output terms. Next, we consider a signature Σ consisting of a finite set of function symbols together with their arity. Terms over S, written $\mathcal{T}(S)$, are inductively generated from S and function symbols from Σ. When $S \subseteq \mathcal{N}$, elements of $\mathcal{T}(S)$ are called *messages*. When $S \subseteq \mathcal{W}$, they are called *recipes* and written M, N. Intuitively, recipes express how a message has been derived by the environment from the messages obtained so far. Finally, we consider an equational theory E over terms to assign a meaning to function symbols in Σ.

▶ **Example 1.** Let $\Sigma = \{\mathsf{enc}/2, \mathsf{dec}/2, \mathsf{h}/1\}$ and E be the equational theory induced by the equation $\mathsf{dec}(\mathsf{enc}(x, y), y) = x$. Intuitively, the symbols enc and dec represent symmetric encryption and decryption, whereas h is used to model a hash function. Now, assume that the environment knows the key k as well as the ciphertext $\mathsf{enc}(n, k)$, and that these two messages are referred to by handles w and w'. The environment may decrypt the ciphertext with the key k, apply the hash function, and encrypt the result using k to get the message $m_0 = \mathsf{enc}(\mathsf{h}(n), k)$. This computation is modelled using the *recipe* $M_0 = \mathsf{enc}(\mathsf{h}(\mathsf{dec}(w', w)), w)$.

▶ **Definition 2.** Processes are defined by the following syntax where $c, a \in \mathcal{C}$, $x \in \mathcal{X}$, $u, v \in \mathcal{T}(\mathcal{N} \cup \mathcal{X})$, and $\vec{c}$ (resp. $\vec{n}$) is a sequence of channels from $\mathcal{C}$ (resp. names from $\mathcal{N}$).

$$P, Q \ ::= \ 0 \ \mid \ (P \mid Q) \ \mid \ \mathtt{in}(c, x).P \ \mid \ \mathtt{out}(c, u).P \ \mid \ \mathtt{if}\ u = v\ \mathtt{then}\ P\ \mathtt{else}\ Q \ \mid \ !^{a}_{\vec{c},\vec{n}} P$$

The last construct combines replication with channel and name restriction: $!^a_{\vec{c},\vec{n}}P$ may be read as $!(\nu\vec{c}.\mathtt{out}(a,\vec{c}).\nu\vec{n}.P)$ in standard applied π-calculus. Our goal with this compound construct is to support replication in a way that is not fundamentally incompatible with the action-determinism condition which we eventually impose on our processes. This is achieved here by advertising on the public channel a any new copy of the replicated process. At the same time, we make public the new channels $\vec{c}$ on which the copy may operate – but not the new names $\vec{n}$. While it may seem restrictive, this style is actually natural for security protocols where the attacker knows exactly to whom he is sending a message and from whom he is receiving, *e.g.,* via IP addresses.

We shall only consider *ground* processes, where each variable is bound by an input. We denote by $\mathsf{fc}(P)$ and $\mathsf{bc}(P)$ the set of *free* and *bound channels* of P.

▶ **Example 3.** The process P_0 models an agent who sends the ciphertext $\mathtt{enc}(n,k)$, and then waits for an input on c. In case the input has the expected form, the constant $\mathtt{ok}$ is emitted.

$$P_0 = \mathtt{out}(c,\mathtt{enc}(n,k)).\mathtt{in}(c,x).\mathtt{if}\ \mathtt{dec}(x,k) = \mathsf{h}(n)\ \mathtt{then}\ \mathtt{out}(c,\mathtt{ok}).0\ \mathtt{else}\ 0$$

The processes P_0 as well as $!^a_{c,n}P_0$ are ground. We have that $\mathsf{fc}(P_0) = \{c\}$ and $\mathsf{bc}(P_0) = \emptyset$ whereas $\mathsf{fc}(!^a_{c,n}P_0) = \{a\}$ and $\mathsf{bc}(!^a_{c,n}P_0) = \{c\}$.

2.2 Semantics

We only consider processes that are *normal* w.r.t. *internal reduction* $\rightsquigarrow$ defined as follows:

$$\left.\begin{array}{l}\mathtt{if}\ u = v\ \mathtt{then}\ P\ \mathtt{else}\ Q \rightsquigarrow P\ \textit{when}\ u =_{\mathsf{E}} v \qquad P \mid Q \rightsquigarrow P' \mid Q \\ \mathtt{if}\ u = v\ \mathtt{then}\ P\ \mathtt{else}\ Q \rightsquigarrow Q\ \textit{when}\ u \neq_{\mathsf{E}} v \qquad Q \mid P \rightsquigarrow Q \mid P'\end{array}\right\}\ \textit{when}\ P \rightsquigarrow P'$$

$$(P_1 \mid P_2) \mid P_3 \rightsquigarrow P_1 \mid (P_2 \mid P_3) \qquad P \mid 0 \rightsquigarrow P \qquad 0 \mid P \rightsquigarrow P$$

Any process in normal form built from parallel composition can be uniquely written as $P_1 \mid (P_2 \mid (\ldots \mid P_n))$ with $n \geq 2$, which we denote $\Pi_{i=1}^n P_i$, where each process P_i is neither a parallel composition nor the process 0.

We now define our labelled transition system. It deals with *configurations* (denoted by A, B) which are pairs $(\mathcal{P};\Phi)$ where $\mathcal{P}$ is a multiset of ground processes and Φ, called the *frame*, is a substitution mapping handles to messages that have been made available to the environment.

Given a configuration A, $\Phi(A)$ denotes its second component. Given a frame Φ, $\mathrm{dom}(\Phi)$ denotes its domain.

In	$(\{\mathtt{in}(c,x).Q\} \uplus \mathcal{P};\Phi) \xrightarrow{\mathtt{in}(c,M)} (\{Q\{M\Phi/x\}\} \uplus \mathcal{P};\Phi)$	$M \in \mathcal{T}(\mathrm{dom}(\Phi))$
Out	$(\{\mathtt{out}(c,u).Q\} \uplus \mathcal{P};\Phi) \xrightarrow{\mathtt{out}(c,w)} (\{Q\} \uplus \mathcal{P};\Phi \cup \{w \mapsto u\})$	$w \in \mathcal{W}$ fresh
Repl	$(\{!^a_{\vec{c},\vec{n}}P\} \uplus \mathcal{P};\Phi) \xrightarrow{\mathtt{sess}(a,\vec{c})} (\{P;!^a_{\vec{c},\vec{n}}P\} \uplus \mathcal{P};\Phi)$	$\vec{c},\vec{n}$ fresh
Par	$(\{\Pi_{i=1}^n P_i\} \uplus \mathcal{P};\Phi) \xrightarrow{\tau} (\{P_1,\ldots,P_n\} \uplus \mathcal{P};\Phi)$	
Zero	$(\{0\} \uplus \mathcal{P};\Phi) \xrightarrow{\tau} (\mathcal{P};\Phi)$	

Rule In expresses that an input process may receive any message that the environment can derive from the current frame. In rule Out, the frame is enriched with a new message. The last two rules simply translate the parallel structure of processes into the multiset structure of the configuration. As explained above, rule Repl combines the replication of a process together with the creation of new channels and nonces. The channels $\vec{c}$ are implicitly made public, but the newly created names $\vec{n}$ remain private. Remark that channels $\vec{c}$ and names $\vec{n}$ must be fresh, *i.e.,* they do not appear free in the original configuration. As

usual, freshness conditions do not block executions: it is always possible to rename bound channels $\vec{c}$ and names $\vec{n}$ of a process $!^a_{\vec{c},\vec{n}}P$ before applying Repl. We denote by $\mathsf{bc}(\mathsf{tr})$ the bound channels of a trace tr, *i.e.,* all the channels that occur in second argument of an action $\mathtt{sess}(a, \vec{c})$ in tr, and we consider traces where channels are bound at most once.

▶ **Example 4.** Going back to Example 3 with $\Phi_0 = \{w_1 \mapsto k\}$, we have that:

$$(\{!^a_{c,n}P_0\};\Phi_0) \xrightarrow{\mathtt{sess}(a,c)} \xrightarrow{\mathtt{out}(c,w_2)} \xrightarrow{\mathtt{in}(c,M_0)} (\{\mathtt{out}(c,\mathsf{ok}).0;!^a_{c,n}P_0\};\Phi)$$

where $\Phi = \{w_1 \mapsto k, w_2 \mapsto \mathsf{enc}(n,k)\}$ and $M_0 = \mathsf{enc}(\mathsf{h}(\mathsf{dec}(w_2,w_1)),w_1)$.

2.3 Equivalences

We are concerned with trace equivalence, which is used [7, 15] to model anonymity, untraceability, strong secrecy, etc. Finer behavioural equivalences, *e.g.,* weak bisimulation, appear to be too strong with respect to what an attacker can really observe. Intuitively, two configurations are trace equivalent if the attacker cannot tell whether he is interacting with one or the other. To make this formal, we introduce a notion of equivalence between frames.

▶ **Definition 5.** Two frames Φ and Φ' are in *static equivalence*, written $\Phi \sim \Phi'$, when $\mathrm{dom}(\Phi) = \mathrm{dom}(\Phi')$, and: $M\Phi =_{\mathsf{E}} N\Phi \Leftrightarrow M\Phi' =_{\mathsf{E}} N\Phi'$ for any terms $M, N \in \mathcal{T}(\mathrm{dom}(\Phi))$.

▶ **Example 6.** Continuing Example 4, consider $\Phi' = \{w_1 \mapsto k', w_2 \mapsto \mathsf{enc}(n,k)\}$. The test $\mathsf{enc}(\mathsf{dec}(w_2,w_1),w_1) = w_2$ is true in Φ but not in Φ', thus $\Phi \not\sim \Phi'$.

We then define $\mathsf{obs}(\mathsf{tr})$ to be the subsequence of tr obtained by erasing τ actions.

▶ **Definition 7.** Let A and B be two configurations. We say that $A \sqsubseteq B$ when, for any $A \xrightarrow{\mathsf{tr}} A'$ such that $\mathsf{bc}(\mathsf{tr}) \cap \mathsf{fc}(B) = \emptyset$, there exists $B \xrightarrow{\mathsf{tr}'} B'$ such that $\mathsf{obs}(\mathsf{tr}) = \mathsf{obs}(\mathsf{tr}')$ and $\Phi(A') \sim \Phi(B')$. They are *trace equivalent*, written $A \approx B$, when $A \sqsubseteq B$ and $B \sqsubseteq A$.

In order to lift our optimised semantics to trace equivalence, we will require configurations to be *action-deterministic*. This common assumption in POR techniques [6] is also reasonable in the context of security protocols, where the attacker knows with whom he is communicating.

▶ **Definition 8.** A configuration A is *action-deterministic* if whenever $A \xrightarrow{\mathsf{tr}} (\mathcal{P};\Phi)$, and P, Q are two elements of $\mathcal{P}$, we have that P and Q cannot perform an observable action of the same nature (`in`, `out`, or `sess`) on the same channel (*i.e.,* if both actions are of same nature, their first argument has to differ).

3 Annotated semantics

We shall now define an intermediate semantics whose transitions are equipped with more informative actions. The annotated actions will notably feature *labels* $\ell \in \mathbb{N}^*$ indicating from which concurrent processes they originate. A *labelled action* will be written $[\alpha]^\ell$ where α is an action and ℓ is a label. Similarly, a *labelled process* will be written $[P]^\ell$. When reasoning about trace equivalence between two configurations, it will be crucial to maintain a consistent labelling between configurations along the execution. In order to do so, we define *skeletons of observable actions*, which are of the form $\mathtt{in}_c$, $\mathtt{out}_c$ or $!^a$ where $a, c \in \mathcal{C}$, and we assume a total ordering over those skeletons, denoted $<$ with $\leq$ being its reflexive closure. Any process that is neither 0 nor a parallel composition induces a skeleton corresponding to its toplevel connective, and we denote it by $\mathsf{sk}(P)$ (*e.g.,* $\mathsf{sk}(\mathtt{in}(c,x).0) = \mathtt{in}_c$).

$$\textsc{In} \quad (\{[\mathtt{in}(c,x).Q]^\ell\} \uplus \mathcal{P}; \Phi) \xrightarrow{[\mathtt{in}(c,M)]^\ell}_a (\{[Q\{M\Phi/x\}]^\ell\} \uplus \mathcal{P}; \Phi) \quad M \in \mathcal{T}(\mathrm{dom}(\Phi))$$

$$\textsc{Out} \quad (\{[\mathtt{out}(c,u).Q]^\ell\} \uplus \mathcal{P}; \Phi) \xrightarrow{[\mathtt{out}(c,w)]^\ell}_a (\{[Q]^\ell\} \uplus \mathcal{P}; \Phi \cup \{w \mapsto u\}) \quad w \in \mathcal{W} \text{ fresh}$$

$$\textsc{Repl} \quad (\{[!^a_{\vec{c},\vec{n}} P_0]^\ell\} \uplus \mathcal{P}; \Phi) \xrightarrow{[\mathtt{sess}(a,\vec{c})]^\ell}_a (\{[P_0]^{\ell\cdot 1}, [!^a_{\vec{c},\vec{n}} P_0]^{\ell\cdot 2}\} \uplus \mathcal{P}; \Phi) \qquad \vec{c}, \vec{n} \text{ fresh}$$

$$\textsc{Par} \quad (\{[\Pi_{i=1}^n P_i]^\ell\} \uplus \mathcal{P}; \Phi) \xrightarrow{[\mathtt{par}(\sigma_{\pi(1)};\ldots;\sigma_{\pi(n)})]^\ell}_a (\{[P_{\pi(1)}]^{\ell\cdot 1}, \ldots, [P_{\pi(n)}]^{\ell\cdot n}\} \uplus \mathcal{P}; \Phi)$$
$$\sigma_i = \mathsf{sk}(P_i) \text{ and } \pi \text{ is a permutation over } [1,...n] \text{ such that } \sigma_{\pi(1)} \leq \ldots \leq \sigma_{\pi(n)}$$

$$\textsc{Zero} \quad (\{[0]^\ell\} \uplus \mathcal{P}; \Phi) \xrightarrow{[\mathtt{zero}]^\ell}_a (\mathcal{P}; \Phi)$$

Figure 1 Annotated semantics.

We then define in Figure 1 the *annotated semantics* $\rightarrow_a$ over configurations whose processes are labelled. In Par, note that $\mathsf{sk}(P_i)$ is well defined as P_i cannot be 0 nor a parallel composition. Note that the annotated transition system does not restrict the executions of a process but simply annotates them with labels, and replaces τ actions by more descriptive actions.

We now define how to extract dependencies from annotated traces, which will allow us to analyse concurrency in an execution without referring to configurations. We obtain sequential dependencies from labels, in a way that is similar, *e.g.,* to the use of *causal relations* in CCS [14]. We also define *recipe dependencies* which are a sort of data dependencies reflecting our specific setting, where we consider an arbitrary attacker who may interact with the process, relying on (maybe several) previously outputted messages to derive input messages.

▶ **Definition 9.** Two labels are *dependent* if one is a prefix of the other. We say that the labelled actions α and β are *sequentially dependent* when their labels are dependent, and *recipe dependent* when $\{\alpha, \beta\} = \{[\mathtt{in}(c, M)]^\ell, [\mathtt{out}(c', w)]^{\ell'}\}$ with w occurring in M. They are *dependent* when they are sequentially or recipe dependent. Otherwise, they are *independent.*

▶ **Definition 10.** A configuration $(\mathcal{P}; \Phi)$ is *well labelled* if $\mathcal{P}$ is a multiset of labelled processes such that two elements of $\mathcal{P}$ have independent labels.

Obviously, any unlabelled configuration may be well labelled. Further, it is easy to see that well labelling is preserved by $\rightarrow_a$. Thus, we shall implicitly assume to be working with well labelled configurations. Under this assumption, we obtain the following lemma.

▶ **Lemma 11.** *Let A be a (well labelled) configuration, α and β be two independent labelled actions. We have $A \xrightarrow{\alpha.\beta}_a A'$ if, and only if, $A \xrightarrow{\beta.\alpha}_a A'$.*

Symmetries of trace equivalence. We will see that, when checking $A \approx B$ for action-deterministic configurations, it is sound to require that B can perform all traces of A in the annotated semantics (and the converse). In other words, labels and detailed non-observable actions $\mathtt{zero}$ and $\mathtt{par}(\sigma_1 \ldots \sigma_n)$ are actually relevant for trace equivalence. Obviously, this can only hold if A and B are labelled consistently. In order to express this, we extend $\mathsf{sk}(P)$ to parallel and 0 processes: we let their skeletons be the associated action in the annotated semantics. Next, we define the labelled skeletons by $\mathsf{skl}([P]^\ell) = [\mathsf{sk}(P)]^\ell$. When checking for equivalence of A and B, we shall assume that $\mathsf{skl}(A) = \mathsf{skl}(B)$, *i.e.,* the configurations have the same set of labelled skeletons. This technical condition is not restrictive in practice.

▶ **Example 12.** Let $A = (\{[\mathtt{in}(a,x).((\mathtt{out}(b,m).P_1) \mid P_2)]^0\}; \Phi)$ with $P_1 = \mathtt{in}(c,y).0$ and $P_2 = \mathtt{in}(d,z).0$, and B the configuration obtained from A by swapping P_1 and P_2. We have $\mathsf{skl}(A) = \mathsf{skl}(B) = \{[\mathtt{in}_a]^0\}$. Consider the following trace:

$$\mathsf{tr} = [\mathtt{in}(a,\mathtt{ok})]^0.[\mathtt{par}(\{\mathtt{out}_b;\mathtt{in}_d\})]^0.[\mathtt{out}(b,w)]^{0\cdot1}.[\mathtt{in}(c,w)]^{0\cdot1}.[\mathtt{in}(d,w)]^{0\cdot2}$$

Assuming $\mathtt{out}_b < \mathtt{in}_d$ and $\mathsf{ok} \in \Sigma$, we have $A \xrightarrow{\mathsf{tr}}_a A'$. However, there is no B' such that $B \xrightarrow{\mathsf{tr}}_a B'$, for two reasons. First, B cannot perform the second action since skeletons of subprocesses of its parallel composition are $\{\mathtt{out}_b;\mathtt{in}_c\}$. Second, B would not be able to perform the action $\mathtt{in}(c,w)$ with the right label. Such mismatches can actually be systematically used to show $A \not\approx B$, as shown next.

▶ **Lemma 13.** *Let A and B be two action-deterministic configurations such that $A \approx B$ and $\mathsf{skl}(A) = \mathsf{skl}(B)$. For any execution $A \xrightarrow{[\alpha_1]^{\ell_1}}_a A_1 \xrightarrow{[\alpha_2]^{\ell_2}}_a A_2 \dots \xrightarrow{[\alpha_n]^{\ell_n}}_a A_n$ with $\mathsf{bc}(\alpha_1.\dots.\alpha_n) \cap \mathsf{fc}(B) = \emptyset$, there exists an execution $B \xrightarrow{[\alpha_1]^{\ell_1}}_a B_1 \xrightarrow{[\alpha_2]^{\ell_2}}_a B_2 \dots \xrightarrow{[\alpha_n]^{\ell_n}}_a B_n$ such that $\Phi(A_i) \sim \Phi(B_i)$ and $\mathsf{skl}(A_i) = \mathsf{skl}(B_i)$ for any $1 \le i \le n$.*

4 Compression

Our first refinement of the semantics, which we call compression, is closely related to focusing from proof theory [2]: we will assign a polarity to processes and constrain the shape of executed traces based on those polarities. This will provide a first significant reduction of the number of traces to consider when checking reachability-based properties such as secrecy, and more importantly, equivalence-based properties in the action-deterministic case. Moreover, compression can easily be used as a replacement for the usual semantics in verification algorithms.

▶ **Definition 14.** A process P is *positive* if it is of the form $\mathtt{in}(c,x).Q$, and it is *negative* otherwise. A multiset of processes $\mathcal{P}$ is *initial* if it contains only positive or *replicated* processes, *i.e.,* of the form $!^a_{\vec{c},\vec{n}}Q$.

The compressed semantics (see Figure 2) is built upon the annotated semantics. It constrains the traces to follow a particular strategy, alternating between *negative* and *positive* phases. It uses enriched configurations of the form $(\mathcal{P}; F; \Phi)$ where $(\mathcal{P}; \Phi)$ is a labelled configuration and F is either a process (signalling which process is *under focus* in the positive phase) or $\varnothing$ (in the negative phase). The negative phase lasts until the configuration is initial (*i.e.,* unfocused with an initial underlying multiset of processes) and in that phase we perform actions that decompose negative non-replicated processes. This is done using the Neg rule, in a completely deterministic way. When the configuration becomes initial, a positive phase can be initiated: we choose one process and start executing the actions of that process (only inputs, possibly preceded by a new session) without the ability to switch to another process of the multiset, until a negative subprocess is released and we go back to the negative phase. The active process in the positive phase is said to be *under focus*. Between any two initial configurations, the compressed semantics executes a sequence of actions, called *blocks*, of the form $\mathtt{foc}(\alpha).\mathsf{tr}^+.\mathtt{rel}.\mathsf{tr}^-$ where tr^+ is a (possibly empty) sequence of input actions, whereas tr^- is a (possibly empty) sequence of $\mathtt{out}$, $\mathtt{par}$, and $\mathtt{zero}$ actions. Note that, except for choosing recipes, the compressed semantics is completely non-branching when executing a block. It may branch only when choosing which block is performed.

$$\textsc{Start/In}\quad \frac{\mathcal{P} \text{ is initial} \quad (P;\Phi) \xrightarrow{\mathtt{in}(c,M)}_a (P';\Phi)}{(\mathcal{P} \uplus \{P\};\varnothing;\Phi) \xrightarrow{\mathtt{foc}(\mathtt{in}(c,M))}_c (\mathcal{P};P';\Phi)}$$

$$\textsc{Start/!}\quad \frac{\mathcal{P} \text{ is initial} \quad (!^a_{\vec{c},\vec{n}}P;\Phi) \xrightarrow{\mathtt{sess}(a,\vec{c})}_a (\{!^a_{\vec{c},\vec{n}}P;Q\};\Phi)}{(\mathcal{P} \uplus \{!^a_{\vec{c},\vec{n}}P\};\varnothing;\Phi) \xrightarrow{\mathtt{foc}(\mathtt{sess}(a,\vec{c}))}_c (\mathcal{P} \uplus \{!^a_{\vec{c},\vec{n}}P\};Q;\Phi)}$$

$$\textsc{Pos/In}\quad \frac{(P;\Phi) \xrightarrow{\mathtt{in}(c,M)}_a (P';\Phi)}{(\mathcal{P};P;\Phi) \xrightarrow{\mathtt{in}(c,M)}_c (\mathcal{P};P';\Phi)}$$

$$\textsc{Neg}\quad \frac{(P;\Phi) \xrightarrow{\alpha}_a (\mathcal{P}';\Phi')}{(\mathcal{P} \uplus \{P\};\varnothing;\Phi) \xrightarrow{\alpha}_c (\mathcal{P} \uplus \mathcal{P}';\varnothing;\Phi')}\ \alpha \in \{\mathtt{par}(\_),\mathtt{zero},\mathtt{out}(\_,\_)\}$$

$$\textsc{Release}\quad (\mathcal{P};[P]^\ell;\Phi) \xrightarrow{[\mathtt{rel}]^\ell}_c (\mathcal{P} \uplus \{[P]^\ell\};\varnothing;\Phi) \quad \text{when } P \text{ is negative}$$

Labels are implicitly set in the same way as in the annotated semantics. Neg is made non-branching by imposing an arbitrary order on labelled skeletons of available actions.

Figure 2 Compressed semantics.

▶ **Example 15.** Consider the process $P = !^a_{c,k}\mathtt{in}(c,x).\mathtt{out}(c,\mathsf{enc}(x,k)).0$. We have that:

$$\begin{array}{lll}(\{P\};\varnothing;\Phi) & \xrightarrow{\mathtt{foc}(\mathtt{sess}(a,c_i))}_c & (\{P\};\{\mathtt{in}(c_i,x).\mathtt{out}(c,\mathsf{enc}(x,k_i)).0\};\Phi)\\ & \xrightarrow{\mathtt{in}(c_i,M_i).\mathtt{rel}}_c & (\{P,\mathtt{out}(c,\mathsf{enc}(M_i\Phi,k_i)).0\};\varnothing;\Phi)\\ & \xrightarrow{\mathtt{out}(c_i,w_i).\mathtt{zero}}_c & (\{P\};\varnothing;\Phi').\end{array}$$

Once a replication is performed, the resulting process is under focus and must be executed in priority until the end. Note that, after executing the input, the resulting process is negative and, thus, still has priority. Thus, on this example, all compressed executions are made of blocks of the form: $\mathtt{sess}(a,c_i).\mathtt{in}(c_i,M_i).\mathtt{out}(c_i.w_i)$.

4.1 Reachability

We now formalise the relationship between traces of the compressed and annotated semantics. In order to do so, we translate between configuration and enriched configuration as follows:

$$\lceil(\mathcal{P};\Phi)\rceil = (\mathcal{P};\varnothing;\Phi),\quad \lfloor(\mathcal{P};\varnothing;\Phi)\rfloor = (\mathcal{P};\Phi)\quad \text{and } \lfloor(\mathcal{P};P;\Phi)\rfloor = (\mathcal{P}\uplus\{P\};\Phi).$$

Similarly, we map compressed traces to annotated ones:

$$\lfloor\epsilon\rfloor = \epsilon,\quad \lfloor\mathtt{foc}(\alpha).\mathsf{tr}\rfloor = \alpha.\lfloor\mathsf{tr}\rfloor,\quad \lfloor\mathtt{rel}.\mathsf{tr}\rfloor = \lfloor\mathsf{tr}\rfloor\quad \text{and } \lfloor\alpha.\mathsf{tr}\rfloor = \alpha.\lfloor\mathsf{tr}\rfloor \text{ otherwise.}$$

We observe that we can map any execution in the compressed semantics to an execution in the annotated semantics. Indeed, a compressed execution is simply an annotated execution with some extra annotations (*i.e.,* `foc` and `rel`) indicating positive/negative phase changes.

▶ **Lemma 16.** *For any configurations A, A' and* tr, $A \xrightarrow{\mathsf{tr}}_c A'$ *implies* $\lfloor A\rfloor \xrightarrow{\lfloor\mathsf{tr}\rfloor}_a \lfloor A'\rfloor$.

Going in the opposite direction is more involved. In general, mapping annotated executions to compressed ones requires to reorder actions. Compressed executions also force negative actions to be performed unconditionally and blocks to be fully executed. One way to handle this is to consider *complete* executions of a configuration, *i.e.,* executions after which no more action can be performed except possibly the ones that consist in unfolding a replication (*i.e.,* rule Repl). Inspired by the positive trunk argument of [19], we show the following lemma.

▶ **Lemma 17.** *Let A, A' be two configurations and* tr *be such that* $A \xrightarrow{\mathsf{tr}}_a A'$ *is complete. There exists a trace* tr_c*, such that* $\lfloor \mathsf{tr}_c \rfloor$ *can be obtained from* tr *by swapping independent labelled actions, and* $\lceil A \rceil \xrightarrow{\mathsf{tr}_c}_c \lceil A' \rceil$.

Proof sketch. We proceed by induction on the length of a complete execution starting from A. If A is not initial, then we need to execute some negative action using Neg: this action must be present somewhere in the complete execution, and we can permute it with preceding actions using Lemma 11. If A is initial, we analyse the prefix of input and session actions and we extract a subsequence of that prefix that corresponds to a full positive phase. ◀

4.2 Equivalence

We now define compressed trace equivalence ($\approx_c$) and prove that it coincides with $\approx$.

▶ **Definition 18.** Let A and B be two configurations. We say that $A \sqsubseteq_c B$ when, for any $A \xrightarrow{\mathsf{tr}}_c A'$ such that $\mathsf{bc}(\mathsf{tr}) \cap \mathsf{fc}(B) = \varnothing$, there exists $B \xrightarrow{\mathsf{tr}}_c B'$ such that $\Phi(A') \sim \Phi(B')$. They are *compressed trace equivalent*, denoted $A \approx_c B$, if $A \sqsubseteq_c B$ and $B \sqsubseteq_c A$.

Compressed trace equivalence can be more efficiently checked than regular trace equivalence. Obviously, it explores fewer interleavings by relying on $\to_c$ rather than $\to$. It also requires that traces of one process can be played exactly by the other, including details such as non-observable actions, labels, and focusing annotations. The subtleties shown in Example 12 are crucial for the completeness of compressed equivalence w.r.t. regular equivalence. Since the compressed semantics forces to perform available outputs before *e.g.* input actions, some non-equivalences are only detected thanks to the labels and detailed non-observable actions of our annotated semantics.

▶ **Theorem 19.** *Let A and B be two action-deterministic configurations with* $\mathsf{skl}(A) = \mathsf{skl}(B)$*. We have $A \approx B$ if, and only if,* $\lceil A \rceil \approx_c \lceil B \rceil$.

Proof sketch. ($\Rightarrow$) Consider an execution $\lceil A \rceil \xrightarrow{\mathsf{tr}}_c A'$. Using Lemma 16, we get $A \xrightarrow{\lfloor \mathsf{tr} \rfloor}_a \lfloor A' \rfloor$. Then, Lemma 13 yields $B \xrightarrow{\lfloor \mathsf{tr} \rfloor}_a B'$ for some B' such that $\Phi(\lfloor A' \rfloor) \sim \Phi(B')$ and labelled skeletons are equal all along the executions. Relying on those skeletons, we show that positive/negative phases are synchronised, and thus $\lceil B \rceil \xrightarrow{\mathsf{tr}}_c B''$ for some B'' with $\lfloor B'' \rfloor = B'$. ($\Leftarrow$) Consider an execution $A \xrightarrow{\mathsf{tr}}_a A'$. We first observe that it suffices to consider only complete executions there. This allows us to get a compressed execution $\lceil A \rceil \xrightarrow{\mathsf{tr}_c}_c \lceil A' \rceil$ by Lemma 17. Since $\lceil A \rceil \approx_c \lceil B \rceil$, there exists B' such that $\lceil B \rceil \xrightarrow{\mathsf{tr}_c}_c B'$ with $\Phi(\lceil A' \rceil) \sim \Phi(B')$. Thus we have $B \xrightarrow{\lfloor \mathsf{tr}_c \rfloor}_a \lfloor B' \rfloor$ but also $B \xrightarrow{\mathsf{tr}}_a \lfloor B' \rfloor$ thanks to Lemma 11. ◀

Improper blocks. Note that blocks of the form $\mathtt{foc}(\alpha).\mathsf{tr}^+.\mathtt{rel.zero}$ do not bring any new information to the attacker. While it would be incorrect to fully ignore such *improper* blocks, it is in fact sufficient to only consider them at the end of traces. We show in [5] that $\approx_c$ coincides with a further optimised compressed trace equivalence that only checks for *proper traces*, *i.e.,* ones that have at most one improper block and only at the end of trace.

5 Reduction

Our compressed semantics cuts down interleavings by using a simple focused strategy. However, this semantics does not analyse data dependency that happen when an input depends on an output, and is thus unable to exploit the independency of blocks to reduce interleavings. We tackle this problem now.

▶ **Definition 20.** Two blocks b_1 and b_2 are *independent*, written $b_1 \parallel b_2$, when all labelled actions $\alpha_1 \in b_1$ and $\alpha_2 \in b_2$ are independent. Otherwise they are *dependent*, written $b_1 \rightleftharpoons b_2$.

Obviously, Lemma 11 tells us that independent blocks can be permuted in a trace without affecting the executability and the result of executing that trace. But this notion is not very strong since it considers fixed recipes, which are irrelevant (in the end, only the derived messages matter) and can easily introduce spurious dependencies. Thus we define a stronger notion of equivalence over traces, which allows permutations of independent blocks but also changes of recipes that preserve messages. During these permutations, we will also require that traces remain *plausible*, which is defined as follows: tr is plausible if for any input $\mathtt{in}(c, M)$ such that $\mathsf{tr} = \mathsf{tr}_0.\mathtt{in}(c, M).\mathsf{tr}_2$ then $M \in \mathcal{T}(\mathcal{W})$ where $\mathcal{W}$ is the set of all handles occurring in tr_0. Given a block b, *i.e.,* a sequence of the form $\mathtt{foc}(\alpha).\mathsf{tr}^+.\mathtt{rel}.\mathsf{tr}^-$, we denote by b^+ (resp. b^-) the part of b corresponding to the positive (resp. negative) phase, *i.e.,* $b^+ = \alpha.\mathsf{tr}^+$ (resp. $b^- = \mathsf{tr}^-$). We note $(b_1 =_{\mathsf{E}} b_2)\Phi$ when $b_1^+\Phi =_{\mathsf{E}} b_2^+\Phi$ and $b_1^- = b_2^-$.

▶ **Definition 21.** Given a frame Φ, the relation $\equiv_\Phi$ is the smallest equivalence over plausible compressed traces such that $\mathsf{tr}.b_1.b_2.\mathsf{tr}' \equiv_\Phi \mathsf{tr}.b_2.b_1.\mathsf{tr}'$ when $b_1 \parallel b_2$, and $\mathsf{tr}.b_1.\mathsf{tr}' \equiv_\Phi \mathsf{tr}.b_2.\mathsf{tr}'$ when $(b_1 =_{\mathsf{E}} b_2)\Phi$.

▶ **Lemma 22.** *Let A and A' be two initial configurations such that $A \xrightarrow{\mathsf{tr}}_c A'$. We have that $A \xrightarrow{\mathsf{tr}'}_c A'$ for any $\mathsf{tr}' \equiv_{\Phi(A')} \mathsf{tr}$.*

We now turn to defining our reduced semantics, which is going to avoid the redundancies identified above by only executing specific representatives in equivalence classes modulo $\equiv_\Phi$. More precisely, we shall only execute minimal traces according to some order, which we now introduce. We assume an order $\prec$ on blocks that is insensitive to recipes, and such that independent blocks are always strictly ordered in one way or the other. We finally define $\prec_{\mathsf{lex}}$ on compressed traces as the lexicographic extension of $\prec$ on blocks.

In order to incrementally build representatives that are minimal with respect to $\prec_{\mathsf{lex}}$, we define a predicate that expresses whether a block b should be *authorised* after a given trace tr. Intuitively, this is the case only when, for any block $b' \succ b$ in tr, dependencies forbid to swap b and b'. We define this with recipe dependencies first, then quantify over all recipes to capture message dependencies.

▶ **Definition 23.** A block b is authorised after tr, noted $\mathsf{tr} \triangleright b$, when $\mathsf{tr} = \epsilon$; or $\mathsf{tr} = \mathsf{tr}_0.b_0$ and either *(i)* $b \rightleftharpoons b_0$ or *(ii)* $b \parallel b_0$, $b_0 \prec b$, and $\mathsf{tr}_0 \triangleright b$.

We finally define $\rightarrow_r$ as the least relation such that:

$$\textsc{Init}\ \frac{}{A \xrightarrow{\epsilon}_r A} \qquad \textsc{Block}\ \frac{A \xrightarrow{\mathsf{tr}}_r (\mathcal{P};\varnothing;\Phi) \quad (\mathcal{P};\varnothing;\Phi) \xrightarrow{b}_c A'}{A \xrightarrow{\mathsf{tr}.b}_r A'}\ \begin{array}{l}\text{if } \mathsf{tr} \triangleright b' \text{ for all } b' \\ \text{with } (b' =_{\mathsf{E}} b)\Phi\end{array}$$

Our reduced semantics only applies to initial configurations: otherwise, no block can be performed. This is not restrictive since we can, without loss of generality, pre-execute non-observable and output actions that may occur at top level.

▶ **Example 24.** We consider roles $R_i := \mathtt{in}(c_i, x).\mathtt{if}\ x = \mathsf{ok}\ \mathtt{then}\ \mathtt{out}(c_i, \mathsf{ok})$ where ok is a public constant, and then consider a parallel composition of n such processes: $P_n := \Pi_{i=1}^n R_i$. Thanks to compression, we will only consider traces made of blocks, and obtain a first exponential reduction of the state space. However, contrary to the case of a replicated process (see Example 15), we still have many interleavings to consider – blocks can be interleaved in all the possible ways. We will see that our reduced semantics cuts down these interleavings.

Assume that our order $\prec$ prioritises blocks on c_i over those on c_j when $i < j$, and consider a trace starting with $\mathtt{in}(c_j, M_j).\mathtt{out}(c_j, w_j)$. Trying to continue the exploration with a block on c_i with $i < j$, the authorisation predicate $\triangleright$ will impose that there is a dependency between the block on c_i and the previous one on c_j. In this case it must be a data dependency: the recipe of the message passed as input on c_i must make use of the previous output to derive ok. Since ok is a public constant, it is possible to derive it without using any previous output and thus the block on c_i cannot be authorised by $\triangleright$. Thus, on this simple example, the reduced semantics will not explore any trace where a block on c_i is performed after one on c_j with $i < j$.

5.1 Reachability

An easy induction on the compressed trace tr allows us to map an execution w.r.t. the reduced semantics to an execution w.r.t. the compressed semantics.

▶ **Lemma 25.** *For any configurations A and A', $A \xrightarrow{\mathsf{tr}}_r A'$ implies $A \xrightarrow{\mathsf{tr}}_c A'$.*

Next, we show that our reduced semantics only explores specific representatives. Given a frame Φ, a plausible trace tr is Φ-minimal if it is minimal in its equivalence class modulo $\equiv_\Phi$.

▶ **Lemma 26.** *Let A be an initial configuration and $A' = (\mathcal{P}; \varnothing; \Phi)$ be a configuration such that $A \xrightarrow{\mathsf{tr}}_c A'$. We have that tr is Φ-minimal if, and only if, $A \xrightarrow{\mathsf{tr}}_r A'$.*

Proof sketch. In order to relate minimality and executability in the reduced semantics, let us say that a trace is *bad* if it is of the form $\mathsf{tr}.b_0 \dots b_n.b'.\mathsf{tr}'$ where $n \geq 0$, there exists a block b'' such that $(b'' =_{\mathsf{E}} b')\Phi$, we have $b_i \parallel b''$ for all i, and $b_i \prec b'' \prec b_0$ for all $i > 0$. This pattern is directly inspired by the characterisation of lexicographic normal forms by Anisimov and Knuth in trace monoids [3]. We note that a trace that can be executed in the compressed semantics can also be executed in the reduced semantics if, and only if, it is not bad. Since the badness of a trace allows to swap b' before b_0, and thus obtain a smaller trace in the class $\equiv_\Phi$, we show that a bad trace cannot be Φ-minimal (and conversely). ◀

5.2 Equivalence

The reduced semantics induces an equivalence $\approx_r$ that we define similarly to the compressed one, and we then establish its soundness and completeness w.r.t. $\approx_c$.

▶ **Definition 27.** Let A and B be two configurations. We say that $A \sqsubseteq_r B$ when, for every $A \xrightarrow{\mathsf{tr}}_r A'$ such that $\mathsf{bc}(\mathsf{tr}) \cap \mathsf{fc}(B) = \varnothing$, there exists $B \xrightarrow{\mathsf{tr}}_r B'$ such that $\Phi(A') \sim \Phi(B')$. They are *reduced trace equivalent*, denoted $A \approx_r B$, if $A \sqsubseteq_r B$ and $B \sqsubseteq_r A$.

▶ **Theorem 28.** *Let A and B be two initial, action-deterministic configurations.*

$$A \approx_c B \text{ if, and only if, } A \approx_r B$$

Proof sketch. We first prove that $\mathsf{tr} \equiv_\Phi \mathsf{tr}'$ iff $\mathsf{tr} \equiv_\Psi \mathsf{tr}'$ when $\Phi \sim \Psi$. ($\Rightarrow$) This implication is then an easy consequence of Lemma 26. ($\Leftarrow$) We start by showing that it suffices to consider a complete execution $A \xrightarrow{\mathsf{tr}}_c A'$. Since A' is initial, by taking tr_m to be a $\Phi(A')$-minimal trace associated to tr, we obtain a reduced execution of A leading to A'. Using our hypothesis $A \approx_r B$, we obtain that $B \xrightarrow{\mathsf{tr}_m}_r B'$ with corresponding relations over frames. We finally conclude that $B \xrightarrow{\mathsf{tr}}_c B'$ using Lemma 22 and the result stated above. ◀

Improper blocks. Similarly as we did for the compressed semantics in Section 4, we can further restrict $\approx_r$ to only check proper traces.

6 Application

We have developed two successive refinements of the concrete semantics of our process algebra, eventually obtaining a reduced semantics that achieves an optimal elimination of redundant interleavings. However, the practical usability of these semantics in algorithms for checking the equivalence of replication-free processes is far from immediate: indeed, all of our semantics are still infinitely branching, because each input may be fed with arbitrary messages. We now discuss how existing decision procedures based on symbolic execution [18, 12, 23, 9] can be modified to decide our optimised equivalences rather than the regular one, before presenting our implementation and experimental results.

6.1 Symbolic execution

Our compressed semantics can easily be used as a replacement of the regular one, in any tool whose algorithm is based on a forward exploration of the set of possible traces. This modification is very lightweight, and already brings a significant optimisation. In order to make use of our final, reduced semantics, we would need to enter into the details of constraint solving. In addition to imposing the compressed strategy and the sequential dependencies imposed by our predicate $\mathsf{tr} \triangleright b$, symbolic execution should be modified to generate *dependency constraints* in order to reflect the data dependencies imposed by $\mathsf{tr} \triangleright b$. The generation of dependency constraints can be done in a similar way to [4]. The constraint solver is then modified in a non-invasive way: dependency constraints are used to dismiss configurations when it becomes obvious that they cannot be satisfied.

▶ **Example 29.** We consider the symbolic reduced executions of process P_n from Example 24. In symbolic executions, input recipes and messages are initially left unknown, and gradually discovered by constraint resolution procedures. Assume that we have already executed a block on c_j. After that, we can execute symbolically an input on c_i, with $i < j$: let us write it $\mathtt{in}(c_i, X_i)$. Because we use the reduced semantics, a dependency constraint $\mathsf{dep}(X_i, w_j)$ is generated, expressing that the recipe denoted by X_i must depend on w_j. After executing its input, process R_i makes a test $(x = \mathsf{ok})$. Its `else` branch is trivial: it leads to an improper block, allowing us to stop any further exploration. When taking the `then` branch, we add a constraint expressing that recipe X_i must derive the message ok. In a tool such as Apte, this constraint is immediately solved by instantiating $X_i := \mathsf{ok}$ (considering other ways to derive ok is useless). After this instantiation, our dependency constraint has become $\mathsf{dep}(\mathsf{ok}, w_j)$ which is obviously unsatisfiable, and thus the branch is discarded.

The modified verification algorithm may explore symbolic traces that do not correspond to Φ-minimal representatives (when dependency constraints cannot be shown to be infeasible) but we will see that this approach allows us to obtain a very effective optimisation. Finally, note that, because we may over-approximate dependency constraints, we must ensure that constraint resolution prunes executions in a symmetrical fashion for both processes being checked for equivalence.

6.2 Experimental results

The optimisations developed in the present paper have been implemented, following the above approach, in the official version of the state of the art tool Apte [10]. We now report on experimental results; sources and instructions for reproduction are available [17]. We only show examples in which equivalence holds, because the time spent on inequivalent processes is too sensitive to the order in which the (depth-first) exploration is performed.

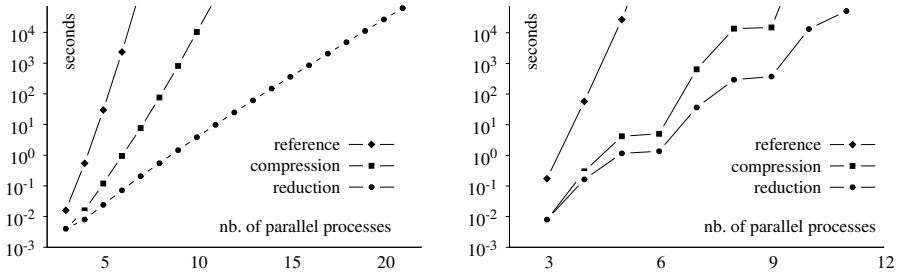


Figure 3 Impact of optimisations on toy example (left) and Denning-Sacco (right).

Toy example. We consider again our simple example described in Section 6.1. We ran Apte on $P_n \approx P_n$ for $n = 1$ to 22, on a single 2.67GHz Xeon core (memory is not relevant). We performed our tests on the reference version and the versions optimised with the compressed and reduced semantics respectively. The results are shown on the left graph of Figure 3, in logarithmic scale: it confirms that each optimisation brings an exponential speedup, as predicted by our theoretical analysis.

Denning-Sacco protocol. We ran a similar benchmark, checking that Denning-Sacco ensures strong secrecy in various scenarios. The protocol has three roles and we added processes playing those roles in turn, starting with three processes in parallel. The results are plotted on Figure 3. The fact that we add one role out of three at each step explains the irregular growth in verification time. We still observe an exponential speedup for each optimisation.

Practical impact. Finally, we illustrate how our optimisations make Apte much more useful in practice for investigating interesting scenarios. Verifying a single session of a protocol brings little assurance into its security. In order to detect replay attacks and to allow the attacker to compare messages that are exchanged, at least two sessions should be considered. This means having at least four parallel processes for two-party protocols, and six when a trusted third party is involved. This is actually beyond what the unoptimised Apte can handle in a reasonable amount of time. We show below how many parallel processes could be handled in 20 hours by the different versions of Apte on various use cases of protocols.

Protocol	ref	comp	red	Protocol	ref	comp	red
Needham Schroeder (3-party)	4	6	7	Denning-Sacco (3-party)	5	9	10
Private Authent. (2-party)	4	7	7	WMF (3-party)	6	12	13
Yahalom (3-party)	4	5	5	E-Passport PA (2-party)	4	7	9

7 Conclusion

We have developed two POR techniques that are adequate for verifying reachability and trace equivalence properties of action-deterministic security protocols. We have effectively implemented them in Apte, and shown that they yield the expected, significant benefit.

We are considering several directions for future work. Regarding the theoretical results presented here, the main question is whether we can get rid of the action-determinism condition without degrading our reductions too much. Regarding the practical application of our results, we can certainly go further. We first note that our compression technique should be applicable and useful in other verification tools, not necessarily based on symbolic execution. Next, we could investigate the role of the particular choice of the order $\prec$, to

determine heuristics for maximising the practical impact of reduction. Finally, we plan to adapt our treatment of replication to bounded replication to obtain a first symmetry elimination scheme, which should provide a significant optimisation when studying security protocols with several sessions.

References

1 M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of POPL'01*. ACM Press, 2001.
2 J.-M. Andreoli. Logic programming with focusing proofs in linear logic. *J. Log. Comput.*, 2(3), 1992.
3 A.V. Anisimov and D.E. Knuth. Inhomogeneous sorting. *International Journal of Computer & Information Sciences*, 8(4):255–260, 1979.
4 D. Baelde, S. Delaune, and L. Hirschi. A reduced semantics for deciding trace equivalence using constraint systems. In *Proc. of POST'14*. Springer, 2014.
5 David Baelde, Stéphanie Delaune, and Lucca Hirschi. Partial order reduction for security protocols. *CoRR*, abs/1504.04768, 2015.
6 C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
7 Mayla Bruso, K. Chatzikokolakis, and J. den Hartog. Formal verification of privacy for RFID systems. In *Proc. of CSF'10*, 2010.
8 V. Cheval. Apte: an algorithm for proving trace equivalence. In *Proc. TACAS'14*, 2014.
9 V. Cheval, H. Comon-Lundh, and S. Delaune. Trace equivalence decision: Negative tests and non-determinism. In *Proc. of CCS'11*. ACM Press, 2011.
10 V. Cheval and L. Hirschi. sources of APTE, 2015. `https://github.com/APTE/APTE`.
11 E. Clarke, S. Jha, and W. Marrero. Efficient verification of security protocols using partial-order reductions. *Int. Journal on Software Tools for Technology Transfer*, 4(2), 2003.
12 H. Comon-Lundh, V. Cortier, and E. Zalinescu. Deciding security properties for cryptographic protocols. Application to key cycles. *ACM Transactions on Computational Logic (TOCL)*, 11(4), 2010.
13 Cas JF Cremers and Sjouke Mauw. Checking secrecy by means of partial order reduction. In *System Analysis and Modeling*. Springer, 2005.
14 Pierpaolo Degano, Rocco De Nicola, and Ugo Montanari. A partial ordering semantics for ccs. *Theoretical Computer Science*, 75(3):223–262, 1990.
15 S. Delaune, S. Kremer, and M. Ryan. Verifying privacy-type properties of electronic voting protocols: A taster. In *Towards Trustworthy Elections – New Directions in Electronic Voting*, volume 6000. Springer, 2010.
16 Patrice Godefroid. *Partial-Order Methods for the Verification of Concurrent Systems – An Approach to the State-Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, 1996.
17 L. Hirschi. APTE with POR. `http://www.lsv.ens-cachan.fr/~hirschi/apte_por`.
18 J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of CCS'01*. ACM Press, 2001.
19 D. Miller and A. Saurin. From proofs to focused proofs: A modular proof of focalization in linear logic. In *Proc. of CSL'07*, volume 4646. Springer, 2007.
20 S. Mödersheim, L. Viganò, and D. Basin. Constraint differentiation: Search-space reduction for the constraint-based analysis of security protocols. *JCS*, 18(4), 2010.
21 D. Peled. Ten years of partial order reduction. In *Proc. of CAV'98*. Springer, 1998.
22 A. Tiu. Spec: `http://users.cecs.anu.edu.au/~tiu/spec/`, 2010.
23 A. Tiu and J. E. Dawson. Automating open bisimulation checking for the spi calculus. In *Proc. of CSF'10*. IEEE Comp. Soc. Press, 2010.